UX-RULE: A JAVASCRIPT FRAMEWORK FOR DYNAMIC AND STATIC ACCESSIBILITY TESTING OF WEBPAGES

By

ERIN S. DUGGAN

B.S., New Mexico Institute of Mining and Technology, 2013

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Masters’ of Science

Department of Computer Science

2014
This thesis entitled: 
ux-rule: A JavaScript Framework for Dynamic and Static Accessibility Testing of Webpages 
written by Erin S. Duggan 
has been approved for the Department of Computer Science

__________________________________________
Tom Yeh

__________________________________________
Shaun Kane

Date ____________

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.
Duggan, Erin S. (M.S., Computer Science)

*ux-rule*: A JavaScript Framework for Dynamic and Static Accessibility Testing of Webpages

Thesis directed by Professors Tom Yeh and Shaun Kane

The Internet is designed to disseminate information to the widest group of people possible. In recent years, this group has been expanded to include people with disabilities through the use of a number of tools. In order to support this, websites need to be designed as accessible and require testing to ensure that they are accessible. The *ux-rule* framework is an automated accessibility testing framework designed for use by website programmers to do both static and dynamic analysis. It establishes a rule syntax for specifying tests that is based on imperative programming language syntax so that rules can be added or modified easily to meet any other requirements in addition to accessibility guidelines. The resulting package is open source and free to reach the largest number of potential users outside of commercial environments. By meeting these goals it fills in some of the limitations of other existing tools, primarily cost, custom rule specification, and dynamic testing.
This thesis is dedicated to my mother, Ruth Duggan, without whom, I wouldn’t have made it this far. Even though she didn’t live to see my thesis work completed, she never had any doubts that I would finish and her support has pushed me forward continually.
I would like to take this opportunity to thank those who supported me in accomplishing this work. Particularly I would like to offer my thanks to my advisors, Professor Tom Yeh and Professor Shaun Kane. I appreciate their willingness to work with me even under my constraints and their tireless efforts in helping me succeed, offering advice and direction when I needed it.

I would also like to offer my sincerest gratitude to Sandia National Laboratories for funding me through their Master’s Fellowship Program. Without their support I would never have had the opportunity to expand my horizons so much and do this research. To my manager, Lilly, and all the folks in the O&M Sustainment Organization, I am eternally grateful for your endless support and your help getting me to this point.

Thank you all very much,

Erin S. Duggan
CONTENTS

CHAPTER

I. INTRODUCTION.........................................................................................1

1.1 Purpose of the Project.................................................................1

1.2 Scope of the Project.................................................................2

1.2.1 Rule Completeness.........................................................3

1.3 Goals of the ux-rule Framework........................................3

1.4 Arrangement of the Thesis.....................................................4

II. REVIEW OF EXISTING TOOLS.............................................................5

2.1 Survey of Existing Accessibility Testing Tools................5

2.1.1 Browser Plugins..........................................................6

2.1.2 Online Tools.................................................................7

2.1.3 Command Line Tools.....................................................8

2.1.4 Desktop Applications....................................................9

2.2 Limitations of Existing Tools..............................................10

2.2.1 Browser Plugins..........................................................10

2.2.2 Online Tools.................................................................10

2.2.3 Command Line Tools.....................................................11

2.2.4 Desktop Applications....................................................11

2.2.5 General Limitations..........................................................12

III. UX-RULE SYNTAX AND SEMANTICS.................................................15

3.1 Design Requirements.............................................................15
3.2 Rule Basics

3.2.1 Construction

3.2.1.1 Objects

3.2.1.2 Object Relationships

3.2.1.3 Grouping Syntax, Ranges, and Scoping

3.2.1.4 Quantifiers

3.2.1.5 Operations

3.2.1.6 Conditional Statements

3.2.1.7 Body Syntax

3.3 Rule Construction Examples

3.3.1 Each Input Has Label

3.3.2 Button Text is Consistent

3.4 Current Parsing Capabilities

IV. UX-RULE IMPLEMENTATION

4.1 Retrieved HTML vs. Rendered HTML

4.2 System Outline

4.2.1 Architecture

4.2.2 Certification Estimation Setup

4.3 Using ux-rule

4.4 Focus Areas

4.4.1 Content Testing

4.4.1.1 Image Alt Text Testing

Alt Text Exists
Alt Text Meets RegEx Condition........39
Alt Text is Consistent..................40
Alt Text is Within Word Limit.........42

4.4.1.2 Webpage Language Testing........44
Language for Entire Page.................44
Some Sections Have Language............45
All Sections Have Language..............46

4.4.1.3 Screen Reader Support............47
Skip to Main Content Link Exists........48

4.4.2 Site Structure Testing..................49

4.4.2.1 External Structure...............50
No Links go to 404 Page..................50
Limit on Number of Redirect Links.......52
Check for Certificate Issues.............53

4.4.2.2 Internal Structure...............55
Navigation Menu Consistency.............55

4.4.3 Input Testing..........................57

4.4.3.1 Embedded Forms..................58
ARIA Compliance on Forms...............58

4.5 Testing Procedure......................60

4.6 Results..................................61

4.6.1 Alt Text Group........................62

4.6.2 Language Group........................63
4.6.3 External Structure Group.................................64
4.6.4 Miscellaneous Group..................................65
4.6.5 Full Coverage Group..................................65
4.6.6 Additional Sites........................................66
4.7 Limitations of *ux-rule*................................68
4.8 Conclusions..............................................69
4.9 Future Work..............................................70

BIBLIOGRAPHY.....................................................73

APPENDIX

A. RULE CONSTRUCTIONS TUTORIAL..........................79
B. RULES AND JSON BREAKDOWN............................85
C. *UX-RULE CODE*...........................................88
D. TEST SITES AND OUTPUT RESULTS.....................109
TABLES

Table

1. Summary of Automated Testing Tools…………………………………………..9
2. Truth Table for the IF-THEN Statement……………………………………..22
3. Truth Table for the IF-THEN-ELSE Statement…………………………..22
4. Ground Truth Chart for Testing Sites…………………………………..60
5. Alt Text Group Results……………………………………………………62
6. Language Group Results…………………………………………………63
7. External Structure Group Results…………………………………………64
8. Miscellaneous Group Results………………………………………………65
9. Full Coverage Group Results………………………………………………66
10. Website to Letter Code Map………………………………………………67
11. Additional Sites Results……………………………………………………67
FIGURES

Figure

1. *ux-rule* architectural diagram..................................................12
2. Sample code for calling *ux-rule* in JavaScript.................................34
3. Command line instructions to run the JavaScript code.........................35
4. Terminal output of running code in Figures 2 and 3.............................35
CHAPTER I

INTRODUCTION

Automated testing has become an invaluable operation that computers can perform. It has its limitations, but can be used to greatly reduce the amount of work a human needs to perform and the occasions of human error as a result. It has played its part in security testing, and is now making its way into the area of accessibility. Over the last few years, accessibility has become a big issue with webpages as there exist more and more tools designed to help people with disabilities use the Internet. The World Wide Web Consortium, or W3C, has established some guidelines that are designed to make websites available to the widest range of people if followed [54]. Many of the existing tools use these guidelines as the basis for their accessibility testing, as does the ux-rule framework.

1.1 Purpose of the Project

While there exist a number of tools out there that look at the HTML of a webpage and run them against a standard set of accessibility standards, few provide a framework that allows the tester to specify their own rules in addition to providing a standard set
[4][9]. However, there may be company policies regarding the requirements on webpage design, which the standard set may not cover. Additionally, the typical accessibility testing tools only scrape the retrieved HTML code of a site, which eliminates any dynamic testing, defined in this thesis as testing any scripted or changeable elements of a webpage including server-related [9].

This project strives to provide such a framework, making it easy for the website programmers to design and specify new rules as needed. In order to help facilitate this goal, the framework is meant to be an open-source npm package, such that anyone can use it and make modifications to the code to support their own testing requirements. The framework also supports dynamic testing, which allows for more thorough and rigorous checks against requirements. Constructing the framework such that the tests can be executed efficiently is also important, so it can provide results quickly to testers. It is meant to perform the tests quicker than a human, so it would not serve its purpose by taking too long to execute. By accomplishing the construction of this framework, according to specifications, this project attempts to fill the gaps left where other accessibility testers leave off.

1.2 Scope of the Project

A primary goal of this project is to provide a proof of concept in regards to the framework. As such, the scope of this project is limited to a small number of rules that show a wide range of rule design and capability.
1.2.1 Rule Completeness

The set of rules used in this project are enough to demonstrate proof of concept, but are not the full coverage of all the Web Content Accessibility Guidelines. The focus instead is on a few small categories within those guidelines, as well as some rules outside of them. These focus areas: content testing, site structure testing, and input testing, contain within them a range of rules, as well as a range between the areas themselves.

1.3 Goals of the ux-rule Framework

Having examined some of the major limitations found in other tools, as expanded upon in Chapter 2, ux-rule seeks to fill in a few of the trouble areas. A primary goal is to provide a tool framework with programmers in mind, both as the users of the tool and the receivers of the end report. It presents a separated design, with the rule specification and logic outside of the execution of the back end, in order to be flexible enough to handle rules that aren’t directly part of guidelines as well as multi-layered, more complex rules. The syntax for the rule language is not based on XML, although that seems to be the language of choice in the other tools allowing custom rules, in order to make the rules more directly understandable by a wider range of programmers and other users. The open-source nature of the framework also means that it can be community supported to keep it up to date as well as maintain an ever-expanding set of written out rules. The support for custom rules also makes it easier to integrate into a design process, since it can be used to test a number of properties of a webpage beyond accessibility.
1.4 Arrangement of the Thesis

This thesis is organized as follows: first, a survey of existing accessibility testing tools and their limitations, followed by the description and specifications of the \textit{ux-rule} rule construction language, and a chapter on the framework itself. That section is further broken down into the explanation of the architecture, followed by an in-depth look at the rules within the chosen focus areas and their current implementations in the back end of the framework. After that are the testing procedures and results, ending with the conclusions and future work, which elaborate on the contributions of the framework and the future directions it can take. At the end of the document is the bibliography and appendices containing a rule construction tutorial, the rules and their JSON breakdowns used in this thesis, the full code of \textit{ux-rule}, and the constructed test websites [62].
CHAPTER II

REVIEW OF EXISTING TOOLS

The W3C Web Accessibility Initiative, the organization that puts out the Web Content Accessibility Guidelines, hereafter referred to as the WCAG, maintains a database of accessibility evaluation tools, with information provided by developers and vendors. They do not endorse any of the tools, but as a resource for locating a range of tools out there, it is invaluable [66]. Since it also provides a way to limit the searches by the type of guidelines they test for, the resulting list can be trimmed to contain only those tools testing under the same guidelines as *ux-rule*. The sections below go over some of the tools from the above database, with greater detail on some of the most similar tools to the *ux-rule* framework, and provide a discussion of the general limitations.

2.1 Survey of Existing Accessibility Testing Tools

This section looks at the various types of tools that test for the WCAG guidelines. There are a number of different types of tools, both web-based and desktop applications, and range from proprietary to open-source. Some have very small, focused areas they test on, and others have a large range of capabilities. The tools are divided into groups
based on their type: browser plugins, online tools, command line tools, and desktop applications.

2.1.1 Browser Plugins

There are a number of tools that provide their services through a browser plugin, at least in some part. Among these are some basic checkers that check the structure of HTML code, such as HeadingsMap [39], and some that just focus on the color contrast rules, like Contrast Checker [32]. The Accessibility Developer Tools Plugin for Chrome runs audits against a number of rules and provides an interface that allows the developer to see the webpage code, view of the page itself, the reports of the tests, and a way to easily see the attribute breakdowns and values. It is also provides a JavaScript API as well as being open-source [21]. A similar tool for Firefox and Internet Explorer 9+, FireEyes, interacts with the Firebug debugging plugin and can simulate a screen reader among its tests. The reports it gives can be seen in the browser or downloaded as a CSV file [37]. Acknowledging that there are some things that cannot be tested for by a computer, HERA-FFX is a Spanish tool for Firefox that does preliminary checks geared towards manual review [40][7]. HTML_CodeSniffer is a JavaScript “bookmarklet” that looks at HTML page source code and permits the definition of custom coding standards. It claims to have standards that enforce the 3 conformance levels of WCAG [42]. A tool that makes use of browser plugins is MAUVE, which is a web-based tool that uses the plugins to check dynamic content. This tool also has its own rule language, based on XML [15][45]. The Total Validator strives to provide a wide range of checks, from the
basic HTML structure to spelling, to accessibility checking. It can run as a desktop application, through the command line, or as a plugin. It is free software, but not open source [58]. The last of the plugin tools is WAVE, which is an online service run by WebAIM. It generates report of accessibility issues and is a free service/plugin, but it is not open source and does not allow for the addition of new rules [64].

2.1.2 Online Tools

Some of the browser plugins also have an online version of the tool, specifically, HTML_CodeSniffer, MAUVE, and WAVE, which will not be mentioned further in this section. In this category are again the basic single function tools, such as the Accessibility Color Wheel [20] and Contrast-Finder [33]. A number of these tools take a site and generate a report based on their sets of tests for the guidelines. These tools are the A11Y Compliance Platform [19], AccessMonitor (a Portuguese Public Administration run site) [26], DaSilva (a Brazilian site) [35], eXaminator (a Western Samoa site) [36], Functional Accessibility Evaluator [38], Tingtun Page Checker [56], and Vamolå (an Italian version of AChecker without the custom rules) [63]. While these tools are free to use, there are other tools that require a subscription or a login account to use features or the entire tool. Many of these also provide training services as well. Among these are AChecker [27][8], which requires an account to specify custom tests, Accessibility Management Platform [22], AccessIn [23], AccessLint [25], HiSoftware Compliance Sherriff Web (which does security compliance as well as accessibility) [41], Siteimprove Accessibility [50], Sitemorse [51], SortSite [52], Tanaguru [53], and
WorldSpace (which uses the FireEyes plugin) [68]. The Tenon tool is an API designed for integration into whatever testing framework is used covering a wide number of tests and environments such as Node and Python. The code is open-source, but there is a subscription fee that is based on the number of API calls made per month. The web service performs testing on a provided site or code snippet, but the individual tests cannot be specified without using the API [54]. The WCAG Compliance Auditor is commercial software that runs the tests and also provides suggestions on fixing them. The big difference is that they also establish benchmarks to check progress against [65].

2.1.3 Command Line Tools

Command line tools have somewhat of a stigma around their use, often considered difficult or complicated to use. There seem to be much fewer of these in the database, either because their programmers wrote them for their own use or they are a harder sell for the average computer user. AccessLint Rubygem runs on the command line or in Ruby using both PhantomJS and Google’s Accessibility Developer Tools JavaScript API to look at the HTML code and make assertions about it. It is an open source and free tool. It also makes its reports as JSON objects [24]. TestPage is a similar tool for Windows that checks a page against the W3C standards, as well as some accessibility standards. The tests it offers are simple and basic, geared towards passing the information off to programmers [55]. Both of the tools mentioned here only do static analysis of the code, looking exclusively at the HTML code of the webpage.
2.1.4 Desktop Applications

Like the command line tools, there are few desktop applications in the database.

A number of the commercial online tools also claim to have desktop applications as well, as do a few of the browser plugin types, but generally require registration of such to access. COMPLYFirst Professional is really the only tool that is specifically only a desktop application. It supports testing the code for various standards, but does not appear to allow custom rules to be designed and applied. It is also commercial software, requiring licensing to use, designed for a corporate environment [31].

<table>
<thead>
<tr>
<th>Tools</th>
<th>Browser/Plugin</th>
<th>Online Command Line</th>
<th>Desktop Application</th>
<th>Certification Reports</th>
<th>Custom Rules</th>
<th>Registration Required</th>
<th>License</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11Y Compliance Platform</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>C, E</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>Accessibility Color Wheel</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility Developer Tools</td>
<td>C</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility Management Platform</td>
<td>Ff, C, IE</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>C, E</td>
<td>$5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccessIn</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>C</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>AccessLint Rubygen</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccessLint.com</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AccessMonitor</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Achecker</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>FS, OS</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>COMPLYFirst Professional</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>C, E</td>
<td>$5</td>
<td></td>
</tr>
<tr>
<td>Contrast Checker</td>
<td>Ff, C</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contrast-Finder</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>FS, OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DaSilva</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eXaminator</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FireEyes</td>
<td>Ff, IE&gt;9</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS, C, E</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional Accessibility Evaluator</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HeadingsMap</td>
<td>Ff, C</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HERA-FFX</td>
<td>Ff</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS, OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HiSoftware Compliance Sheriff Web</td>
<td>Ff, IE&gt;9</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>C, E</td>
<td>trial or $5</td>
<td></td>
</tr>
<tr>
<td>HTML Code Sniffer</td>
<td>Ff, C, IE, S</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>OS, C, E</td>
<td>$5</td>
<td></td>
</tr>
<tr>
<td>MAUVE</td>
<td>Ff, C</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>FS</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>Siteimprove Accessibility</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>FS</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>SiteImprove</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>C</td>
<td>$5</td>
<td></td>
</tr>
<tr>
<td>SortSite</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>C</td>
<td>trial or $5</td>
<td></td>
</tr>
<tr>
<td>Tantagura</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tenon</td>
<td>C</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>OS, C, E</td>
<td>$5</td>
<td></td>
</tr>
<tr>
<td>TestPage</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>OS</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>Tintung Accessibility Checker</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>OS</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>Total Validator</td>
<td>Ff, C</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>FS, C</td>
<td>free use</td>
<td></td>
</tr>
<tr>
<td>Vamlala</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS, OS</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAVE</td>
<td>Ff, C</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>FS, C</td>
<td>free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCAG Compliance Auditor</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>C</td>
<td>$5</td>
<td></td>
</tr>
<tr>
<td>WorldSpace</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>C</td>
<td>$5</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Summary of automated testing tools showing their type, their capabilities, and licensing. Browser plugins are Ff: FireFox, C: Chrome, IE: Internet Explorer. Licenses are C: commercial, E: enterprise, OS: open source, FS: free software.
2.2 Limitations of Existing Tools

Due to the nature of the accessibility test, there are some limitations of the tools that exist across the board. There are also some limitations that stem from the type of tool. This section presents the limitations of the individual types and ends with a discussion of the overall limitations.

2.2.1 Browser Plugins

The biggest limitation of the browser plugins is finding one that can run in any browser and still have the functionality that is required. Many of the tools run as Firefox or Chrome plugins, but otherwise there isn’t much support for other browsers. Other tools may require the plugins to operate, so any changes in the implementation for the browser updates ripple out into various other tools. There are quite a large number of tools that fall under this category, but they are only as useful as how easily using them can be integrated into a development process.

2.2.2 Online Tools

The main limitations here are that it is difficult to specify custom rules and tests, as well as maintaining the privacy of a site run through testing. They also require the testing machine to be online, and can’t be reached on a closed network. Another limitation is that many of the more extensive testing tools are subscription based or
commercial software. This contributes to the lack of modifiable tests, since the code cannot be seen or reverse-engineered. The pricing also makes it difficult for small businesses or individuals working on their sites to use the tools.

2.2.3 Command Line Tools

Using these tools requires some level of background in computing and are often difficult to jump directly into as a person unfamiliar with the command line. They also can suffer from the portability problems of desktop applications depending on which type of command line they are designed to run in or what other programs they need to use. Most of these tools are likely to be UNIX based, and won’t run in Windows unless they can sit in another environment that does run on Windows.

2.2.4 Desktop Applications

To make a desktop application effective, it needs to work across different platforms. As the number of mobile computing devices increases as well, having the same tools available on those platforms too may be a necessary step. The applications also are generally commercial software, which makes it difficult to support adding rules and affordability for non-corporate users.
2.2.5 General Limitations

At this point in time, an automated accessibility tester can only check things up to a certain point. It is difficult for computers to establish meaning in any sort of text, which forms the core of several guidelines. It is more efficient to direct a human where to look and let them make a determination. It is also difficult for computers to see images the same way humans do. There are image-processing techniques to recognize things within images, but they are still not very efficient. Computers can recognize text in images, but it is not trivial to determine if that text is important [9][4].

Forms are another source of limitations. The labels can be checked for existence, but like the image alt text, cannot be automatically checked for meaning or usefulness. The errors they generate as well have the same issues. To determine meaningful error messages is subjective even among humans, making it a far more complicated for computers to handle alone. Since the errors are typically executed from a script, and scripts and other “rich code” are difficult to analyze, it limits the checks that can be done [9].

The so-called “rich code” is part of what makes a webpage dynamic. The increasing number of web applications that load in content with JavaScript poses a significant problem for accessibility checkers [5]. These checkers only look at a static representation of the code and do not account for any changes in the DOM that occur as a result of user action or scripts within the page. Checking the full extent of this content is difficult because not only must all the actions be accounted for and their changes to the page state determined, but the results of a dynamic page may not be the same at different access times.
There are also some limitations that come from what the tools are checking. Many tools are limited by the guidelines that they check for, focusing on the accessibility certification and passing test on all those guidelines. This makes it difficult for them to test for specific subsets of the guidelines that pertain to certain disabilities and checking that the requirements for those groups are met. This focus on guidelines may also blind developers of the tools to the needs of the actual end users and how they actually use the webpages [6][17].

The way the tools build their guidelines specifications can also pose issues. Tools that do not separate their rule/ guideline specification logic from the execution part of tool can be difficult to keep up to date with changing requirements and less flexible in specifying rules [16][2][15]. There have been a few tools that meet that separation, but the language in which they specify the rules is an expanded version of XML [16][ee][15], which makes rule construction complicated if programming in XML is unfamiliar. Also, it is not immediately apparent what the rule is about. An inflexible rule system limits the use of the tool and might prevent custom rules such as “when form input is entered, or a skip link followed, the page should not refresh”, which is not a guideline, but would prevent some of the worst frustrations for screen reader users [17].

The issue of integrating a tool into design process is a limitation that extends beyond just the browser plugins. There are a number of factors that contribute to this problem, some of which are out of range of the tool itself. As with any project or undertaking, it is easier and cheaper to work elements in from the beginning, instead of putting them in later in the design or development process. This applies to working accessibility into the design of webpages as well, but it isn’t something taught to most
programmers in their basic schooling. Add in the fact that accessibility flies below the
radar of those in charge or the customers, and the accessibility features aren’t likely to get
much consideration [11][10].

A factor that is within the control of the tool to a certain extent is the reporting
mechanism of the tool. The results should make sense not only to the programmers, but
the managerial team as well, which is an area that many tools fall short in, producing
instead verbose reports geared towards accessibility experts [10]. Since the people
responsible for making the changes suggested by the tool are the programmers, the tools
should report results that are meaningful for them in as simple a way as possible.
CHAPTER III

UX-RULE SYNTAX AND SEMANTICS

3.1 Design Requirements

A primary goal of the ux-rule framework is the ability to specify new rules beyond the initial set provided with the tool. In order to accomplish that, the language for describing the rules must meet certain design requirements. The framework is designed for use by programmers, so the language should be in a form they can understand and work with easily. The purpose of the rules should also be reasonably easy to interpret by non-programmers. Additionally, the rule language should be flexible enough to handle both simple rules and complex rules.

3.2 Rule Basics

The rule language is based on imperative programming languages, in a pseudo-code form. The elements of the rules are taken from HTML, both tags and attributes. However, to better support construction of custom rules, additional properties can be generated and assigned to either existing tags or attributes. These property names are the only language elements that can be defined by the user, similar to variables. Rules are built with at least 2 aspects, a quantifier and an object or object set the rule is applied to,
referred to as the *setup block* in this thesis. Most rules will also include a third aspect, a Boolean expression describing the property the rule is checking, the *loop block*. It should be noted that the rules are never executed, only parsed out to choose the appropriate backend JavaScript function to actually perform the test on the rule requirements.

### 3.2.1 Construction

There are a few keywords that are used for constructing rules: **run, for, if, then, else, and, or, and not**. The starting point for rules is either **run** or **for**. The **run** rules are a mechanism to allow the checking of multiple rules on the site. The other keywords and operators are used within the **for** loops to create the conditions and content of the rule. Underscores are also reserved for identifying distinct objects within the tuple and should not be used for naming properties; camel case is recommended instead.

### 3.2.1.1 Objects

Objects in the *ux-rule* rule specification language in general refer to HTML elements, as well as containing any user-specified custom properties that can apply to those HTML elements. These form the atomic level portion of the language.

**Tags**

This set is the names of all HTML tag types, such as *div* or *img*. They are only the name, not the surrounding tag brackets. This set of tags also uses *link* to represent a tag type of *a* and *page* to represent a tag type of *html* to make the rules easier to read and understand. It also includes a tag type of * which represents any tag/all tags, not the empty set. The value of a tag is a set of all its attributes and their values.
Attributes
This is the set of all possible attributes of all HTML tags. These include attributes that can be applied to any tag, such as `lang`, as well as attributes that are specific to certain tags, such as `alt`. It also includes attributes that support ARIA roles and requirements, such as `role` and `aria-describedby`. The value of an attribute is the Boolean, string, or number value assigned to it within an instance of a given tag.

Properties
This set of objects includes all of the properties that are not specified as an HTML tag or attribute, but are named by the programmer to describe the property they represent. These sort of objects would be similar to `isARIA`, which holds whether a tag or attribute meets an ARIA requirement, or `words`, which could represent a word count of a string valued tag or attribute. The value of a property is assigned by the programmer can be either a Boolean, string, number, or range, depending on what best represents the property. For example, rules like `isARIA` would most likely have a Boolean value, while other rules, such as `words` might be a number or a range.

3.2.1.2 Object Relationships

Although atomic portions, objects have a relationship to each other and the language supports specifying that. In the syntax listed, $T$ should be taken as follows:

$T \rightarrow t | T: t$, where $t$ is a tag object.

$T: t$
The syntax represents a parent : child relationship between two tags. Since tags are often nested many levels deep within HTML code, this allows tags to be fetched only when they occur under a specific parent tag. For example `li:link` would be the set of `a` tags that occur as list items on a webpage. It can also be used multiple times to create a hierarchy to generate a very specific set of resulting tag objects. For instance, `nav:ul:li:link` would describe the set of `a` tags that exist within a navigation menu list.

$T.a$
Given the tag set of $T$, this is the set of all values of attribute $a$ for those tags. Each instance of a tag has its own version of the attribute $a$ and that value forms one item of the final set. It could be similar to `img.alt`, which builds from the `alt` attribute of all the `img` tags, or `li:link.href`, which is the set of all `href` attribute values from links within lists.
This set is similar to the above; instead of working with existing attributes for the tags, however, the programmer is assigning a property $d$ to the tag instances. Similar to the previous examples, this could be something like `form.isARIA`, which represents testing for meeting ARIA requirements, or `li:link.code`, which would represent the server return code of each link in a list.

As properties can be applied to tags, they can also be applied to tag attributes. A property $d$ applies to a specific instance of attribute $a$, which is linked to a specific instance of a tag $T$. An example of this would be `img.alt.words`, which represents the number of words in the image alternate text string.

### 3.2.1.3 Grouping Syntax, Ranges, and Scoping

To support direct comparisons of tags and attributes for consistency testing purposes, `ux-rule` has grouping syntax to provide the necessary tuples and differentiation labels. Also in this section is a description of the syntax for defining a range and how the scoping for the blocks should be interpreted when reading the rules. Since rules are not executed, the only thing inherited across scope are the objects.

#### <obj₁, obj₂>

A comma-separated list of objects representing a tuple. This is at minimum a pair of objects. Objects do not need to be of the same type; a group can contain tags, attributes, constructed properties, or any combination of those types. Groups are only used within the `for` loop setup block. Following the object name with an underscore and a number distinguishes the objects in the set if they share the same name. It is not necessary when objects are different, but if it is used, it should be used the same throughout the loop block. As an example `<img₁, img₂>`, a pair of `img` tags, or `<label, input>`, which could also be written `<label₁,input₂>` as long as within the loop block they are referenced consistently by these names.

#### [m₁, m₂]

A pair of numbers enclosed within a set of square brackets and separated by a comma represents an inclusive range. The numbers come from the set of real numbers, except where a range is used as a quantifier, in which case the numbers must be integers. Mathematically, a value $v$ is within the
range if and only if \( v \geq m_1 \) and \( v \leq m_2 \). Either number can be filled with a * indicating no lower or upper bound, for less than or greater than functionality, as well as any numerical value functionality when it is substituted for both bound values. Some example ranges might be \([2,4]\), \([*,8]\) (indicating at most 8), \([2.5,\ast]\) (indicating at least 2.5), or \([*,\ast]\) (indicating any number).


\[
\{ \ldots \}
\]

Anything within curly braces is its own scope. Elements of a scope are inherited from an enclosing scope as well as anything defined within the current scope, but the inheritance does not work the other way. Given a case of \( \{ s_1 \{ s_2 \} \} \), within scope \( s_2 \) information about scope \( s_1 \) can be seen, but within scope \( s_1 \), information about scope \( s_2 \) cannot be seen. If any object in the inner scope is the same as an object in the outer scope, the object from the outer scope will not be inherited. If the same object is to be used in both scopes, appending the underscore followed by a number to differentiate the objects should be used to allow proper inheritance.

3.2.1.4 Quantifiers

The quantifier is what determines the evaluation of the \texttt{for} loop rule body. It is also a main component of a setup block and necessary for every rule. It can have a numerical value or range, as well as a set of keywords describing more abstract quantities: \texttt{all}, \texttt{some}, \texttt{most}, and \texttt{no}.

\( n \)

When the quantifier is a single integer, it means that exactly that many objects must meet the rule condition. It must be an integer value greater than 0, since 0 is covered under the \texttt{no} quantifier and it is impossible to have an object fractionally meet a Boolean condition, which excludes any decimal numbers.

\([n_1,n_2]\)

A quantifier can be an inclusive range of integers; again because it is impossible for an object to fractionally meet a Boolean condition, it is limited to integers. This is the same range described in the Grouping Syntax section, only limited to integer values. Using * as a value for range bounds is acceptable as well, replacing either bound or both bounds.

\texttt{all}

This means that all the objects must meet the condition of the rule in the loop block. This is the only quantifier used in the \texttt{run} keyword setup block, since it should evaluate all the rules in a set.

\texttt{some}

This quantifier indicates that at least 1 object must meet the rule condition in the loop block, and places no upper bound on the number beyond that.
most

Under this quantifier, at least half of the objects in the set must meet the rule condition.

no

Selecting this quantifier means that none of the objects must meet the rule condition. It is used in place of a 0 as a numerical quantifier. It should not be used in conjunction with a not operation when the not is within the immediately following loop block.

3.2.1.5 Operations

These operations are all Boolean operations in that they have a Boolean result, even if the operands are objects instead of Boolean expressions. The variable $B$ represents this set of expressions in the syntax constructions below. They will be used only within a loop block. For these operations $T$ and $A$ should be taken as follows:

$$T \rightarrow t \mid T: t$$, where $t$ is a tag

$$A \rightarrow T. a \mid T. d \mid T. a.d$$, where $a$ is an attribute and $d$ is a constructed property

$T_1 == T_2$

This is a comparison between two tag objects. It evaluates to true if and only if all the attributes of $T_1$, and their values, exactly match those of $T_2$. This equivalence also extends to the tag type; comparison between two different tag types will result in a false return value. As examples, $p_1 == p_2, img_1 == img_2$. This comparison can only be done with a group of objects specified in the setup block.

$A_1 == A_2$

This comparison is between two attributes or properties. The operands can be two different attributes or properties of the same tag instance, or can be the same attribute or property of different tag instances. It evaluates true if the attribute or property instance values are equal and false otherwise. For example, $img_1.alt == img_2.alt$, which compares the alternate text of two img tags. This comparison can only be done with a group of objects specified.

$A = r$

This operation applies a regular expression $r$ to the string type attribute or property value. If the value is consistent with the regular expression, it
evaluates true, otherwise, it evaluates false. As much logic as possible should be contained within the regular expression. It would look similar to \texttt{img.alt = "^(!\ ((img|image)\ ).+"}, where the string on the right side is the regular expression, in this case saying the text is not “img” or “image”.

\[ A = m \]

This is a numerical comparison between a real number \( m \) and the real number type attribute or property value. If the values are mathematically equal, then it evaluates true. An example: \texttt{img.height = 15.5} returns true if the image height attribute is 15.5, and false otherwise.

\[ A = [m_1, m_2] \]

This comparison checks that the value of the attribute or property is within the inclusive range specified by real numbers \( m_1 \) and \( m_2 \). If this is the case, it evaluates true, false otherwise. For example, \texttt{img.width = [10, 30.5]} or \texttt{img.width = [25, *]}, which says the image width must be at least a value of 25.

\[ A < m \]

This checks if the attribute or property numerical value is strictly less than real number \( m \), evaluates true if it is, otherwise false.

\[ A <= m \]

This checks if the attribute or property numerical value is less than or equal to real number \( m \). If it is less than or equal, it evaluates true, otherwise false.

\[ A > m \]

This checks if the attribute or property numerical value is strictly greater than real number \( m \), evaluating true if it is, false in all other cases.

\[ A >= m \]

This checks if the attribute or property numerical value is greater than or equal to real number \( m \). If it is, it evaluates true, if not, it evaluates false.

\[ A = b \]

This compares the Boolean value of an attribute or property to values of either \texttt{true} or \texttt{false}. If the attribute or property value is \texttt{true} and \( b \) is \texttt{true}, or the value is \texttt{false} and \( b \) is \texttt{false}, then it evaluates true. Otherwise it evaluates to false.

\texttt{not B}

Negates the result of any other operation expression. If \( B \) evaluates to \texttt{true}, this evaluates to false. If \( B \) evaluates to \texttt{false}, this evaluates to true.

\texttt{B_1 and B_2}

Performs a logical \texttt{and} operation on the evaluations of \( B_1 \) and \( B_2 \). This evaluates to true only when both \( B_1 \) and \( B_2 \) evaluate to \texttt{true}.

\texttt{B_1 or B_2}

Performs a logical \texttt{or} operation on the evaluations of \( B_1 \) and \( B_2 \). This evaluates to false only when both \( B_1 \) and \( B_2 \) evaluate to \texttt{false}. 
3.2.1.6 Conditional Statements

Conditional statements can be nested and can contain `for` loops within any of their blocks, as well as any of the Boolean operations from the above section. This set of statements that can be inside the conditional blocks will be referred to as $F$.

**if** $(F_1)$ **then** $(F_2)$

The logic for this statement is concerned primarily with the evaluation of $F_1$, in that unless $F_1$ evaluates to `true` and $F_2$ evaluates to `false`, the statement evaluates to `true`, as seen in the truth table below:

<table>
<thead>
<tr>
<th>$F_1$</th>
<th>$F_2$</th>
<th><em>If-then</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

*Table 2: Truth Table for IF-THEN statement*

**if** $(F_1)$ **then** $(F_2)$ **else** $(F_3)$

The main factor in evaluating the logic of this statement is again the evaluation of $F_1$. The truth table below enumerates the possible results given all three evaluations:

<table>
<thead>
<tr>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th><em>If-then-else</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True/False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True/False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True/False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True/False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

*Table 3: Truth Table for IF-THEN-ELSE statement*

3.2.1.7 Body Syntax

A rule must start with a rule body and is filled in from that point. There are two types of rule bodies, `run` and `for`, which serve different purposes. $F$ again refers to the set
of all the above operators and conditional statements, as well as \textit{for} loops. \( O \) refers to objects: tags, attributes, properties, and tuple groups \((<,>)\), and \( q \) refers to any of the quantifiers mentioned above.

\textbf{run( all \( l \))}

This rule body is designed for running a group of rules, as specified by \( l \). It produces a set of results, one value for each of the rules that were included in the set. The \textit{all} quantifier is the only quantifier allowed, since the groups of rules should always be run in their entirety. This rule body also is part of the function supporting certification estimation. The names for the rule groups can be any string that does not start and end with a bracket of any sort, such as \texttt{ARules or LangTests}, which would look like \texttt{run( all ARules )} and \texttt{run( all LangTests)}, when put into the rule body. The former would be to run the A level certification rules and the latter would be to run all of the tests relating to language. The actual listing of the rules is currently done in the back end JavaScript code, by running the appropriate functions, but future work will allow a set of rule strings to be passed in through \( l \), either retrieved from a JSON file or a string directly passed in. It \textbf{should not} be used for running single rules; passing the single rule to the \texttt{ux-rule} check function is more efficient.

\textbf{for (q O){F}}

The loop rule body evaluates \( F \) on every object in the set \( O \), and returns an evaluation based on the quantifier. The starting rule body of this type is not permitted to have a property as the object or within a group, but any instances after that may. Using the example of alt text existence, a simple version of the rule would look like this: \texttt{for (all img){img.\texttt{alt} = \\\texttt{"+"}}}.

\subsection*{3.3 Rule Construction Examples}

This section provides a couple examples of rules outside of the ones implemented for this thesis to demonstrate some of the possibilities. Keeping in line with accessibility, these rules are nominally Each Input Has Label and Button Text is Consistent.
### 3.3.1 Each Input Has Label

Within the WCAG are a number of guidelines dealing with ensuring various non-text elements of webpages are properly labeled. This rule looks specifically at input and whether they have a label. The English version of the rule would be “for every input there is one label associated with that input.” Since the rule doesn’t involve running a group of rules, the starting point for the rule would be a `for` loop. There are two different quantifiers in the rule, “every” and “one”, so two `for` loops are needed. There are also two different tags, but they cannot be put into a group because of the different quantifiers. On the HTML side, in order to link labels and inputs, the label has a “for” attribute that matches the “name” attribute of the input tag. That will be the core of the conditional portion of the rule, a direct comparison between two attributes. Putting it all together, the resulting rule in the `ux-rule` language looks like this:

```
for (all input){ for (1 label){ input.name == label.for }}
```

### 3.3.2 Button Text is Consistent

Consistency is another big area within the WCAG. The implemented consistency rule deals with image alternate text, but there are other elements that should be consistent beyond that. This rule is dealing with the same functionality having the same labeling. Its written English version would be something along the lines of “for every pair of buttons, if the button type matches, the button text also matches, and if the button text matches, the button type matches.” Again, this rule is a single rule, not one that executes
multiple rules, so its starting point should also be a for loop. It has two sets of
conditionals, combined with an and. There is no need for any else statements so the if-
then is all that is necessary. The button tag in HTML has a type attribute and a value
attribute that are used for this comparison. Assembling the rule gives:

for (all < button_1, button_2 >) { if (button_1.type == button_2.type) 
  then (button_1.value == button_2.value) and
  if (button_1.value = button_2.value) 
  then (button_1.type = button_2.type)}

3.4 Current Parsing Capabilities

The current implementation of the parser is only capable of parsing out the rule
sets from the run rule construction. This was done to allow scripted testing in the back
end code verification. However, the pseudo-code style of the rule language is easily
parsed into an abstract syntax tree (AST), which is a parsing technique used by
compilers. Due to time constraints the full parser was not implemented, but the
groundwork is contained within the rule language to specify how it should be parsed.
CHAPTER IV

UX-RULE IMPLEMENTATION

The ux-rule framework is intended to cover some of the limitations mentioned in the previous chapter. It is meant to be an open source and free package, so that it can be used by anyone designing a site. Building it in JavaScript to run in Node means that it can cover a wide range of platforms. The rule language that it defines within the framework is similar to code, since the intended audience is programmers themselves, and flexible enough to allow the specification of a variety of rules both inside and outside the WCAG guidelines. It also attempts to overcome the limitation of looking only at HTML code for analysis and perform a number of dynamic content tests as well. The system has the set up to estimate the completion of a certification level, but makes no guarantees of certification, as there are rules that cannot be automatically tested.

4.1 Retrieved HTML vs. Rendered HTML

There are two different types of webpages, static pages and dynamic pages. Static pages involve HTML code that doesn’t change and anything that changes a webpage state, takes the browser to a new page. Dynamic sites have content that is filled in by a
script at the time of load or as some aspects of the page change [12]. Sites like Tumblr or Twitter have this functionality, with their infinite loading or posts or tweets as the user scrolls down. The Weather Channel website is another good example, as it determines the background picture for the forecast display by a script depending on time of day and predicted weather. The dynamic site functionality has recently become more common in websites, which poses a difficulty for web scraping libraries, which were initially developed to handle the static sites [12]. This “rich code” is a limitation that current web scraping techniques share with automated webpage testing.

The only way that exists as of yet to collect the rendered HTML is to use a headless browser or WebKit based library [5]. Of the two libraries for fetching the HTML code for a page used in the ux-rule implementation, only PhantomJS is a web kit and can collect the rendered HTML [13]. Request can only grab the HTML code as it is initially served from the webserver, which limits some of the ux-rule framework implemented rules from functioning properly on dynamic content. The other rules implemented with PhantomJS are checking rendered HTML, but may not be grabbing the fully rendered HTML depending on load times of the elements. This is because specifying actions to occur when the page load is finished is difficult to accomplish with Phridge as the bridge, although the other advantages of using it still make it the best choice.
4.2 System Outline

The construction of the *ux-rule* framework requires the use of a number of npm packages. To illustrate how these packages interact with the program, the sections below describe the architecture of the system. Also included below is a section describing how the rules are constructed and reasons behind these design choices. This section of the paper provides the necessary background information to understanding the function of the code described in the next section, which goes into detail about the focus areas and the rules written as the proof of concept.

4.2.1 Architecture

The *ux-rule* framework is designed to run in the Node environment, which comes with the node.js package, along with its package manager, npm. Since this framework is designed to be an open-source npm package release, ensuring that it runs in Node is necessary. In addition to Node, *ux-rule* makes use of the following packages: Request, Async, Cheerio, Lodash, PhantomJS, Phridge, and Jsonfile. Figure 1 shows how these packages interact with each other and the system as a whole.
Working from right to left in the diagram, the first packages to expand on are Phridge and PhantomJS. PhantomJS is a package designed for headless webpage testing and serves as a standalone environment for running JavaScript code. However, it does not run within Node, making a bridge necessary to interact with it and make use of its capabilities [47]. Phridge serves as this bridge, spawning a PhantomJS environment and passing it commands to execute within that environment. It also handles the callbacks, providing a way to specify code to execute once the PhantomJS commands are finished executing, the environment closed, and the memory freed [48]. It is PhantomJS that interfaces directly with the provided webpage, loading up the page real time.

The next package is Async. Since the JavaScript code is executed asynchronously, relying on callbacks to force any synchronicity [3][59], there are times
when working with large data sets that can cause issues with memory or efficiency. In this case, with so many links to open connections to, there is a point where no more child processes can be spawned or sockets hang. To prevent this, certain sections of code need to execute synchronously, which Async handles [29]. It is not as efficient as running in parallel, time-wise, but it is more efficient memory-wise.

The functions that Lodash provide are for set processing, and more efficient than the standard JavaScript functions. Particularly, the code uses the `forEach` and `every` functions, which take a set and a function that processes an element of the set down to a Boolean result. Since the processing function returns a Boolean, they can support early exiting when a false result is returned. The `every` function is very good in this case because it tests whether a whole set conforms to the property described in the processing function. The Lodash functions do not support a callback however, which does pose a challenge when the sets are large enough that execution on them hasn’t completed before using the return value [44].

The Request package also forms a connection to a given URL, but is designed more to make the connections than perform any headless testing on the pages. Since it connects directly to the webpage over http or https, depending on the URL passed to it, obtaining the server return code and other information like it is very easy. It has the capability to traverse redirects to their end, if it is within a certain number of jumps, as well as stop at a redirect with a setting in connection setup. It can also load the body of a webpage, meaning the HTML code that makes it up [49]. Request itself doesn’t have any functionality to scrape the code or process it in any way, but with the help of another package, Cheerio, obtaining specific information from the HTML is easy.
Cheerio is a package designed for web scraping, and provides objects representing the tags scraped for, which include their many possible attributes. Both tags and their attributes can be scraped for, even looking for specific values attached to those attributes [30]. It can also run in the PhantomJS environment, to allow scraping outside of the JavaScript DOM element ‘document’ [43], which is the included method within PhantomJS [47].

The last package is Jsonfile, which provides a mechanism to write to JSON files in a simpler way than the JSON write functions [46]. Since ux-rule provides a mechanism for estimating certification under the WCAG, it uses this package to write the results of the certification estimate as a JSON object to a JSON file, so the end user can read in the values and format them to their liking. Passing the information this way provides the raw data instead of a verbose report, and allows for the data to be reported in the best manner for the person looking at it, be it programmer or supervisor.

There are two packages used in ux-rule that interface with a webpage and its HTML code, but while they appear to be able to show the same end capability and are interchangeable, they are not and each serve their own purpose. PhantomJS is used when there needs to be preprocessing on the data before it is passed back to Node and to handle dynamic pages. Phridge’s callback system is well suited to this, ensuring that the data is returned in its final form before it is used. Request and Cheerio are best in the other cases, where the data can be scraped and used as is. Request is also better for getting information from the webserver, which is a more complicated process in PhantomJS, and when the only thing that needs to be obtained is the existence of the page at the other end, since it doesn’t require the memory that the PhantomJS environment does. Request does
not collect rendered HTML. While there is no documentation for Request specifying that it does or does not grab rendered HTML, in the course of testing this tool, it was found that it was unable to retrieve the dynamic data. Choosing it then for its efficiency, does pose a problem with testing dynamic pages. However, by using both of them, this proof of concept can demonstrate their uses and in which situations they fall short of what is needed from them to make the tool as effective as its potential to be.

Each package used in the construction of *ux-rule* serves a purpose and is used only for that purpose. The functionality of the packages cannot be interchanged with one another, no matter how similar they might seem.

4.2.2 Certification Estimation Setup

The WCAG [67] consists of a number of guidelines in regards to designing a website to be accessible. Several of these guidelines have levels to them, with increasingly stringent requirements, and are ranked as A, AA, or AAA level, as well as a small subset of 4 A rules that constitute a Non-Interference group, which requires that anything on a page not used in a way to support accessibility must at least not interfere with the users ability to access anything else on the page. The WCAG also lays out certification guidelines for each of those levels, which involve meeting all the guidelines of a particular level.

Some of these guidelines have requirements on meaning, which is still a difficult problem for computers to solve, even with context analysis. For example, the requirement that image alternate text be descriptive of the image is hard for a computer to
check. Image processing could be done to pull out what is in the image, and look for keywords in the text, but that would all take as long, if not longer, than a human doing the same job. It is far more efficient and accurate for a human to look at the image and make the determination on the text themselves. Additionally, doing that sort of processing can become incredibly slow, which goes against the principle of *ux-rule*. Computers can process language and images, but understanding them is still beyond their capabilities.

That being the case, *ux-rule* cannot say for certain that a website passes a certification level, so it offers an estimation instead. For the guidelines, and their associated rules, that can be completely checked by a computer, a pass equates to a score of 1. These include all guidelines that specify “programmatically determined” at a bare minimum. The rules which require some form of human component, but can have a preliminary check done by computer, score 0.5 on a pass. Image alternate text is a prime example of this. Any remaining guidelines or a fail receive a score of 0. Using these scores, it is impossible to obtain 100% for the estimation, as it should be. The estimate is not a guarantee, but a way to see how close to the certification a website is and reduce the amount of verification that has to be done by a human. Other tools generate reports on the status of the certification, but do not necessarily report any measurable metric beyond the number of failures or warnings generated. In this respect, *ux-rule* differs, providing instead a way to know how close to covering all the necessary guidelines a page is, which would be important to know if, for example, all of the failures in the HTML page are related to a single guideline, or there are a number of failures spread out across many guidelines.
Any set of tests can be grouped and run in a similar fashion should there be some other certification or requirement that must be met outside of the WCAG guidelines, specified by laws or corporate standards for example. As this project is a proof of concept, the full set of guidelines from the WCAG has not been programmed in; however, the framework to add them in is there. The guidelines that have been programmed in thus far are the image alternate text existence, which scores 0.5 points, a page having a language and parts of a page having a language, which both score 1 point, navigation menu consistency, which also scores 1 point, skip link existence (a way to bypass blocks), scoring 1 point again, and all forms meet the ARIA error requirements (input error identification guideline), which scores 1 point.

4.3 Using *ux-rule*

Using the *ux-rule* package takes only a few steps. Before stepping into the coding itself, the rule should be designed and written up; the details of constructing the rules have already been discussed in Chapter 3 and a tutorial can be found in Appendix A. Once the rule is constructed, it can be passed to the execution side of *ux-rule* via the package commands in JavaScript. Figure 2 depicts an example of what the call would look like.

![Figure 2: *ux-rule* code called in JavaScript. Line 1 - including the library. Line 3 - the URL to be checked. Line 4 – the call to *ux-rule* with the rule, in this case, the rule for image alternate text existence.](image-url)
The first line of code includes the library so that the program running the code can find the commands it needs to execute. Line 3 is where the URL to be checked is stored as a string. This could be passed directly into the check function, but coding it this way allows the programmer to more easily change the URL for different tests of the same rule. Line 4 is where the actual call to the library is done, providing it the URL and the string containing the rule. The rule used for this example is the rule testing for image alternate text existence. If the rule is a custom rule, not already included within the framework, then it will need to be coded into the ux-rule back end. After it has been coded in once, it can be used from that point on.

Since ux-rule is coded to run in Node, that is what should be used to execute the JavaScript file(s) containing the call to the library. If the code above is saved in the examples folder as example2.js, then it can be run from the terminal in the manner shown in Figure 3.

![Figure 3: Running the JavaScript code containing the ux-rule call in Node.](image)

Running that produces the following output in the terminal:

![Figure 4: Terminal output for running the command in Figure 3.](image)
The output tells what rule is being run, in this case the image alternate text existence. From there it differs by rule, which will be elaborated on in the next sections. For this particular rule, it gives the number of images it checked, how many are missing alternate text, and then its determination on whether the rule was passed or failed. This example passed the test, checking the 2 images on the page and finding them both to have alternate text.

4.4 Focus Areas

To best demonstrate that ux-rule succeeds as a proof of concept, the rules chosen are a small subset based upon the WCAG guidelines, rather than the complete set. A few of the rules are not based upon those guidelines, to show the potential for added rules as well as the range of rules that can be written using the framework. The selected rules can be broken down into 3 categories: content testing, site structure testing, and input testing.

4.4.1 Content Testing

This area of focus is about the HTML content of the websites and the information that can be scraped from it. It covers a number of rules about image alternate text as well as programmatically determining the language of a website and its sections. It also
includes a rule about the existence of skip links, which are an important factor for screen reader support.

4.4.1.1 Image Alt Text Testing

The most common thing for accessibility testers to check is that all images have an alternate text attribute specified. That being the case, it is a good starting point to show the potential range of rules that can be expanded from it. None of these functions can check anything about the meaning of the alternate text string, but they can impose some restrictions on the formatting of the string, as might conceivably be required by a corporate coding standard. To start with is a rule to check that the alternate text exists. From there is a check that the text meets a condition provided by a regular expression, followed by checking that the text is consistent between the same image and checking that it is within a particular word limit. The majority of these rules have been used previously in this paper to show the rule construction, but will be detailed more in this section.

*Alt Text Exists.* Checking for the existence of image alternate text is one of the simplest tasks that an accessibility checker can perform. The rule for it, under the framework is:

```javascript
for (all img) { img.alt="+" }
```

which can be represented with JSON objects as follows:
{quant: 'all',
tag: 'img',
attrs: { alt: '+' },
func: 'allImageTagHasAltAttribute'}

After parsing the rule down to its elements, the function to run can be determined, and is linked to the rule with the **func** object. The actual function names should be a description of the rule they check, both to make it clearer to others the task of that function as well as allow programmatic construction of the name from the rule itself.

The way *ux-rule* presently goes about checking this condition is to scrape the image tags from the HTML code and check for the existence of the alt attribute as well as the attribute containing more than an empty string. It retrieves the HTML code of the page, using Request to form a connection to the provided URL, and then uses Cheerio to collect all the image tags. The image tags set does not require any pre-processing, so using Request and Cheerio is the more efficient choice, and overall requires fewer lines of code than using PhantomJS and Phridge, although it cannot grab any dynamically loaded images. Then, using the Lodash function **forEach**, it fetches the alt attribute of each image and checks it against the conditions for existence. Since outputting information about the images failing the test is important, the Lodash **every** function cannot be used, as it would stop running through the images when it first encounters one that fails the test. Once all the image tags have been processed, it outputs the number of images it tested, the number of images missing their alternate text, and its final conclusion about whether all the images have their alternate text, either true if they all passed or false if any images failed the test. The idea behind reporting the total number of images is to give the tester an idea of whether most of the images are missing alternate
text or a few images were forgotten. The callback for this function passes a score, either
0.5 for a pass or 0 for a fail, to support the certification testing.

Alt Text Meets RegEx Condition. There are a number of conditions that
developers might want to apply to their image alternate text, either for personal reasons
or to meet coding standards. Many of these conditions can be expressed with regular
expressions, since the requirements are more likely to do with formatting than the actual
content of the text. The *ux-rule* framework makes the following rule to accomplish this:

```
for (all img) { img.alt=CONDITION }
```

where *CONDITION* is replaced with a regular expression. The JSON breakdown of this
rule is similar to the previous rule, since the condition of existence can be expressed with
the regular expression ‘+’:

```
{quant: 'all',
tag: 'img',
attrs: { alt: CONDITION},
func: 'allImageTagAltMeetsCondition'}.
```

There are many conditions that can be expressed by regular expressions; for example, the
tester could check that the alternate text is a full sentence, or that it isn’t just a link. They
could also check that the text isn’t ‘img’ or ‘image’, which would pass the existence
criteria, but are essentially filler. The exact rule for doing that would look like this:

```
for (all img) { img.alt="^(?!(img|image)$).*" }
```

The condition would be parsed out of the rule and stored in a global variable, but for the
time being it has been hard coded as the regular expression describing a full sentence,
using a simple definition of a sentence [14].
The associated function is different than the existence function, because it has different functionality. Where the first function checks only for the existence of the alternate text, this function performs an additional step and checks the string against the regular expression as well. The code between the two functions is nearly identical, and makes use of the same packages and functions. The difference lies in how it processes the image tags. It first checks for the existence of the alternate text. If it doesn’t exist, it doesn’t even bother checking the text against the condition and outputs information about the image so the tester can find it in the HTML. If the text does exist, it applies the regular expression test to the string. In the case that it fails the test, it outputs both the image information and the offending alternate text. The function reports the number of images processed, the number of images missing alternate text, the number of images whose alternate text fails the regular expression condition, and the result of the set processing, either true or false for all images having alternate text that meets the given condition. The callback for this test also returns either 0.5 on a pass or 0 on a fail; although the conditions tested are not part of the WCAG guidelines, it does check for image existence and is therefore scored the same way.

*Alt Text is Consistent.* Consistency is a major factor in a number of the WCAG guidelines, and although there isn’t a specific guideline referring to it in the case of image alternate text, checking for that consistency is a logical step. It should be noted however that there may be times when this rule would not desired as a test and there is no guideline that specifically requires it. This rule also serves to demonstrate a multi-
layered rule, showing the flexibility for adding additional conditions to the test they
describe. The rule:

```javascript
for (all <img_1, img_2>) {
  if (img_1.src == img_2.src)
    then (img_1.alt == img_2.alt)
  else (not img_1.alt == img_2.alt)
}
```

shows how the framework allows for those conditions to apply to subsets of objects,
using the ‘<’ and ‘>’ enclosure. In this case it represents a pair of objects, but could
easily represent a triple or more as well. The JSON breakdown is different from the rules
mentioned previously, but still maintains a similar, if a larger, set of objects:

```json
{quant: 'all', tag: 'img', cond: {
  if: {'==', {Tag1: {name: 'img_1', att: 'src'},
             Tag2: {name: 'img_2', att: 'src'}}},
  then: {'==', {Tag1: {name: 'img_1', att: 'alt'},
              Tag2: {name: 'img_2', att: 'alt'}}},
  else: {not: {'==', {Tag1: {name: 'img_1', att: 'alt'},
                     Tag2: {name: 'img_2', att: 'alt'}}}},
  func: 'allImageTagAltIsConsistent'}
```

The `cond` object is where the information about any if-then-else statements within the
loop body is broken down. These if-then-else statements can also support `not`, `and` and,
or operators, to handle more complex conditions.

The function that performs the test uses the Request-Cheerio combination to
obtain the images from a page, along with their attributes. Then using nested Lodash
`forEach` loops, it compares each image with every other image in the set, checking first
to see if the image has alternate text, then checking that if the source of the images are the
same, then the alternate text is the same as well and checking that if the alternate text is
the same, the images have the same source. In the case that the images are the same, but
don’t have the same alternate text, it outputs the image information and the differing alternate texts. In the other case, where the alternate text is the same but not the images, it outputs the information from both images and the duplicate alternate text once. Once it is done, it reports the number of images processed, the number of images missing alternate text, the number of images that were inconsistent, and the conclusion of whether all the images are consistent, true for a pass and false for a fail. Since the function compares every image to every other image, there will be twice as many recorded inconsistencies, so the number that it reports at the end is that count divided by 2. Like the other image alternate text functions, it checks for alternate text existence, so its callback is also 0.5 in the case of a pass and 0 in the case of a fail.

*Alt Text is Within Word Limit.* In order to keep the alternate text on images from becoming too verbose, it could be useful to place a word limit on the text. This rule is separate from the rule above that accepts regular expressions as a condition, since any regular expressions dealing with limiting words are extremely unwieldy. The word count of the text could also be considered a property of that text, so this rule also provides an example for when the condition is not directly an attribute of a given tag:

```javascript
for (all img) { img.alt.words=[X,Y] }
```

where X and Y are two integers representing the inclusive limits on the number of words. This range can be conveniently be stored as a JSON array within the rule breakdown:

```json
  { quant: 'all',
    tag: 'img',
    attrs: {
      alt: {
        words: [X,Y]
      },
      func: 'allImageAltTextWithinWordLimit'
    }
  }
```
and the `words` property of `alt` is stored in `alt` as an object containing the array. Like the regular expression condition mentioned in the above function, the range for this has also been hard-coded. Since multiple functions make use of the range global variable, the function sets the range to `[2,4]` before using it to facilitate testing.

Testing this condition happens similarly to the other image alternate text functions. Request and Cheerio are used to obtain the images. A Lodash `forEach` loop is used to iterate through the images and it first checks for the existence of alternate text. The simplest way to get the number of words in a string is to split that string on a ``, or space character, and take the length of the resulting array. Defining what constitutes a word and a sentence can quickly become complicated, so this makes use of a simple definition of a word under the standard way strings are written in English. If the number that it gets is outside of the range provided to the function, it outputs the image information and the failing alternate text. When it completes the test, it reports the number of images it went through, the number of images missing alternate text, the number of images with alternate text containing more or less words than the limit range, and the test conclusion, true if all the image alternate text has a number of words within the range and false if any image alternate text fails the test. Its callback returns 0.5 and 0 for a pass and fail respectively, again because it also checks for image alternate text existence.
4.4.1.2 Webpage Language Testing

The WCAG guidelines specify that the language of a page should be programmatically determinable, which means that the computer should be able to determine the language on its own. While there are natural language processing techniques to determine the language of a selection, HTML has an lang attribute which can be applied to any tag and is the preferable choice since that attribute is what screen readers look at for the language. Finding a language specified for the entire webpage is an A level certification criterion, and all sections having a language belongs under the AA certification level. Checking that some sections have a language is not for any guideline, but more of a demonstration of a different quantifier.

Language for Entire Page. This rule is an existence check primarily, so its structure is very similar to that of the image alternate text existence rule:

```javascript
for (all page) { page.lang="" }  
```

where `page` refers to the `<html>` tag and the content inside of it. Page is easier to understand, and can still be mapped to `<html>`. Using “page” here might be confused with representing all the elements on the page, but the same goes for using “html” there instead, which could also be taken as a representation of all the HTML elements of the webpage. Given this confusion, “page” was selected because in translating the rule from the pertinent guideline, it allows for the most directly similar phrasing. As `alt` is an actual HTML attribute of the image tag, `lang` is an attribute for any tag in HTML, in this case, looking specifically at the attribute associated with an `<html>` tag. The JSON
breakdown is similar in structure to the image alternate text existence rule, but it reflects
the mapping from page to <html>:

```json
{quant: 'all',
tag: 'html',
attrs: { lang: '+'},
func: 'allPageHasLang'}
```

The tag value has been stored as html instead of page, but all other aspects of the
breakdown remain the same directly as they are from the rule.

Since this function is only retrieving one thing, the most efficient course of action
is to use Request and Cheerio to get the lang attribute of the <html> tag. If the value
is undefined, then the attribute isn’t specified and therefore doesn’t exist. In that case the
function reports that it couldn’t find a page language. If it has any value, the function
reports that it found a language and the language code it found. The callback is a
Boolean value, either true or false, that should be interpreted as integers 1 and 0
respectively, for certification scoring purposes.

*Some Sections Have Language.* Checking that languages exists elsewhere on the
page, perhaps where there might be a small section in another language, is another rule
that can easily be programmatically checked. It also shows that even more abstract
concepts can be applied under the framework, handling things such as the some
quantifier:

```javascript
for ( some TAG ) { TAG.lang="+" }
```

The some quantifier indicates more than one object passes the test, but does not exclude
the possibility of all objects passing the test. The JSON breakdown shares its structure
with the previous rule:
{quant: 'some',
tag: TAG,
attrs: { lang: '+' },
func: 'someSectionsHaveLang'}

The code for this rule is also nearly identical to the previous rule, the only difference being that it scrapes all the tags of the proper type as well as the overall <html> tag and loops through them with the Lodash forEach, checking for the existence of any lang attributes. It keeps track of the number of occurrences and once it has gone through all of the tags, it checks if that number is greater than 1. If it is, the function reports that it found languages for some sections, otherwise it reports that it did not find any, since it doesn’t count the page language as a section language. Its callback returns a Boolean, which should be interpreted as a Boolean, not an integer value, since it is not one of the scoring tests under the certification requirements.

All Sections Have Language. This rule covers an AA certification guideline, which says that all of the sections of the webpage have a language specified. The definition of a section however, is ambiguous, so it can better be presented as all of a certain tag type must have a lang attribute. By allowing the tester to specify which tag level constitutes a section, the rule removes that ambiguity and even permits checking all tags with ‘*’ passed in the rule. Since this is a AA certification guideline, it requires that the A level version be passed as well, which means that the page must also have an overall language specified.

This rule is written similar to the previous two:

for (all TAG) { TAG.lang='+' }
where \textit{TAG} refers to the tag that the tester defines as a section, such as \texttt{<div>} or \texttt{<p>} for example. The JSON breakdown for this rule is again nearly the same as the previous rules and maintains the similar structure:

\begin{verbatim}
{quant: 'all',
tag: \textit{TAG},
attrs: { lang: '+' },
func: 'allSectionsHaveLang'}
\end{verbatim}

where the \textit{TAG} value under the \texttt{tag} object represents the tester chosen tag. The tag is stored in a global variable when the rule is parsed out. Its value is presently hard coded as ‘p’ to allow for testing.

The code of this function extends the code from the previous rule. Instead of scraping for all the tags, it only scrapes for the type of tag specified in the rule by the tester. It then goes through this shorter list with a Lodash \texttt{forEach} loop verifying the existence of a \texttt{lang} attribute. On completion it reports the total number of tags it checked, the number of tags missing a \texttt{lang} attribute, and its conclusion on the result of the test, either true or false for pass or fail respectively. The return from the callback is a Boolean value, which should be interpreted as an integer value for the certification estimation score.

4.4.1.3 Screen Reader Support

Many of the WCAG guidelines are designed to make things standardized for accessibility tools, such as screen readers. These tools rely on elements in the HTML code to do their processing and know what to output. Screen readers in particular have to go through the text on a webpage linearly, which can be inconvenient if there is no way
to skip sections or menus. Since many of the WCAG guidelines are intended to make the
screen readers easier to use, including at least one rule with this in mind was logical.
Allowing the user to skip sections of content is an important capability of screen readers,
which has to be facilitated in the HTML code; checking for those skip links therefore
falls under content testing. In particular, a skip link to the main content that allows the
screen reader to skip over any menus and start directly at the meat of the page is a
minimum requirement [57].

Skip to Main Content Link Exists. A skip link is different from other links in that
it is a local link, to a point elsewhere on a particular webpage. Its URL is not in the
standard http/https format, it starts instead with a ‘#’. The rule to check that such a link
exists makes use of comparisons on the href attribute:

```
for ([1,1]*) link) {link.href='^main*'}
```

with a quantifier of [1,1] to show that only 1 link has to meet the condition of having its
href attribute be a link containing the substring ‘#main’, but there can be more links
that meet the criteria. There isn’t a standard way to define skip links, but this can be used
to check that skip links are defined uniformly within the page and the substring can
changed to reflect the standards for the given website. Additionally, link is used
instead of a as the tag name to make the rule content clearer, but is mapped to the <a>
tag as seen in the JSON breakdown below:

```
{quant: [‘1’,’*’],
tag: 'a',
attr: { href: 'main*'},
func: 'skipLinkExists'}
```
If the user was looking to have exactly one skip link, then using $1$ instead of a range is an option, as well as providing a range of $[1, 1]$. In this case, the * indicates that there is no upper bound on the number of links matching that criteria.

The function to perform this rule check is short and makes use of Request and Cheerio to get all the links from a webpage. It then uses a Lodash forEach loop to search the href attribute of each link for the substring `'#main'`, which indicates a skip link. It then reports if it found such a link or not, as well as returning that same Boolean value in its callback. The return value should be taken as an integer value, since it falls under a guideline for skipping blocks of content, an A level rule.

4.4.2 Site Structure Testing

This second area of focus looks at the site structure as a whole. It goes beyond just the individual pages and looks deeper into the site by traversing links. The ux-rule framework allows these sort of rules to be defined as well, in a similar manner to the content testing rules. It is the functions that support these types of rules that make use of the PhantomJS environment to do some preprocessing on the tag set returned. The rules in this area can be divided into two categories: external structure and internal structure, which are expanded on in the following sections.
4.4.2.1 External Structure

The external structure of a website refers to any interactions with the webservers that serve the individual pages up. The sort of information that could be found in this area is, for example, server return codes and the certificate for the site. The majority of the site structure testing rules fall under this category, and none of the rules therein link directly to any WCAG guidelines, but rather to basic website design standards and functionality.

**No Links go to 404 Page.** Good website design demands that none of the links on a page should point to a webpage that does not exist, whether within the site itself or to an external site [18]. While control can only be exercised on link destinations within the site, removing or changing any links that might go to external webpages that do not exist is the minimal level of correction expected in such cases. However, pages have numerous links that can be time consuming to visit individually, so by providing a rule to perform this type of check, *ux-rule* demonstrates its flexibility to rules outside of the WCAG guidelines while maintaining the same rule structure and components. The rule for this case still uses the *for* operator, the *all* quantifier and the *link* tag:

```
for (no link) { link.code=404 }
```

This allows the attribute of *code* to be added to the set that already exist for the *link* tag, rather than adding *code* as an attribute to the *href* for example, which would be similar to the Alt Text is Within Word Limit rule. The JSON breakdown of the rule
shows that the `code` object is placed within the `attrs` object, rather than a level deeper as the attribute of an attribute:

```json
{  
    quant: 'no',
    tag: 'a',
    attrs: {code: '404'},
    func: 'noPageHas404'
}
```

The JSON structure of the code is similar to the basic Alt Text Exists rule, although the types of checks being performed are different.

While all the previous functions made use of Request, Cheerio, and Lodash exclusively to deal with connections and scraping the HTML, this function uses Phridge and PhantomJS as well as Async to perform its operations. It does also use Request, but not Cheerio or Lodash. First the function sets up a sub-function to retrieve the server return code and check its value using Request and assigns the function to a variable. Next it uses Phridge to spawn a PhantomJS environment and open the webpage. Inside PhantomJS, all the links in the page are retrieved using the HTML DOM Document Object and looped through to pull out all the links that are valid http/https links, not local links or CSS links. This complete list of good links gets returned in the callback to Phridge and Node and the PhantomJS environment is closed and disposed of. Rather than using Lodash loops to iterate over the set of links, this function uses Async.map, which takes a set and a function to apply the individual elements to synchronously, and supports a callback function. The Lodash loops do not support callbacks, which are necessary in this case to ensure completion of the webpage connections. Async maps the list of links returned from PhantomJS to the sub-function specified above, calling that function with each link as the argument synchronously. This is done to prevent socket hang-ups and memory over-use. Async then passes an array of the return values to its
callback function, the same length as the links array passed to it. Inside the sub-function, if any links have a 404 return code, the code and the link are outputted, so the function only needs to check through the results array until it finds a value of false. Lastly, it reports its conclusions on the test, reporting true or false that no 404 pages were discovered. Its callback returns a Boolean, and since it is not a scoring rule, it should be interpreted in its Boolean form. It will catch the majority of 404 pages, but may miss some edge cases in which the server is misconfigured such that it serves up a custom 404 page, but because it can find that custom page, it returns a code indicating the page is there. The standard, pre-loaded 404 pages for a server are already configured properly and will be caught.

Limit on Number of Redirect Links. There are different types of redirect links out there, some that simply take the user from an http page to an https page, to long strings of redirect pages that can keep the user in the dark about their final destination. Since redirects of more than 1 step do obscure their final location, which could be a malicious page, and complicate page navigation, limiting the number of such links on a webpage, or even discovering the number of redirect links in general could prove useful. The indicator for a redirect page is a server return code in the 300’s, and isn’t apparent from the link itself, even identifying which links are redirects is a challenge without visiting each link and looking at that code. Unlike pages that return a 404 page, there often isn’t an easy way to tell that a page has redirected on the human end. This sort of rule can be tested under the ux-rule framework, just using the basic rule structure:

\[
\text{for } ([X,Y] \text{ link}) \{ \text{link.code}=[300,399] \}
\]
where $X$ and $Y$ are the limits on the acceptable quantity of these redirect links. It breaks down in the standard JSON format that the majority of the rules described share:

```
{quant: [X, Y],
tag: 'a',
attrs: {code: ['300', '399']},
func: 'redirectsWithinLimit'}
```

where $X$ and $Y$ are concrete integer values. Those range values are parsed into a global range variable and so are hard coded as [0,4] for the time being to facilitate testing.

Since this function also deals with server return codes, it follows the same pattern of execution as the No Links Go To 404 Page rule described in detail above. The major difference is in the sub-function that looks at the return codes. Request by default follows redirect links to the end of the chain or until the maximum jump limit is reached and returns that value as the server return code. Therefore, in order to detect the redirect links, Request is told to open the link and not follow redirects using the followRedirect option set to false. A running count is kept of the links that then have a return code between 300 and 399 inclusive, and returned as the callback for the sub-function into the Async results array. Checking the final value of that count against the range determines the result of test. The function reports the number of redirects it found and the conclusion about whether that number is within the specified range, either true or false. This callback also returns a Boolean value, to be interpreted as such, as it does not fall within any of the WCAG certification requirements.

*Check for Certificate Issues.* Certificates basically say that some steps have been taken to ensure that a webpage is legitimate and are signed by a trusted Certificate Authority. There are also self-signed certificates, which are not as trustworthy because it
is simply the signer declaring that they are who they claim to be. Sometimes these certificates can lose trust, or not be set up properly, which would be good to know for checking the basic functionality of a website. This can occur when the certificate authority becomes untrusted, if the site name on the certificate doesn’t match the site, or a certificate expires [60]. By adding `cert` as an attribute to a `link` tag, the following rule supports a test of certificate validity:

```
for (all link) {link.cert = true}
```

In this case, either the certificate can be valid or not, there is not middle ground, so assigning a Boolean value to the `cert` attribute best describes the condition. This rule also breaks down into the standard JSON form for the rules:

```
{quant: 'all',
tag: 'a',
attrs: {cert: 'true'},
func: 'noPageHasBadCert'}
```

where again the `link` tag is mapped to the `<a>` tag.

The main chunk of this code is identical to that of the previous two functions, using Phridge and PhantomJS to obtain a complete list of http/https links to check against and using Async to run those links through the sub-function. The reason for including this repeating chunk of code is that there may be other preprocessing conditions that need to happen in PhantomJS specific to individual functions, although that is not the case in these external structure testing functions. Regardless, it is the sub-function that holds the core of the differences. This sub-function uses Request to make a connection and checks if it receives any errors pertaining to the certificate, at which point it outputs the error and the URL that caused it. Since the webserver hosting the testing webpages has a good certificate, a forced [CERT_UNTRUSTED] error is triggered for certain cases within the
sub-function code. The sub-function returns a value of false if it encounters a certificate error, otherwise it return a value of true in the Async results array. The function checks for any false results in that array and reports its conclusion on whether no bad certificates were discovered, true or false, which is also returned through the callback and should be taken as Boolean values, since again it isn’t a scoring test.

4.4.2.2 Internal Structure

In this case, internal structure refers to the structure within the website, and how its pages are organized. Included in this is anything pertaining to site navigation and structure. There are a number of WCAG guidelines that deal with this area, but one selected for this proof of concept deals with the navigation menus, using both the HTML5 nav tags and ARIA roles attached to ul tags [34]. Specifically, this rule tests for navigation menu consistency between pages.

Navigation Menu Consistency. Maintaining a consistent navigation mechanism between pages of a site is important for continuity and for establishing a constant structure to follow no matter where the user is within the site. On the code side as well, the manner in which the navigation mechanism is set up should also be consistent, for debugging and to aid the programmers if they take over the code from someone else. This rule checks for both:

```java
for (all <page_1, page_2>) {
  page_1:nav.structure == page_2:nav.structure and page_1:nav.menu == page_2:nav.menu
}
```
A navigation menu is an attribute of the page and there exist actual nav tags that can be used to define such a section. Since there are no other requirements, as in the other consistency rule for image alternate text, there is no need for an additional conditional statement, only the test for equality of the navigation attribute, as seen in the JSON breakdown:

```json
{quant: 'all', tag: 'html', and: {expr1: {'==',
    Tag1: {name: {parent: 'page_1', child: 'nav'},
           att: 'structure'},
    Tag2: {name: {parent: 'page_2', child: 'nav'},
           att: 'structure'}},
    expr2: {'==', Tag1: {name: {parent: 'page_1',
                           child: 'nav'}, att: 'menu'},
           Tag2: {name: {parent: 'page_2', child: 'nav'},
                  att: 'menu'}}}, func: 'allNavMenuConsistent'}
```

In this case the main operation with in the rule is the and, so it becomes the primary object in the part of the breakdown. Also, since the comparison is being done directly between two objects, and not to a discrete value of some sort, only the information about the objects in question is stored, not their values.

The function that executes on this rule checks for both code and content consistency of the navigation menus. It looks at the main page passed in and its matter of navigation, either nav tags or ul tags with a “menu”, “menubar”, or “navigation” role specified. Since those tags are used by screen readers to identify navigation menus, they should be used and are also easy to copy between pages when building them. Using that as the basis, it then uses Phridge and PhantomJS to retrieve all the links that share the same core as that page; in other words, it chooses only links that belong to that website, and ignores all external links, which wouldn’t match the site navigation anyway. After Phridge and PhantomJS are closed, the list is trimmed of any duplicates and then checked
against the main page navigation. It first looks at whether the code mechanism is the same. If it isn’t, then the navigation is already inconsistent. If the forms of navigation are the same, the next check is that the contents are the same by collecting all the links in the menus from the other pages and comparing them to the links from the starting page menu. This also checks that the order of the links is consistent as well. If both tests are passed, then the navigation is considered consistent and available to screen readers under ARIA specifications. It reports the number of pages it compared to the main page, the number that it calls inconsistent, and its conclusion on whether all the pages are consistent.

4.4.3 Input Testing

Typically webpages will have at least one source of user input, such a search box. These input fields are a point of access for a number of code injection attacks and should have their input checked in some way. Testing for these checks can be useful and knowing where they are missing before something malicious takes advantage can be invaluable. While most of these checks happen behind the scenes and the user never sees the results of checks, there is one particular set of inputs that the user does see some level of validation for: forms.
4.4.3.1 Embedded Forms

Online forms are slowly taking the place of paper forms. They can be submitted almost instantaneously and their inputs can be checked and validated, providing feedback, before final submission. There is an accessibility standard concerning form input feedback under the ARIA guidelines. These guidelines focus on making web content more accessible to screen readers in particular. Screen readers cannot easily detect the way that some errors in forms are displayed, making their interaction with forms stressful and somewhat limited. ARIA introduced the concept of roles and a number of additional attributes for HTML tags, both of which are now the primary way that screen readers can detect form error events [61][57].

**ARIA Compliance on Forms.** The other rules discussed in this thesis have all had more discrete values and comparisons. This rule describes a property, so the way to express that in the *ux-rule* syntax is as follows:

```plaintext
for (all form) { form.isARIA = true }
```

Using “isSomething” indicates a property, and it is stored as an attribute of the form tag with a Boolean value associated, since a webpage can only either have a property or not. A form tag would be the way to specify input, so that the input can be sent somewhere, and as a tag, it can have attributes. The JSON breakdown of this sort of rule is similar to the rest:

```json
{quant: 'all',
tag: 'form',
attrs: { isARIA: 'true' },
func: 'allFormsAreAria'}
```
The property of isARIA is stored as an attribute of the tag, where it fits.

The function executing this rule uses Phridge and PhantomJS to get a list of both the input tags and tags that have a class of “error”. This is because there needs to be both connected to each other for the form to be ARIA compliant. Within PhantomJS it first checks that all the error class tags have a role of alert. If any are missing the role, it stores the tag information and returns an array of 4 values, 3 being -2 to indicate missing information and the last value being the stored tag information. If all the error class tags have the appropriate role, it then checks that all the input tags have an `aria-describedby` attribute. If there are no inputs, it passes an array of -1’s and the information that there is no form on the page. If any inputs are missing the attribute, then the array of -2 and the tag information is passed on. And if both sets of tags meet their requirements, they are then passed out of PhantomJS. Back in Node, the function starts with basic tests on the lengths of the arrays. If there are more errors than inputs, it reports that and both numbers, and ends the test with that conclusion. After that, it checks that the value of `aria-describedby` matches one of the error id values. If there were any input tags that didn’t find a match, they are reported. Any other tags missing information are reported at that point, followed by the conclusion as to whether all the form inputs are ARIA compliant. In the event that there are no inputs, instead of a conclusion, it reports that the test is inconclusive since there is nothing to test. Unlike the other function using PhantomJS and Phridge, it does not use Request, Async, and Lodash functions. It does not need to check any other pages, so Request is not needed. There is no need to map the resulting arrays to any functions, so Async isn’t used. Also, the results array passed back from PhantomJS and Phridge is an array of arrays, which would be much more
complicated to map directly. The Lodash functions are more efficient because they run in parallel, but for this case, a regular for loop was used wherever looping is required, so that the inside of the loop is executed in sequence the same way every time, which is not the case with the Lodash functions. The return value in its callback is a Boolean, which should be interpreted as an integer for scoring purposes. If a page does not have a form, this value is true, or 1, since the guideline is met by the lack of the form object.

4.5 Testing Procedure

In order to demonstrate that the results obtained from *ux-rule* are accurate and can be trusted, testing was done on a set of constructed websites that have full rule coverage as well as a small selection of existing websites. The HTML code of these constructed sites is very basic, so that human verification of their properties is easier in regards to establishing ground truth. The full code for the sites can be found in Appendix D and Table 4 reports the ground truth for the tests and sites.

<table>
<thead>
<tr>
<th>Rules</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>allImageTagHasAltAttribute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allImageTagMeetsCondition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allImageTagIsConsistent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allImageAltTextWithinWordLimit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allHasLang</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>someSectionsHaveLang</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allSectionsHaveLang</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noHas404</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redirectsWithinLimit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noHasBadCert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allMenuConsistent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allFormsAreAria</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skipLinkExists</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Ground Truth Chart for Testing Sites. The tests are listed on the left of the chart, while the site numbers are used across the top. A passing test is marked with a 'p' as well as a green square. Failing tests are marked with a 'f' and a red square. Tests a particular site does not undergo are greyed out.
Since the set of testing sites would be unreasonably large if every possible combination of passing and failing tests were accounted for, the tests are grouped into sets of similar rules and the combinations are tested fully within that group. Testing sites 1 and 23 have full test coverage, with all passing and all failing respectively. As this is the case, there is no need to have a full pass or full fail test site within the groups. Some combinations cannot be tested since failing one test may keep any of the other tests from passing as well, such as Image Alt Text Existence and Some Sections Have Language. Failing these tests means that none of the other alternate text tests can pass for the first and All Sections Have Language for the second. The code for test sites 1, 17-22, and 23 comes from an example on the ARIA wiki [28].

Included with the ux-rule code is a bash script that runs through all the test cases. The JavaScript code for each case consists of a URL and a call with that URL to ux-rule.check(), as can be seen in the following example:

```javascript
var ux = require('../../lib');
var cu = 'https://csel.cs.colorado.edu/~erdu8260/site1
ux.check(cu, 'run(all Tests)')
```

The rule passed in follows the ux-rule syntax for running groups of rules, since that is exactly what each of tests cases are doing. The next section discusses the results of each of the test cases.

### 4.6 Results

The results for each of the test cases is better discussed within the context of their testing group, so this section is further broken down into sections by those groups. First is a look at the Alt Text Group, followed by the Language and External Structure Groups.
The remaining cases are split between the Miscellaneous Group and the Full Coverage Group. In all of these groups, even if a website fails a test, the tool still passes its test of correctness since that failure was expected under the ground truth for that site. The final section lists the results of testing on existing websites outside the constructed set.

### 4.6.1 Alt Text Group

This group provides coverage of all the image alternate text rules, which covers test cases 2-8. Table 5 shows the final conclusions of each rule run on the test cases.

These conclusions are Booleans, and a true result always reflects a pass.

<table>
<thead>
<tr>
<th>Rules \ Test Case</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt Text Exists</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Alt Text Meets RegEx Condition</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text is Consistent</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Alt Text is Within Word Limit</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**Table 5: Results From Alt Text Group.** Green squares for true results and red squares for false.

For each test case, all the images have an alternate text attribute, otherwise none of the other tests could ever return true. The regular expression that is tested for all these cases is a basic definition of a full sentence in the English language.

Cases 3, 6 and 7 have alternate text that are all sentences, and so return true for the Alt Text Meets RegEx Condition rule. Cases 4, 6, and 8 have alternate text that is consistent between the same images, and return true for that rule. Cases 5, 7, and 8 have
alternate text with between 2 and 4 words inclusive, passing that rule and returning true. This pattern of results matches with the section of the ground truth chart for this group.

4.6.2 Language Group

Consisting of 2 test cases, this group covers all the rules pertaining to language on a webpage. In Table 6 are the results of running the language rules on this set of test cases.

<table>
<thead>
<tr>
<th>Rules \ Test Cases</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language for Entire Page</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Some Sections Have Language</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>All Sections Have Language</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 6: Results From Language Group. Green squares for true results and red squares for false.

Cases 9 and 10 both have a language specified for the whole page, and therefore return true for that rule. Case 10 also has some sections that specify a language, returning true for the Some Sections Have Language rule. Neither of these test cases return a true result for the All Sections Have Language, since it would mean that both of the other tests are also true and is covered under test case 1. The pattern of these results matches with the ground truth chart, showing the correctness of the functions.
4.6.3 External Structure Group

The focus of this group is on the External Structure category of rules under the Site Structure category, which covers test cases 11-16. The results from these test cases are enumerated in Table 7.

<table>
<thead>
<tr>
<th>Rules \ Test Cases</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Links got to 404 Page</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Limit on Number of Redirect Links</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Check for Certificate Issues</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Table 7: Results From External Structure Group. Green squares for true results and red squares for false.

Test cases 11, 14 and 16 have all their links going to locations that exist and return true for the No Links go to 404 Page rule. Cases 12, 14, and 15 have between 0 and 4 redirect links, so they return true under the Limit on Number of Redirect Links rule. Cases 11, 12, and 14 are hard-coded cases to trigger a certificate error, and return false for that rule, where all the other cases return true since the hosting server has a valid certificate (true at the time of testing). Like with the above groups, these results match the ground truth chart.
4.6.4 Miscellaneous Group

This group of test cases, 17-22, fills out the remaining rules: Navigation Menu Consistency, ARIA Compliance on Forms, and Skip to Main Content Link Exists. Table 8 shows their results.

<table>
<thead>
<tr>
<th>Rules \ Test Cases</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation Menu Consistency</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>ARIA Compliance on Forms</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Skip to Main Content Link Exists</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Table 8: Miscellaneous Group Results. Green squares for true results and red squares for false.

Cases 17, 20, and 22 have `nav` tags with consistent links to each other within their code, so they return a true result where cases 18, 19, and 21 do not. Cases 18, 20, and 21 have forms that meet the ARIA standards, and return true for that rule. Cases 19, 21, and 22 have local links to the main content, which means they return a true result for the Skip to Main Content Link Exists rule. These results are consistent with what the ground truth table lists for expectations.

4.6.5 Full Coverage Group

The test cases in this group, 1 and 23, run through all the rules, rather than a small subset of them. The results from these cases are in Table 9 below.
<table>
<thead>
<tr>
<th>Rules \ Test Cases</th>
<th>1</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt Text Exists</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text Meets RegEx Condition</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text is Consistent</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text is Within Word Limit</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Language for Entire Page</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Some Sections Have Language</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>All Sections Have Language</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>No Links got to 404 Page</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Limit on Number of Redirect Links</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Check for Certificate Issues</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Navigation Menu Consistency</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>ARIA Compliance on Forms</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Skip to Main Content Link Exists</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 9: Full Coverage Group Results. Green squares for true results and red squares for false.

Case 1 meets all the conditions for all the rules, and so returns true results for each of them. Case 23 meets none of the requirements, and returns false results for all the rules. These cases cover the all true and all false results of the other groups, which is why those conditions were not among the earlier test cases. These cases report results as expected from the ground truth chart.

4.6.6 Additional Sites

The suite of constructed sites described above are intended to show that the results from *ux-rule* are correct and can be trusted. To show that the framework performs as well on more complex pages, the tests for all the rules were performed on existing webpages. These sites have been assigned a letter code, to make showing the results more compact, which can be found in Table 10.
Table 10: Website to Letter Code Map. Gives the site name, URL used, and the letter code associated with each site.

The CU Boulder Homepage was used for familiarity and the NMT CS Homepage was used to show https functionality. The CNN Homepage was selected for the variety of content it presents and Google.fr was chosen as a foreign language website with an input field.

Running these sites through the all the rules currently in *ux-rule* produces the following results, as seen in Table 11.

<table>
<thead>
<tr>
<th>Website</th>
<th>URL</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU Boulder Homepage</td>
<td><a href="http://www.colorado.edu">http://www.colorado.edu</a></td>
<td>A</td>
</tr>
<tr>
<td>New Mexico Tech CS Homepage</td>
<td><a href="https://cs.nmt.edu">https://cs.nmt.edu</a></td>
<td>B</td>
</tr>
<tr>
<td>CNN Homepage</td>
<td><a href="http://www.cnn.com">http://www.cnn.com</a></td>
<td>C</td>
</tr>
<tr>
<td>Google.fr</td>
<td><a href="https://www.google.fr">https://www.google.fr</a></td>
<td>D</td>
</tr>
</tbody>
</table>

Table 11: Additional Website Results. Green squares indicate true results and red squares indicate false.

<table>
<thead>
<tr>
<th>Rules \ Test Cases</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt Text Exists</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text Meets RegEx Condition</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text is Consistent</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Alt Text is Within Word Limit</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Language for Entire Page</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Some Sections Have Language</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>All Sections Have Language</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>No Links got to 404 Page</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Limit on Number of Redirect Links</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Check for Certificate Issues</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Navigation Menu Consistency</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>ARIA Compliance on Forms</td>
<td>false</td>
<td>-</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Skip to Main Content Link Exists</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
The CU Boulder homepage (A) was the only one to not have consistent navigation. Google.fr (B) passed 4 of the tests. CNN (C) only passed 2 of the tests, one of which being the certificate test that all of the sites passed. The NMT Computer Science site (B) passed the most tests at 6, meeting the basic condition for image alt text existence where the other sites did not. It did not have a form or any input on the page, so it could not provide a conclusion for the ARIA Compliance on Forms rule.

4.7 Limitations of *ux-rule*

In its current state, there are some limitations on the functionality of *ux-rule*. It does not have full rule coverage, so its certification testing report will be unable to produce proper estimates. This is because it is only checking a few rules out of the total number of possibly testable guidelines, meaning the percentage value it returns will never be above a certain value. It also doesn’t have full coverage of dynamic site for all the rules. Only certain rules currently have the capability to retrieve the dynamic content of a page, but the tool has shown that JavaScript libraries have the ability to retrieve the rendered HTML of page effectively. As mentioned in the section about rule construction, it doesn’t have a complete parser for the rule language, as a result of time constraints. Parsing the rule language being an easy problem to solve, it was prioritized low in the development of the tool. There is an approach in mind, however, making use of imperative code compiler techniques to parse it out into an abstract syntax tree (AST). In spite of these limitations, as a proof of concept it serves its purpose effectively.
4.8 Conclusions

The *ux-rule* framework described within this thesis makes a number of contributions to automated accessibility testing. Its rule specification language is flexible enough to allow running custom groups of rules, as well as individual custom rules that go beyond those specified directly in guidelines, but users have expressed would make a page more accessible. Testing can be geared towards an individual disability group in addition to the overall testing for all the accessibility guidelines. This rule language also allows for creating other rules outside of the accessibility sphere, such as security testing, helping to make integration in the development process easier. Also, by separating out the rule specification from the execution, updates can be more easily applied when maintain the tool.

Having the rule language not based on XML makes the rules written in it easier to understand at a glance and makes constructing those rules with limited programming experience in specific languages possible. In the course of study, programmers will have learned at least one imperative language, on which the rule syntax is based, so it does not require any understanding of a specific language such as XML to understand how to build the rules. By also allowing the specification of groups of elements to use in the rule, more complex conditions can be expressed, such as tests for consistency, which can prove valuable when testing across larger sites, as well as help prevent regression errors.

By making it a command line tool, it can easily be scripted for running large numbers of pages for testing and produce output that is useful to a programmer in a data form that can parsed into a report for other personnel. It can also be run on a closed
network, and does not store or report any information about the page within the program or to the program developers, ensuring that the user’s privacy with regards to their website is respected. Its open source and free code means that it has the potential to reach a great number of users, to help those outside of large development environments to keep accessibility in mind, as well as integrate into corporate development processes.

JavaScript has many tools that help with processing webpages, and by using JavaScript for the back end of the framework there is support for handling both static and dynamic pages. It also then runs across multiple platforms, which avoids the portability limitation of many command line and desktop tools.

Through all of these factors, the ux-rule framework, both the rule definition language and the JavaScript back end, achieves its goals stated in Section 1.3. The overall tool caters to programmers, provides a flexible rule language, and support the customization of rules beyond those pertaining to specific predetermined guidelines.

4.9 Future Work

In order to turn this proof of concept into a full capability toolkit, expanding the rule set to encompass all of the tests under the WCAG guidelines that can be either programatically existence checked or programatically determined is the first step. Once it reaches this level, it can be maintained and changed according to any changes in the guidelines. Since it is open source, the expected users, programmers, can add their own rules to test beyond the accessibility guidelines.
Creating a more generic back end would increase the power of the tool, such that any rules specified in the rule language can be executed with minimal to no back end editing required. Currently the back end is very specific to the rules created for the thesis, primarily to test the suitability of the JavaScript libraries and language itself to execute with enough range to handle each of the different rule cases. Handling the custom properties in a generic back end would be the main challenge, but at the very least the amount of code required to add into the back end would be minimized if all the user needed to do is define a small function defining the custom property in a way relatable to the code.

Other work might include making improvements on the certification estimation system, specifically on how things are scored, as well as adding additional types of certifications the toolkit can make estimations for, such as US 508 and other similar foreign certifications. Adding better spidering so that entire sites can be checked at once, instead of just individual pages for all the rules would also be a valuable addition.

More extensive testing of dynamic webpages is an important further step to bring out the full potential of the tool, although it may require reworking the framework to run in PhantomJS instead of Node to make use of the WebKit interactions with those dynamic pages. The capability is there, but needs to be integrated with the rest of the tool. Being able to test dynamic pages completely would put the tool above a large chunk of the existing tools. Going further to test more than just the HTML code as well would make it a more powerful tool. Adding CSS testing capabilities as well as integrating some programming language techniques to map out the executions of the JavaScript and AJAX code to determine if the scripting is valid, could make the tool
applicable to more areas of webpage design and execution. Since having a wide set of capabilities might make it fit better in design processes, those sort of abilities would add another step in that direction.

Right now, the tool only behaves as though it were a desktop browser, so expanding its capability to simulate other devices or programs, such as mobile devices or screen readers, could improve the accessibility on the many versions of the webpage. With mobile computing becoming extremely common, supporting accessibility on mobile versions of the webpages is just as important as having it on the main browser version.

The end reports being a limitation of existing tools, finding a way to provide possible fixes for accessibility issues, without insulting the programming knowledge of the webmasters or site maintainers, would make it really stand apart. As of now, it reports information that a programmer might find useful, but suggesting possible fixes would improve the report quality, as long as it doesn’t drastically increase report length, and help less experienced programmers, such as those building and maintaining a personal site (a target user group of ux-rule).
BIBLIOGRAPHY


This tutorial asks some questions about the rule you want to construct, to help you find the right syntax to express it. Since there are 2 different rule bases, there are 2 different sections on their construction.

1. **Are you trying to run multiple rules or just design a single rule?**

For multiple rules, look at Path 1. For a single rule, look at Path 2.

**Path 1**

In this case you want to use the *run* syntax. The quantifier you will use is *all*.

The last piece of the rule is the list of rules to run or the name of the rule group. Groups of rules can be hard coded under a name in the back end, or provided to the run command as a list of single rules. For example, if you wanted to run the A-level certification group, you would use this syntax, with *ARules* being the name of the group:

```
run (all ARules)
```

If you wanted to pass a list of rules to run directly, instead of a list hard coded in the back end, the syntax would look like this instead:

```
{ for(q O){... },
  for(q O){... },
  ...
  for(q O){... }
}
```
This syntax is listing a number of single rules between brackets and separated by commas. Path 2 contains the tutorial on constructing the individual rules that would be placed in the list.

**Path 2**

For building a single rule, the rule base to use is the *for* loop. It has 3 parts to it: a quantifier, an object or object group, and a body section. The first thing to do before starting to construct the rule, is to write it out in a single sentence of the form:

“For –quantifier- -object/objects-, -conditions-“

A few examples might be:

“for every input there is exactly one label linked to that input”

“for at least 1 <p> tag, there exists a lang attribute”

“for no links, the server code is 404”

“for between 2 and 20 links, the server code is between 300 and 399”

“for each pair of images, if the image sources are the same, then the image alternate text is the same, otherwise, the image alternate text is not the same”

Since there are 3 sections to the rule base, there are 3 sets of questions to help build each part. It will step you though finding your quantifier, determining your object(s), and figuring out the body of the rule. These questions are based on your sentence representation.
**Quantifier**

2. Did you use “all” or “every” in your rule?
   Yes: Then your quantifier is *all*.

3. Did you use “at least 1” or “some”?
   Yes: Then your quantifier is *some*.

4. Did you use “more than half” or “most”? 
   Yes: Then your quantifier is *most*.

5. Did you use “no” or “none”? 
   Yes: Then your quantifier is *no*.

6. Did you use “exactly \( x \)” or \( x \), where \( x \) is a number? 
   Yes: Then your quantifier is \( x \).

7. Did you use “between \( x \) and \( y \)”, where \( x \) and \( y \) are numbers? 
   Yes: Then your quantifier is \( [x,y] \).

8. Did you use “at least \( x \)”, where \( x \) is a number greater than 1? 
   Yes: Then your quantifier is \( [x,*] \).

9. Did you use “at most \( y \)”, where \( y \) is a number? 
   Yes: Then your quantifier is \( [* ,y] \).

10. Do you have more than one of these quantifiers in your rule? 
    Yes: Then you will have additional *for* loops within the body. If you have 2 quantifier, then you will have 1 loop within the block, if you have 3 quantifiers, then you will have 2 loops within the block, and so on. There’s more information on how to handle this case in the section about constructing the body.
Objects

11. Does the number of objects (tags or attributes) match the number of quantifiers?
Yes: Then you’re only dealing with a single object per `for` loop, the tag or attribute.

12. Did you use “pair” or “triple”, or the objects outnumber the quantifiers?
Yes: then your object is a group of objects enclosed in “<” and “>”.

13. Is your object an attribute or property?
Yes: Then you will use the `tag.attribute`, `tag.property`, or `tag.attribute.property` syntax.

14. Is your object a nested tag, i.e. links in a list, nav menu on a page?
Yes: Then you will use the `parent : child` syntax.

15. Is your object just a tag?
Yes: Then your object is just the tag name.

Body

These questions deal with the main condition of the rule, generally after the first comma in your sentence.

16. Did you use the words “if” and “then”, but not “otherwise” or “else”?
Yes: Then you will probably need an `if-then` statement.

17. Did you use the words “if”, “then” and “otherwise” or “else”?
Yes: Then you will probably need and `if-then-else` statement.

18. Did you use the word “and”?
Yes: Then you probably need the `and` statement.

19. Did you use the word “or”?
Yes: Then you probably need the `or` statement.
20. Did you use the word “not”?
Yes: Then you probably need the not statement.

21. Are you doing any comparisons directly between tag values, attribute values, or property values? i.e. “image alternate text is the same”
Yes: Then you will need to use \( \equiv \) as your comparison operator.

22. Are you comparing an attribute or property value to a string?
Yes: Then you should use “RegEx”.

23. Are you comparing an attribute or property value to a Boolean value?
Yes: Then you should use = true or = false.

24. Are you comparing an attribute or property value to a number?
Yes: Then you should use any of the following depending on the comparison: =, <, <=, >, or >=.

25. Are you comparing an attribute or property value to a range?
Yes: Then you should use = [x,y].

Building a Rule
To demonstrate how to put everything together, this part of the tutorial will step you through deconstructing a rule sentence and creating the rule in the ux-rule syntax. The rule sentence we’re going to use is:

“for every input there is exactly one label linked to that input”

It’s a single rule, so we’re going to use the for syntax, as per question 1. The next step is to identify the quantifier(s). This rule has both “every” and “exactly one”, which according to questions 2 and 6 tells us the quantifiers are all and 1. Now that we have the
quantifiers, we need the objects the rule is acting on. There are 2 objects, “input” and “label.” Since there are 2 objects and 2 quantifiers, questions 10 and 11 tell us that we will need multiple \texttt{for} loops and that we only have 1 object per loop, so we don’t need to specify a group of objects.

The last thing we need is the meat of the rule, the main conditional. Labels are linked to inputs with the \texttt{name} attribute of the input tag and the \texttt{for} attribute of the label tag. If they are the same value, then the input and label are linked. This is a direct comparison between two attributes, so question 21 says that we need to use \texttt{==} as our comparison operator and question 13 gives us the way to write our objects: \texttt{input.name} and \texttt{label.for}.

Now comes putting it all together. We know we have 2 \texttt{for} loops, so let’s put those down:

\begin{verbatim}
for( ){ for ( ){ })
\end{verbatim}

Then let’s fill in the quantifiers and their objects:

\begin{verbatim}
for( all input ){ for ( 1 label ){ })
\end{verbatim}

Lastly, fill in the comparison we have:

\begin{verbatim}
for( all input ){ for ( 1 label ){ input.name == label.for })
\end{verbatim}

And, there’s our final rule, the one we would pass to the \texttt{ux-rule check} command.
APPENDIX B

RULES AND JSON BREAKDOWN

Rules

Example Rules for Implemented Functions

Certification Rules:
- \texttt{run(all ARules)} run all A level certification rules
- \texttt{run(all AARules)} run all AA level certification rules
- \texttt{run(all AAARules)} run all AAA level certification rules
- \texttt{run(all NIRules)} run all NI level certification rules
- \texttt{run(all rules)} run all sets of certification rules
- \texttt{run(all Tests)} run all rules
- \texttt{run(all ImgAlt)} run all image alt text rules
- \texttt{run(all Lang)} run all language rules
- \texttt{run(all Site)} run all site external structure rules
- \texttt{run(all Other)} run all miscellaneous rules

Other rules:

All images have alt text
\begin{verbatim}
for (all img) { img.alt="" }
\end{verbatim}

All images have alt text meeting RegExp condition
* uses RegExp for text not being ‘img’ or ‘image’
\begin{verbatim}
for (all img) { img.alt="^!(img|image)$." }
\end{verbatim}

Image alt text is consistent
\begin{verbatim}
for (all <img_1, img_2>) { if (img_1.src == img_2.src) then (img_1.alt == img_2.alt) else (not img_1.alt == img_2.alt)}
\end{verbatim}

Image alt text has between [x,y] words
* x = 2, y = 4
\begin{verbatim}
for (all img) { img.alt.words=[2,4] }
\end{verbatim}

Can an overall page language be determined
\begin{verbatim}
for (all page) { page.lang="" }
\end{verbatim}

Some pages have section languages specified
* uses p as the section definition
\begin{verbatim}
for (some p) { p.lang="" }
\end{verbatim}
All instance of 'tag' have a language specified
* uses p as the tag
for (all p) { p.lang="+" }  

Check for skip to main content link
for ([1,]* link) { link.href="#main" }  

No link goes to 404 page
for (no link) { link.code=404 }  

Number of redirects within [x,y], i.e. [0,4] means >= 0 and <= 4 redirects allowed for passing
*x = 0, y = 4
for ([0,4] link) { link.code=[300,399] }  

Check for certificate errors
for (all link) { link.cert=true }  

Check for nav menu consistency
for (all <page_1, page_2>) { page_1:nav.structure == page_2:nav.structure and page_1:nav.menu == page_2:nav.menu}  

Check for ARIA compliance in forms
for (all form) { form.isARIA=true }  

**JSON Breakdown**

```json
{ rules: [  
    {run: [  
        {quant: 'all', tag: 'ARules', func: 'allARules'},  
        {quant: 'all', tag: 'AARules', func: 'allAARules'},  
        {quant: 'all', tag: 'AAARules',  
          func: 'allAAARules'},  
        {quant: 'all', tag: 'NIRules', func: 'allNIRules'},  
        {quant: 'all', tag: 'rules', func: 'allRules'}  
        {quant: 'all', tag: 'Tests', func: 'allTests'}  
        {quant: 'all', tag: 'ImgAlt',  
          func: 'allImgAltTests'}  
        {quant: 'all', tag: 'Lang', func: 'allLangTests'}  
        {quant: 'all', tag: 'Site', func: 'allSiteTests'}  
        {quant: 'all', tag: 'Other', func: 'allOtherTests'}  
    ]}  
]}
```
{for: [
    {quant: 'all', tag: 'img', attrs: { alt: '+'},
        func: 'allImageTagHasAltAttribute'},
    {quant: 'all', tag: 'img', attrs: {
        alt: CONDITION},
        func: 'allImageTagAltMeetsCondition'},
    {quant: 'all', tag: 'img', cond: {
        if: {'==', {Tag1: {name: 'img_1', att: 'src'},
                       Tag2: {name: 'img_2', att: 'src'}}},
        then: {'==', {Tag1: {name: 'img_1', att: 'alt'},
                      Tag2: {name: 'img_2', att: 'alt'}}},
        else: {not: {'==', {Tag1: {name: 'img_1', att: 'alt'},
                       Tag2: {name: 'img_1', att: 'alt'}}}},
        func: 'allImageTagAltIsConsistent'},
    {quant: 'all', tag: 'img', attrs: { alt: {words: [X,Y]}},
        func: 'allImageAltTextWithinWordLimit'},
    {quant: 'all', tag: 'html', attrs: { lang: '+'},
        func: 'allPageHasLang'},
    {quant: 'some', tag: TAG, attrs: { lang: '+'},
        func: 'someSectionsHaveLang'},
    {quant: 'all', tag: TAG, attrs: { lang: '+'},
        func: 'allSectionsHaveLang'},
    {quant: [1,'*'], tag: 'a', attrs: {
        href: '#main*'},
        func: 'skipLinkExists'},
    {quant: 'no', tag: 'a', attrs: {code: '404'},
        func: 'noPageHas404'},
    {quant: [X,Y], tag: 'a', attrs: {
        code: [300,399]},
        func: 'redirectsWithinLimit'},
    {quant: 'all', tag: 'a', attrs: {cert: 'true'},
        func: 'noPageHasBadCert'},
    {quant: 'all', tag: 'html', and: {expr1: {'==',
                              Tag1: {name: {parent: 'page_1', child: 'nav'},
                                      att: 'structure'},
                              Tag2: {name: {parent: 'page_2', child: 'nav'},
                                      att: 'structure'}},
        expr2: {'==', Tag1: {name: {parent: 'page_1',
                                   child: 'nav'}, att: 'menu'},
                Tag2: {name: {parent: 'page_2', child: 'nav'},
                        att: 'menu'}},
        func: 'allNavMenuConsistent'},
    {quant: 'all', tag: 'form', attrs: {isARIA: 'true'},
        func: 'allFormsAreAria'}
    ]}
APPENDIX C

UX-RULE CODE

```javascript
var lib = module.exports = {};

var request = require('request');
var async = require('async');
var cheerio = require('cheerio');
var lodash = require('lodash');
var phantomjs = require('phantomjs');
var phridge = require('phridge');
var fs = require('fs');

// full sentence condition
var condition = /["']?[A-Z][^?!]+\((?!\d)[^?!]+\)["']?\s['"]?[A-Z][^?!]+\s['"]?\d+["']?\+|
// character limit condition
var condition = /\w[3,8]$/;
// not src condition
var condition = /[?!\!\!(http|https|www)\.].*$/;
// not 'img' or 'img'
var condition = /[?!\!(img|image)].*$/;

var A = 26, AA = 13, AAA = 22, NI = 4;
var d = new Date();
var certReport = d.getMonth().toString()+'-'+d.getDate().toString()+
|+''+d.getFullYear().toString()+'json';
var estimate = [];
var tag = 'p';
var range = [2,4];

lib.check = function(html, rule){
  // parse rule
  var ast = parse(rule);

  // get the handler function of this rule
  var handler = lookupRuleHandler(ast);

  // evaluate the rule
  return handler(html, function(err, results){
    if(err){} //console.log(err))
  });

  // parse a rule into an ast (abstract syntax tree), which is
  // an internal representation we can use
  // e.g.,
  // rule:
  // <img alt=".*">
  // =>
  // {tag: 'img', attrs: { alt: '.*' }}
  //
  function parse(rule){
    //return {}
    return rule
  }
```
function lookupRuleHandler(ast)
{
    // hard-coded for now
    if (ast == 'run(all Tests)')
        return allTests;
    else if (ast == 'run(all ImgAlt)')
        return allImgAltTests;
    else if (ast == 'run(all Lang)')
        return allLangTests;
    else if (ast == 'run(all Site)')
        return allSiteTests;
    else if (ast == 'run(all Other)')
        return allOtherTests;
    else if (ast == 'for (all img) { img.alt = "*" }')
        return allImageTagHasAltAttribute;
    else
        return allTests;
}

/* functions to run all tests and return percentage of certification */
// run (ARules)
function allARules(url, callback){
    var overall = true;
    var Acnt = 0;
    allImageTagHasAltAttribute(url, function(val){});
    noPageHas404(url, function(val){});
    allPageHasLang(url);
    allRules();
    allARules();
    allImageTagAltMeetsCondition();
    allImageTagAltIsConsistent();
    allImageAltTextWithinWordLimit();
    allSectionsHaveLang();
    skipLinkExists();
    redirectsWithinLimit();
    allNavMenuConsistent();
    allFormsAreAria();
    callback(null, overall);
}

Acnt += val;
console.log("Running A Certification Rules...");
allImageTagHasAltAttribute(url, function(val){
    Acnt += val;
    var est = (Acnt/Math.floor((Acnt/A)*1000))/10."%";
    estimate[estimate.length] = est;
    writeFile(outRep, estimate, function(err) {
        console.log(err)
    });
    callback(null, overall);});
```javascript
});

// Acnt = Math.random() * A;

}).

// run (AAARules)
function allAAARules(url, callback) {
    var AAcnt = 0;
    console.log("Running AA Certification Rules...");
    AAcnt = Math.random() * AA;
    var est = {AACert: Math.floor((AAcnt/AA)*1000)/10+"%"};
    estimate[estimate.length] = est;
    jf.writeFile(outRep, estimate, function(err) {
        console.log(err)
    })
    callback(null, AAcnt);
}

// run (AAARules)
function allAAARules(url, callback) {
    var AAAcnt = 0;
    console.log("Running AAA Certification Rules...");
    AAAcnt = Math.random() * AAA;
    var est = {AAACert: Math.floor((AAAcnt/AAA)*1000)/10+"%"};
    estimate[estimate.length] = est;
    jf.writeFile(outRep, estimate, function(err) {
        console.log(err)
    })
    callback(null, AAAcnt);
}

// run (NIARules)
function allNIARules(url, callback) {
    var NICnt = 0;
    console.log("Running Non-Interference Certification Rules...");
    NICnt = Math.random() * NI;
    var est = {NICert: (NICnt/NI)+"%"};
    estimate[estimate.length] = est;
    callback(null, NICnt);
}

// run (all Rules)
function allRules(url, callback) {
    console.log(url);
    allARules(url, function(err, step){
    allAAARules(url, function(err, step){
    allNIARules(url, function(err, step){
    });});});
}

// runs all tests (including those not required for certification)
// run (all Tests)
function allTests(url, callback) {
    console.log("Running all tests...");
    allImageTagHasAltAttribute(url, function(err, step){
    allImageTagAltMeetsCondition(url, function(err, step){
    allImageTagAltIsConsistent(url, function(err, step) {
```
```javascript
allImageAltTextWithinWordLimit(url, function(err, step){
    allPageHasLang(url, function(err,step){
        someSectionsHaveLang(url, function(err, step){
            allSectionsHaveLang(url, function(err, step){
                skipLinkExists(url, function(err, step) {
                    noPageHas404(url, function(err, step) {
                        redirectsWithinLimit(url, function(err, step) {
                            noPageHasBadCert(url, function(err, step) {
                                allNavMenuConsistent(url, function(err, step) {
                                    allFormsAreAria(url, function(err,step) {
                                        console.log("Tests complete.");
                                    });});});});});});});});});});});});
}

//run(all ImgAlt)

function allImgAltTests(url, callback){
    console.log("Running all tests... ");
    allImageTagHasAltAttribute(url, function(err, step){
        allImageTagAltMeetsCondition(url, function(err, step){
            allImageTagAltIsConsistent(url, function(err, step){
                allImageAltTextWithinWordLimit(url, function(err, step){
                    console.log("Tests complete.");
                });});});});});
}

//run(all Lang)

function allLangTests(url, callback){
    console.log("Running all tests...");
    allPageHasLang(url, function(err,step){
        someSectionsHaveLang(url, function(err, step){
            allSectionsHaveLang(url, function(err, step){
                console.log("Tests complete.");
            });});});
}

//run(all Site)

function allSiteTests(url, callback){
    console.log("Running all tests...");
    noPageHas404(url, function(err, step) {
        redirectsWithinLimit(url, function(err, step) {
            noPageHasBadCert(url, function(err, step) {
                console.log("Tests complete.");
            });});});
}

//run(all Other)

function allOtherTests(url, callback){
    console.log("Running all tests...");
    skipLinkExists(url, function(err, step) {
        console.log("after skip: ";step);
    });
        allNavMenuConsistent(url, function(err, step) {
            console.log("after nav: ";step);
        });
        allFormsAreAria(url, function(err,step) {
            console.log("after aria: ";step);
        });
        console.log("Tests complete.");
    });});});
```
function allImageTagsHaveAltAttribute(url, callback) {
  console.log("checking for image alt text");
  var ret = request.get({url: url, rejectUnauthorized: false}, function(err, resp, html) {
    if (err)
      { console.log(err);
        callback(err);
      }
    else
      {
        // console.log(html);
        $ = cheerio.load(html)
        imgs = $('img').toArray()
        // console.log(imgs);
        var missCnt = 0;
        var imgCnt = 0;
        var ret = lodash.forEach(imgs, function (img) {
          // console.log(img.attribs.alt)
          imgAlt = img.attribs.alt
          imgCnt += 1;
          // if the alt text isn't there or is empty, return false
          if ([typeof imgAlt == 'undefined' || imgAlt == ''])
            { console.log("FALSE");
              missCnt += 1;
            }
          return true;
        })
        if(missCnt == 0)
          ret = true;
        else
          ret = false;
        // console.log("TRUE");
        console.log("Number of images = " + imgCnt);
        console.log("Number missing alt text = " + missCnt);
        console.log("All images have alt text: " + ret + ",n");
        if (ret == true)
          { //Acnt += 0.5;
            callback(null, 0.5);
          }
        else
          callback(null, 0);
      });
  // callback(null, ret);
  // return ret; // all images have alt text
function allImageTagAltMeetsCondition(url, callback) {
    console.log("Checking for image alt text with conditions");
    var ret = request.get({url: url, rejectUnauthorized: false}, function(err, resp, html) {
        $ = cheerio.load(html)
        imgs = $('img').toArray()
        //console.log(imgs);
        var missCnt = 0;
        var failCnt = 0;
        var imgCnt = 0; //imgs.length;
        var ret = lodash.forEach(imgs, function (img) {
            //console.log(img.attribs.alt)
            imgAlt = img.attribs.alt
            imgCnt += 1;
            // if the alt text isn't there or is empty, return false
            if (typeof imgAlt === 'undefined' || imgAlt === '') {
                console.log("FALSE");
                console.log("\t\t"+img.attribs.src);
                missCnt += 1;
                //return false;
            }
            // if the alt text doesn't meet the condition, return false
            else if (condition.test(imgAlt) === false) {
                console.log("\t\t"+img.attribs.src);
                console.log("\t\t"+imgAlt);
                failCnt += 1;
                //return false;
            }
            else {
                return true;
            }
        }) //console.log("TRUE");
        if(failCnt === 0 && missCnt === 0)
            ret = true;
        else
            ret = false;
        console.log("Number of images = "+imgCnt);
        console.log("Number missing alt text = "+missCnt);
        console.log("Number failing test = "+failCnt);
        console.log("All images meet alt text condition: "+ret+"\n");
        if (ret === true) {
            Acnt += 0.5;
            callback(null, 0.5);
        } else
            callback(null, 0);
    });
}
```javascript
// For (all <img1, img2>){if img1 = img2 then alt1 = alt2}
function allImageTagAltIsConsistent(url, callback){
  console.log("\tChecking for image alt text consistency");
  var ret = request.get({url : url, rejectUnauthorized : false}, function(err, resp, html) {
    $ = cheerio.load(html)
    $img = $("img").toArray()
    var missCnt = 0;
    var failCnt = 0;
    var imgLength = $img.length;
    var ret = lodash.forEach(imgs, function(img) {
      var imgAlt = img.attr('alt');
      if (typeof imgAlt === 'undefined' || imgAlt.length === 0) {
        console.log("\t\t\tMissing: " + img.attr('src'));
        missCnt += 1;
      }
      return retIn = lodash.forEach(imgs, function(img2) {
        var img2Alt = img2.attr('alt');
        // if the alt text isn't there or is empty, return false
        if (typeof img2Alt === 'undefined' || img2Alt.length === 0) {
          return true;
        }
        // if the alt text doesn't meet the condition, return false
        else if (img1.attr('src') == img2.attr('src') && img1.attr('alt') === img2.attr('alt')) {
          console.log("\t\t\t" + img1.attr('src'));
          console.log("\t\t\t" + img1.attr('alt'));
          console.log("\t\t\t" + img2.attr('src'));
          failCnt += 1;
        }
        else if (img1.attr('alt') == img2.attr('alt') && img1.attr('src') != img2.attr('src')) {
          console.log("\t\t\t" + img1.attr('src'));
          console.log("\t\t\t" + img1.attr('alt'));
          console.log("\t\t\t" + img2.attr('src'));
          failCnt += 1;
        }
        return true;
      });
      // if the alt text isn't there or is empty, return false
      if (failCnt + missCnt == 0) {
        ret = true;
        return true;
      }
      // if the alt text doesn't meet the condition, return false
      else if (img1.attr('src') == img2.attr('src') && img1.attr('alt') != img2.attr('alt')) {
        console.log("\t\t\tNumber of images = " + imgs.length);
        console.log("\t\t\tNumber of images missing alt text = " + missCnt);
        console.log("\t\t\tNumber of inconsistencies found = " + failCnt);
        console.log("\t\t\tAll images are consistent: " + ret + "\n");
        if (ret == true) {
          callback(null, 0.5);
        } else {
          callback(null, 0);
        }
      }
    });
  });
}
```
```javascript
function allImageAltTextWithinWordLimit(url, callback) {
  range[0] = 0; //
  range[1] = 20; //
  console.log('Checking for image alt text with word limits: [' + range[0] + ', ' + range[1] + ']');
  var ret = request.get({url: url, rejectUnauthorized: false}, function(err, resp, html) {
    $ = cheerio.load(html)
    imgs = $('img').toArray()
    var missCnt = 0;
    var failCnt = 0;
    var imgCnt = 0; // imgs.length;
    var ret = lodash.forEach(imgs, function (img) {
      var imgAlt = img.attribs.alt
      imgCnt += 1;
      var altWords = [];
      // if the alt text isn't there or is empty, return false
      if (typeof imgAlt == 'undefined' || imgAlt == '') {
        console.log('\"alt\"=' + img.attribs.src);
        missCnt += 1;
      } else
        altWords = imgAlt.split(' ').length
      // if the alt text doesn't meet the condition, return false
      if (altWords < range[0] || altWords > range[1]) {
        console.log('\"alt\"=' + img.attribs.src);
        console.log('\"alt\"=' + imgAlt);
        failCnt += 1;
      } else
        ret = true;
    });
    if(failCnt == 0 && missCnt == 0)
      ret = true;
    else
      ret = false;
    console.log('Number of images = ' + imgCnt);
    console.log('Number missing alt text = ' + missCnt);
    console.log('Number not within word limit = ' + failCnt);
    console.log('All images meet alt text word limit: ' + ret + '\n');
    if (ret == true)
      callback(null, 0.5);
    else
      callback(null, 0);
  });
}
```
/* STATIC: Rule to check if the overall webpage’s language can be determined*/
//Naive function checking tags, NLP route for more complete processing
// for (all page) { (lang='*')

function allPageHasLang(url, callback) {
    console.log("\tChecking for page language...");
    var ret = request.get({url: url, rejectUnauthorized: false}, function(err, resp, html) {
        $ = cheerio.load(html)
        var langP = $("html").attr('lang');
        var isFound = true;
        if (langP !== undefined) {
            console.log("\tLanguage found on "+url+": "+langP+"\n");
        } else {
            isFound = false;
            console.log("\tLanguage not found on "+url+"\n");
        }
        callback(null, isFound);
        return isFound;
    });
}

/* check if at least some section have a language specified*/
// for (some tag) { (lang='*')

function someSectionsHaveLang(url, callback) {
    console.log("\tChecking for some section languages...");
    var ret = request.get({url: url, rejectUnauthorized: false}, function(err, resp, html) {
        $ = cheerio.load(html)
        var langP = $("p"),
        var langCnt = 0;
        var ret = lodash.forEach(langP, function (lang) {
            if($(lang).attr('lang') !== undefined)
                langCnt++;
        });
        var isFound = true;
        if (langCnt > 1) {
            console.log("\tLanguage found for some sections on "+url+"\n");
        } else {
            isFound = false;
            console.log("\tLanguage not found for any sections on "+url+"\n");
        }
        callback(null, isFound);
        return isFound;
    });
}
void allSectionsHaveLang(url, callback){
    console.log("Checking for lang on all <\:tag\:*> tags...");
    var ret = request.get({url : url, rejectUnauthorized : false}, function(err, resp, html) {
        $ = cheerio.load(html)
        var missCnt = 0;
        var totalCnt = 0;
        var langC = $('\:tag\:*');
        var isFound = lodash.forEach(langC, function (sec) {
            totalCnt++;
            if (sec.attribs.lang == undefined)
            {
                missCnt++;
            }
            return true;
        })
        if (missCnt == 0)
        {
            isFound = true;
        }
        else
        {
            isFound = false;
        }
        console.log("\nNumber of <\:tag\:*>: "+totalCnt);
        console.log("\nNumber of <\:tag\:*> missing lang: "+missCnt);
        console.log("\nAll <\:tag\:*> have lang attribute: "+isFound+"\n");
        callback(null, isFound);
    return isFound;
})
/
// skip link to main content exists
// for ([1,\:*] links) { link = '#main\:*' }
//
function skipLinkExists(url, callback) {
    console.log("Checking for skip to main content link...");
    var ret = request.get({url : url, rejectUnauthorized : false}, function(err, resp, html) {
        $ = cheerio.load(html)
        var skips = $('\:a\:*');
        var isFound = false;
        var isIn = lodash.forEach(skips, function (link) {
            href = link.attr('\:href\:*');
            if (href.indexOf('#main\:*') > -1 || href.indexOf('#skip') > -1)
            {
                isFound = true;
            }
            return true;
        })
        console.log("\nSkip to main link found: "+isFound+"\n");
        callback(null, isFound);
        return isFound;
})
function noPageHas404(url, callback) {
    console.log("Checking for 404 pages: " + url);
    var count = 0;

    /* set up code to get server status */
    var getCode = function(url, cb) {
        request.get({url: url, rejectUnauthorized: false}, function(error, response, body) {
            if (error)
                cb(error);
            else
                if (response.statusCode == 404)
                    console.log("404" + response.statusCode + "":" + url)
                    cb(null, false);
                else
                    cb(null, true);
        });
    }

    /* have phride spawn an instance of phantomjs */
    phrand.spawn()
        .then(function(phantom) {
            // open the webpage
            return phantom.openPage(url);
        })

    // pull all the links off the webpage
    .then(function(page) {
        // page.run(fn) runs fn inside PhantomJS
        return page.run(url, function(url) {
            // Here we're inside PhantomJS, so we can't reference variables in the scope
            // 'this' is an instance of PhantomJS' WebPage as returned by require('webpage').create()
            return this.evaluate(function () {
                var glinks = [document.URL];
                var links = document.links;
                // return links.length;
                for (var i=0; i< links.length; i++) {
                    var url = '+' + links[i].href;
                    if(url.indexOf("http") > -1 && url.indexOf('#') == -1)
                        glinks[glinks.length] = '+' + url;
                }
                return glinks;
                // return document.querySelectorAll("h1").innerText;
            });
        });
    });
}
```javascript
// phridge.disposeAll() exits cleanly all previously created child processes.
// This should be called in any case to clean up everything.
finally(phridge.disposeAll)

// we've got the links, now check them all
// (even if main page is 404, still helps to know if the error page has broken links)
done(function (links) {
    //console.log(links);
    async.map(links, getCode, function(err, results) {
        if(err)
        {
            console.log(err);
        }
        else
        {
            //console.log(results);
            var res = true;
            for(var i = 0; i < results.length;i++){
                if (results[i]===false)
                {
                    res = false;
                    break;
                }
            }
            console.log("\tNo 404 pages discovered: "+res+"\n");
            callback(null, res);
        }
    }, function (err) {
        // Don't forget to handle errors
        // In this case we're just throwing it
        throw err;
    });
    //return check;
}

// limit redirects
// for (let [x,y] links) { link <= 3. }
function redirectsWithinLimit(url, callback) {
    var count = 0;
    range[0] = 0;
    range[1] = 4;
    console.log("\tChecking for ["+range[0]+","+range[1]+"] redirect pages...");
    //set up code to get server status/
    var getcode = function(url, cb)
    {
        request.get(url, rejectUnauthorized : false, followRedirect: false), function(error, response, body) {
            if (error)
                cb(error);
            else
            {
                if(response.statusCode >= 300 & response.statusCode <= 399)
                {
                    count++;
                }
            }
        };
    }
    //for (let [x,y] links) { link <= 3. }
    for (let [x,y] links) {
        url = x;
        getcode(url, function(err, res, body) {
            if (err)
                callback(null, err);
            else
                callback(null, res);
        });
    }
}
```

cb(null,count);
}

};

} // have phridge spawn an instance of phantomjs

phridge.spawn()
.then(function (phantom) {
  return phantom.openPage(url);
})

// pull all the links off the webpage
.then(function (page) {
  // page.run(fn) runs fn inside PhantomJS
  return page.run(url, function (url) {
    // Here we're inside PhantomJS, so we can't reference variables in the scope
    // 'this' is an instance of PhantomJS' WebPage as returned by require('webpage').create()

    return this.evaluate(function () {
      var glinks = [document.URL];
      var links = document.links;
      for(var i=0;i< links.length;i++) {
        var url = links[i].href;
        if(url.indexOf("http") > -1 && url.indexOf('#') == -1)
          glinks[glinks.length] = url;
      }

      return glinks;
    });
  });
});

});

// phridge.disposeAll() exits cleanly all previously created child processes.

finally(phridge.disposeAll)

// we've got the links, now check them all
// (even if main page is 404, still helps to know if the error page has broken links)

.done(function (links) {
  async.map(links, getCode, function(err, results) {
    if(err)
      console.log(err);
    else
      {
        var res = true;
        if (count < range[0] || count > range[1])
          res = false;
        console.log("\Number of redirects found: "+count);
        console.log("\Number of redirects within ["+range[0]+","+range[1]+"]: "+res+"\n");
        callback(null,res);
      }
    }, function (err) {
      // Don't forget to handle errors
      // In this case we're just throwing it
      throw err;
    });
  });

// return check;
// check for certificate errors
// for all links { link.cert = true }

function noPageHasBadCert(url, callback) {
  console.log("\tChecking for bad certificates...");
  var count = 0;

  /* set up code to get server status */
  var getCode = function(url, cb) {
    request.get({url: url, rejectUnauthorized: false}, function(error, response, body) {
      // for testing purposes, pretent bad cert for these cases: site1, site2, site3
      if (url.indexOf('site1') > -1 || url.indexOf('site2') > -1 || url.indexOf('site3') > -1)
        error = "CERT_UNTRUSTED";
      if (error)
      {
        if (error.indexOf('CERT') > -1)
        {
          console.log("\t\terror\t\":\t" + url);
          cb(null, false);
        } else
          cb(error);
      } else
      {
        cb(null, true);
      }
    });
  }

  /* have phridge spawn an instance of phantomjs */
  phridge.spawn()
    .then(function(phantom) {
      // open the webpage
      return phantom.openPage(url);
    })
    .then(function(page) {
      // page.run(fn) runs fn inside PhantomJS
      return page.run(url, function() {
        // Here we're inside PhantomJS, so we can't reference variables in the scope
        // 'this' is an instance of PhantomJS' WebPage as returned by require("webpage").create()
        return this.evaluate(function () {
          var glinks = [document.URL];
          var links = document.links;
          for(var i=0; i < links.length; i++) {
            var url = '+' + links[i].href;
            if(url.indexOf('http') > -1)
              glinks[glinks.length] = '+' + url;
          }
          return glinks;
        });
      });
  });
}
// phridge.disposeAll() exits cleanly all previously created child processes.
// This should be called in any case to clean up everything.
}.finally(phridge.disposeAll)

// we've got the links, now check them all
// (even if main page is 404, still helps to know if the error page has broken links)
$.done(function (links) {
  $.async.map(links, getCode, function(err, results) {
    if(err)
      {
        console.log(err);
      }
    else
      {
        var res = true;
        for(var i = 0; i < results.length; i++){
          if (results[i]===false)
            {
              res = false;
              break;
            }
        }
        console.log("\nNo bad certificates discovered: "+res+\n");
        callback(null, res);
      }
  }, function (err) {
    // Don't forget to handle errors
    // In this case we're just throwing it
    throw err;
  });

// check for nav menu consistency
// for (all <page1, page2> { page1.nav == page2.nav }
function allNavMenuConsistent(url, callback){
  var useNav = true;
  var mlinks = [];
  var gnav = [];
  var mnavs = [];
  var nlinks = [];
  console.log("\nChecking for nav menu consistency...");
  request.get({url : url, rejectUnauthorized : false}, function(err, resp, body) {
    if (err)
      console.log(err);
    else {
      $ = cheerio.load(body);
      var nav = []
      nav = $('nav');
      // if no nav tags
      if (nav.length < 1)
      {
        useNav = false;
        nav = $('ul');
      }
  });
}
// grab ul tags that have navigation roles

lodash.every(nav, function(nav){
  if (nav.attr('role') !== undefined &&
      nav.attr('role') === 'menu' || nav.attr('role') === 'menubar' || nav.attr('role') === 'navigation'))
  gnava[gnava.length] = nav;
  return true;
});

for(var i = 0; i < gnava.length; i++)
{
  var inter = $(gnava[i]).html;
  $ = cheerio.load(inter);
  var list = $('a');
  lodash.forEach(list, function (link) {
    mlinks[mlinks.length] = $(link).attr('href')+'';
    return true;
  })
}

else
{
  var mnav = $(nav).html;
  $ = cheerio.load(mnav);
  mlinks = $('a');
}

});

var getLinks = function(navsi, cb) {
    mnavs = [];
    mlinks = [];
    lodash.every(navsi, function(nav){
      if (nav.attr('role') !== undefined &&
          nav.attr('role') === 'menu' || nav.attr('role') === 'menubar' || nav.attr('role') === 'navigation'))
        mnavs[mnavs.length] = nav;
        return true;
    });

    for(var i = 0; i < mnavs.length; i++)
    {
      var inter = $(mnavsi[i]).html;
      $ = cheerio.load(inter);
      var list = $('a');
      lodash.forEach(list, function (link) {
        mlinks[mlinks.length] = link;
        return true;
      })
    }

    cb(null, [mlinks, mnavs]);
}
var getNav = function(link, cb) {
    request.get({url: link, rejectUnauthorized: false}, function(err, resp, body) {
        if (err)
            cb("Error":err);
        else {
            $ = cheerio.load(body);
            var navsi = [];
            navsi = $('nav');
            // no nav tags, use <ul> tags
            if (useNav == false && navsi.length < 1)
                navsi = $('ul');
            var rets = getLinks(navsi, function(err, arr) {
                if(mlinks.length != nlinks.length)
                    cb(null, true);
                else {
                    $ = cheerio.load($(navsi).html);
                    for(var i = 0; i < mlinks.length ; i++)
                        if($(mlinks[i]).attr('href') != $(nlinks[i]).attr('href'))
                            cb(null, true);
                    cb(null, false);
                }
            })
        }
        else if(useNav == false)
            cb(null, true);
        else {
            var nnav = $(navsi).html;
            $ = cheerio.load(nnav);
            nlinks = $('a');
            if(mlinks.length != nlinks.length)
                cb(null, true);
            else {
                var allMatch = true;
                for(var i = 0; i < mlinks.length ; i++)
                    if($(mlinks[i]).attr('href') != $(nlinks[i]).attr('href'))
                        allMatch = false;
                cb(null, true);
            }
            if(allMatch)
                cb(null, false);
        }
    });
phridge.spawn()

.then(function (phantom) {
    return phantom.openPage(url);
})

// pull all the links off the webpage
.then(function (page) {
    return page.run(fn, function (url) {
        // 'this' is an instance of PhantomJS' WebPage as returned by require("webpage").create()
        return this.evaluateFunction() {
            var splitter = document.URL.split('/');
            var core = splitter[2];
            var glinks = [document.URL];
            var links = document.links;
            for(var i=0; i < links.length; i++) {
                var url = `'+links[i].href;
                if(url.indexOf("http") > -1 & url.indexOf('://') == -1 &
                (url.indexOf(core) == 7 || url.indexOf(core) == 8)
                & (url.charAt(url.length-4) != "."))
                glinks[glinks.length] = `'+url;
            return glinks;
        });
    });
})

// phridge.disposeAll() exits cleanly all previously created child processes.

// we've got the links, now check them all

.done(function (links) {
    var nLinks = [links[0]];
    for (var i = 1; i < links.length; i++) {
        var match = false;
        for (var j = 0; j < nLinks.length; j++) {
            if(links[i] == nLinks[j])
                match = true;
        }
        if(match == false)
            nLinks[nLinks.length] = links[i];
    }
    async.map(nLinks, getNav, function(err, results) {
        if(err)
            { console.log("Error:"+err);
            }
        else
            { var res = true;
                var cnt = 0;
                for(var i = 0; i < results.length; i++) {
                    if (results[i]==true)
                        { console.log("\n"+nLinks[i]);
                            cnt++;
                            res = false;
                        }
```javascript
987 };
988 console.log("\tSites checked with main page: "+nLinks.length-1));
989 console.log("\tSites inconsistent: "+cnt);
990 console.log("\tConsistent navigation menu: "+res="\n");
991 callback(null, res);
992 });
993 });
994 }, function (err) {
995 // Don't forget to handle errors
996 // In this case we're just throwing it
997 throw err;
998 });

// check for ARIA error for form input
// for (all form) { form.isARIA = true }

function allFormsAreARIA(url, callback){
  console.log("\tChecking for Form ARIA compliance...");
  // collect all input 'text' tags and all div tags with class = 'error'
  phantom.spawn()
    .then(function (phantom) {
      // open the webpage
      return phantom.openPage(url);
    })
    .then(function (page) {
      // page.run(fn) runs fn inside PhantomJS
      return page.run(url, function (url) {
        // Here we're inside PhantomJS, so we can't reference variables in the scope
        // 'this' is an instance of PhantomJS' WebPage as returned by require("webpage").create()
        return this.evaluate(function () {
          var gins = [];
          var ginsi = [];
          var ginsID = [];
          var gerrs = [];
          var berrs = [];
          var bins = [];
          var bad = [];
          var forms = document.getElementsByTagName('form');
          var ins = document.getElementsByTagName('input');
          var errs = document.getElementsByTagName('error');
          //console.log(ins);
          if (ins.length < 1 || typeof forms === 'undefined') // ins == undefined
          {
            bad[bad.length] = "\tNo form on page."
            return([-1,-1,-1,bad]);
          }
          // check if all error class tags have role set as alert
          for(var i=0;i < errs.length;i++) {
            var lerr = errs[i].getAttribute('role');
            if(lerr == 'alert')
              gerrs[gerrs.length] = errs[i].id;
            else
              berrs[berrs.length] = errs[i];
        }```
if ( gerrs.length < errs.length )
{
  bad[bad.length] = \"Error class tags missing \"role: \'alert\"\":\";\n  for( var i = 0; i < berrs.length; i++)
  {
    bad[bad.length] = \"\t\tid = \"+ berrs[i].id;
  }
  return([-2,-2,-2,bad]);
}

// get all inputs not having type='button' or 'submit'
for( var i=0; i< ins.length; i++) {
  var linT = ins[i].getAttribute(\"type\");
  if(linT != 'button' & linT != 'submit' & linT != 'hidden' & linT !=
    ginsi[ginsi.length] = ins[i];
}

// check if all input tags have aria-describedby attribute
for( var i = 0; i < ginsi.length; i++) {
  if(ginsi[i].getAttribute(\"aria-describedby\")
  {
    ginsi[ginsi.length] = ginsi[i].getAttribute(\"aria-describedby\");
    ginsID[ginsID.length] = ginsi[i].id;
  }
  else
    bins[bins.length] = ginsi[i];
}

if ( bins.length > 0 )
{
  bad[bad.length] = \"Input tags missing aria-describedby:\";
  for( var i = 0; i < bins.length; i++)
  {
    bad[bad.length] = \"\t\tid = \"+ bins[i].id;
  }
  return([-2,-2,-2,bad]);
}
else if( ginsi.length == 0)
{
  bad[bad.length] = \"No form on page.\";
  return([-1,-1,-1,bad]);
}
return [gissID, gins, gerrs, bad];
});
}
}

// phridge.disposeAll() exits cleanly all previously created child processes.
// This should be called in any case to clean up everything.
}.finally(phridge.disposeAll)

//we've got the tags now compare
.done(function (results) {
  var ins = results[0];
  var insa = results[1];
  var errs = results[2];
  var bads = results[3];
var rets = [];
var res = true;

if(results[0] == -2)
{
    res = false;
}
else if(results[0] == -1)
{
    res = -1;
}

if(errs.length > ins.length)
{
    console.log("More error class tags than inputs: "+errs.length+","+ins.length
    callback(null, false);
}

// check that aria-describedby matches error id
for(var i = 0; i < ins.length; i++)
{
    for(var j = 0; j < errs.length; j++)
    {
        if(ins[i] == errs[j])
        {
            rets[i] = true;
            break;
        }
    }
    if(rets.length == i)
    rets[i] = false;
}

// output results of test
for(var k = 0; k < rets.length;k++)
{
    if(rets[k] == false)
    {
        console.log("Input has no matching error class: "+ins[k]);
        res = false;
    }
}

if (res !== -1)
{
    for(var i = 0; i< bads.length;i++)
    {
        console.log(bads[i]);
        console.log("All form input ARIA compliant: "+res+"\n");
        callback(null, res);
    }
}
else
{
    console.log(bads[0]);
    console.log("Test inconclusive: nothing to test"+"\n");
    callback(null, res);
}
}, function (err) {
    throw err;
});
APPENDIX D

TEST SITES AND OUTPUT RESULTS

Site 1

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb" lang='en-us'>
<!--Form code taken from
http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3--!
<head>
<style type="text/css">
div.text, 
div.textinvalid, 
div.button
{
  margin: 0;
  padding: 0;
  padding-left: 20px;
  padding-bottom: .5em;
  display: block;
}
div.text label, 
div.textinvalid label {
  margin: 0;
  padding: 0;
  display: block;
  padding-top: .25em;
}
div.text input, 
div.textinvalid input {
  margin: 0;
  padding: 0;
  display: inline;
}
span.inst {
  font-size: 75%;
  color: blue;
  padding-left: .25em;
}
div.text input:active, 
div.text input:focus,
div.text input:hover {
    border-color: gray;
    background-color: #E0E0E0;
}

div.textinvalid input:active,
div.textinvalid input:focus,
div.textinvalid input:hover {
    border-color: gray;
    background-color: #FF8080;
}

form div.error {
    display: none;
    border: thin solid red;
    padding: .25em;
    color: red;
    font-size: 80%;
}

.offscreen {
    position: absolute;
    top: -30em;
    left: -300em;
}

</style>

<script type="text/javascript">

//
<!-[CDATA[//><!--

function validateForm() {
    var errorMessage = "Please complete the following fields:";
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;
    // check required fields
    // Check first name
    var node = document.getElementById("fname");
    if( node && node.value == "" ) {
        result = "\nYou must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }
    // Check last name
    node = document.getElementById("lname");
    if( node && node.value == "" ) {

result = "\nYou must enter your last name";
errorElements.push(result);
showError(node, result);
if( node_focus == null )
    node_focus = node;
}

// Check date
node = document.getElementById("date");
if( node ) {
    result = validateDate( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }  // end if
} // end if

// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }  // end if
} // end if

// Check Phone
node = document.getElementById("phone");
if( node && node.value == "" ) {
    result = "\nYou must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        }  // end if
    } // end if
} // end if
if( node_focus )
    node_focus.focus();
return false;
} else {
    // supress submit for test; for a real form, return true
    alert("If this were a real form, you would have just submitted it.");
    return false;
}

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation(event);
}

function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation(event);
}

function checkDate(event) {
    var node = getTarget(event);
    if( node ) {
        var result = validateDate(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}
function validateDate(node) {
    var errorMessage = "Date needs to be in date format, such as 1/31/2001."
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
            if (node.value.match(/D\d{1,2}$/) &&
                dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
                dateValue.setFullYear(dateValue.getFullYear() + 100);
            }
            // format as mm/dd/yyyy
            node.value = (dateValue.getMonth() + 1) + "/" +
                dateValue.getDate() + "/" + dateValue.getFullYear();
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function checkPhone(event ) {
    var node = getTarget( event);
    if( node ) {
        var result = validatePhone( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validatePhone( node ) {
    var errorMessage = "Phone needs to be a phone number with area code, such as (555) 123-4567."
    if (node.value != "") {
        // replace all non-digit characters, then match string starting with 2-9 (US phone numbers can't start with 0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/D/g, "]).match(/([2-9]\d{2})(\d{3})(\d{4})/) ) {
            // use sub-matches captured above to format as (###) ###-####
node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
    return "";
} else {
    return errorMessage;
}

return "You must enter a phone number, such as (555) 123-4567";
}

function checkSSN(event) {
    var node = getTarget( event );
    if( node ) {
        var result = validateSSN( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateSSN(node) {
    var errorMessage = "Social Security Number needs to be a nine-digit number, such as 123-45-6789.";
    if (node.value != "") {
        // replace all non-digit characters, then match 3 digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/((\d{3})(\d{2})(\d{4})/) ) {
            // use sub-matches captured above to format as ###-##-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById( id );
    if( node_error )
        node_error.innerHTML = "<span class='offscreen'>In valid: </span>" + message;
    if( message == "" ) {
        node.setAttribute('aria-invalid', 'false');
node_error.style.display = "none";
} else {
    node.setAttribute('aria-invalid', 'true');
    node_error.style.display = "inline";
}
}
function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
        return e.srcElement;
}
function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble)
        event.cancelBubble = true;
    if( event.returnValue)
        event.returnValue = false;
    return false;
}

//--><!]></script>
</head>
<body>
<div id="skip-link">
    <a href="#main-content" class="element-invisible element-focusable">Skip to main content</a>
</div>
<nav id="primaryNav">
    <h2>Navigation Menu</h2>
    <ul class="nav-list-primary">
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1'">Site 1</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17'>Site 17</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20'>Site 20</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22'>Site 22</a></li>
    </ul>
</nav>
Sample Form

Fields marked with an <img src="images/required.png" class="required" alt="Field is Required."/> are required.

<form id="SampleForm" action="#" method="post">

  <div class="text">
    <label for="fname">First Name</label>
    <input type="text" name="fname" id="fname" size="12" onblur="checkFirst(event)"
          aria-required="true"
          aria-invalid="false"
          aria-describedby="fname_error"/>
    <div class="error" id="fname_error" role="alert"></div>
  </div>

  <div class="text">
    <label for="lname">Last Name</label>
    <input type="text" name="lname" id="lname" size="20" onblur="checkLast(event)"
           aria-required="true"
           aria-invalid="false"
           aria-describedby="lname_error"/>
    <div class="error" id="lname_error" role="alert"></div>
  </div>

  <div class="text">
    <label for="date">Birth Date</label>
    <input type="text"

</form>
<form>
  <div class="error" id="date_error" role="alert"></div>
  <div class="text">
    <label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
    <input type="text" name="ssn" id="ssn" size="10" onblur="checkSSN(event)"
    aria-required="false" aria-invalid="false"
    aria-describedby="ssn_error" />
    <div class="error" id="ssn_error" role="alert"></div>
  </div>
  <div class="text">
    <label for="phone">Phone</label>
    <input type="text" name="phone" id="phone" size="14" onblur="checkPhone(event)"
    aria-required="true" aria-invalid="false"
    aria-describedby="phone_error" />
    <div class="error" id="phone_error" role="alert"></div>
  </div>
  <div class="button">
    <input type="button" value="Check Form"
    onclick=" validateForm()" />
  </div>
</form>
Results

Running ./tests/test1.js
Running all tests...
  Checking for image alt text
  Number of images = 4
  Number missing alt text = 0
  All images have alt text: true

  Checking for image alt text with conditions
  Number of images = 4
  Number missing alt text = 0
  Number failing test = 0
  All images meet alt text condition: true

  Checking for image alt text consistency
  Number of images = 4
  Number of images missing alt text = 0
  Number images inconsistent = 0
  All images are consistent: true

  Checking for image alt text with word limits
  Number of images = 4
  Number missing alt text = 0
  Number not within word limit = 0
  All images meet alt text word limit: true

  Checking for page language...
  Language found on https://csel.cs.colorado.edu/~erdu8260/site1: en-us

  Checking for some section languages...
  Language found for some sections on https://csel.cs.colorado.edu/~erdu8260/site1

  Checking for lang on all <p> tags...
  Number of <p>: 2
  Number of <p> missing lang: 0
  All <p> have lang attribute: true

  Checking for skip to main content link...
Skip to main link found: true

Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site1
No 404 pages discovered: true

Checking for [0,4] redirect pages...
Number of redirects found: 0
Number of redirects within [0,4]: true

Checking for bad certificates...
No bad certificates discovered: true

Checking for nav menu consistency...
Sites checked: 5
Sites inconsistent: 0
Consistent navigation menu: true

Checking for form ARIA compliance...
All form input ARIA compliant: true

Tests complete.

Site 2

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox"
cellspacing="0" style="">
<tr class="sites-header-primary-row" id="sites-chrome-userheader">
</tr>
<tr class="sites-header-secondary-row" id="sites-chrome-horizontal-nav">
```
<td colspan="2" id="sites-chrome-header-horizontal-nav-container" role="navigation">
</td>
</tr>
</table>
<table id="sites-chrome-main" class="sites-layout-hbox cellspacing="0" cellpadding="{scmCellpadding}" border="0">
<tr>
<td id="sites-chrome-sidebar-left" class="sites-layout-sidebar-left" style="display: none; width: 150px">
<div xmlns="http://www.w3.org/1999/xhtml" id="COMP_2bd" class="sites-embed role="navigation">
</div>
</td>
<td id="sites-canvas-wrapper">
</td>
</tr>
</table>
<table border="0" bordercolor="#888" cellspacing="0" style="color:rgb(56,118,29);border-collapse:collapse;border-color:rgb(136,136,136);border-width:0px">
  <tbody>
    <tr>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/mauifromsea.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/pacific.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/fishy.JPG" alt="image"/></div>
      </td>
    </tr>
    <tr>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/sharktank.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/mauifromsea.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/pacific.JPG" alt="image"/></div>
      </td>
    </tr>
    <tr>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/fishy.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/sharktank.JPG" alt="image"/></div>
      </td>
      <td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/mauifromsea.JPG" alt="image"/></div>
      </td>
    </tr>
  </tbody>
</table>
Results

Running ./tests/test2.js
Running all tests...
Checking for image alt text
Number of images = 7
Number missing alt text = 0
All images have alt text: true

Checking for image alt text with conditions
images/mauifromsea.JPG
image
images/pacific.JPG
image
images/fishy.JPG
image
images/sharktank.JPG
image
images/steelshark.JPG
Number of images = 7
Number missing alt text = 0
Number failing test = 6
All images meet alt text condition: false

Checking for image alt text consistency
image
images/rainbowmaui.JPG
images/sharktank.JPG
image
images/rainbowmaui.JPG
images/steelshark.JPG
image
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 15
All images are consistent: false

Checking for image alt text with word limits
images/mauifromsea.JPG
image
images/pacific.JPG
image
images/fishy.JPG
image
images/sharktank.JPG
image
images/steelshark.JPG
image
images/rainbowmaui.JPG
image
Number of images = 7
Number missing alt text = 0
Number not within word limit = 6
All images meet alt text word limit: false

Tests complete.

Site 3

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox"
cellspacing="0" style="">

Maui Pictures

All images created/photographed by author. Copyright 2014.

| ![Maui from Sea](images/mauifromsea.JPG) | ![Pacific](images/pacific.JPG) | ![Fishy](images/fishy.JPG) |
Results

Running ./tests/test3.js
Running all tests...
  Checking for image alt text
  Number of images = 7
  Number missing alt text = 0
  All images have alt text: true

  Checking for image alt text with conditions
Number of images = 7
Number missing alt text = 0
Number failing test = 0
All images meet alt text condition: true

Checking for image alt text consistency
images/mauifromsea.JPG
images/pacific.JPG
This is a picture of something.
images/mauifromsea.JPG
images/fishy.JPG
This is a picture of something.
images/mauifromsea.JPG
images/sharktank.JPG
This is a picture of something.
images/mauifromsea.JPG
images/steelshark.JPG
This is a picture of something.
images/mauifromsea.JPG
images/rainbowmaui.JPG
This is a picture of something.
images/pacific.JPG
images/mauifromsea.JPG
This is a picture of something.
images/pacific.JPG
images/fishy.JPG
This is a picture of something.
images/pacific.JPG
images/sharktank.JPG
This is a picture of something.
images/pacific.JPG
images/sharktank.JPG
This is a picture of something.
images/pacific.JPG
images/rainbowmaui.JPG
This is a picture of something.
images/fishy.JPG
images/mauifromsea.JPG
This is a picture of something.
images/fishy.JPG
images/mauifromsea.JPG
This is a picture of something.
images/fishy.JPG
images/pacific.JPG
This is a picture of something.
images/fishy.JPG
images/sharktank.JPG
This is a picture of something.
images/fishy.JPG
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
This is a picture of something.
images/rainbowmaui.JPG
images/steelshark.JPG
This is a picture of something.
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 15
All images are consistent: false

Checking for image alt text with word limits
images/mauifromsea.JPG
This is a picture of something.
images/pacific.JPG
This is a picture of something.
images/fishy.JPG
This is a picture of something.
images/sharktank.JPG
This is a picture of something.
images/steelshark.JPG
This is a picture of something.
images/steellowshark.JPG
This is a picture of something.
Number of images = 7
Number missing alt text = 0
Number not within word limit = 6
All images meet alt text word limit: false

Tests complete.

Site 4

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
  <head>
    <title>site2</title>
  </head>
  <body>
    <table id="sites-chrome-header" class="sites-layout-hbox"
      cellspacing="0" style="">
      <tr class="sites-header-primary-row" id="sites-chrome-userheader">
        </tr>
    </table>
  </body>
</html>
```
<table>
<thead>
<tr>
<th>Class</th>
<th>Tag</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sites-header-secondary-row</td>
<td>td</td>
<td>colspan=&quot;2&quot;</td>
</tr>
<tr>
<td>sites-chrome-horizontal-nav</td>
<td>tr</td>
<td>id=&quot;sites-chrome-header-horizontal-nav-container&quot; role=&quot;navigation&quot;</td>
</tr>
<tr>
<td>sites-chrome-main</td>
<td>table</td>
<td>class=&quot;sites-layout-hbox&quot; cellspacing=&quot;0&quot; cellpadding=&quot;{scmCellpadding}&quot; border=&quot;0&quot;</td>
</tr>
<tr>
<td>sites-chrome-sidebar-left</td>
<td>td</td>
<td>id=&quot;sites-chrome-sidebar-left&quot; class=&quot;sites-layout-sidebar-left&quot; style=&quot;display: none; width: 150px&quot;</td>
</tr>
<tr>
<td>sites-embed</td>
<td>div</td>
<td>xmlns=&quot;<a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a>&quot; id=&quot;COMP_2bd&quot; class=&quot;sites-embed&quot; role=&quot;navigation&quot;</td>
</tr>
<tr>
<td>sites-sidebar-nav</td>
<td>ul</td>
<td>jotId=&quot;navList&quot; class=&quot;has-expander&quot; li class=&quot;topLevel nav-first parent &quot; wuid=&quot;gx:1f8bd6bae67c3bfbb&quot;</td>
</tr>
<tr>
<td>sites-navigation-link topLevel</td>
<td>div</td>
<td>id=&quot;/home&quot; jotId=&quot;wuid:gx:1f8bd6bae67c3bfbb&quot; class=&quot;sites-navigation-link topLevel&quot;</td>
</tr>
<tr>
<td>sites-navigation-link topLevel</td>
<td>div</td>
<td>id=&quot;/gallery&quot; jotId=&quot;wuid:gx:4f3d62c52ca985a1&quot; class=&quot;sites-navigation-link&quot;</td>
</tr>
<tr>
<td>sites-navigation-link</td>
<td>div</td>
<td>id=&quot;/links&quot; jotId=&quot;wuid:gx:299d2a6b1f87e0e1&quot; class=&quot;sites-navigation-link&quot;</td>
</tr>
<tr>
<td>sites-navigation-link</td>
<td>div</td>
<td>id=&quot;/madbot&quot; jotId=&quot;wuid:gx:399a0598e9c220e0&quot; class=&quot;sites-navigation-link&quot;</td>
</tr>
<tr>
<td>sites-navigation-link</td>
<td>div</td>
<td>id=&quot;/quotes&quot; jotId=&quot;wuid:gx:6145cb6acf2c8a06&quot; class=&quot;sites-navigation-link&quot;</td>
</tr>
<tr>
<td>sites-navigation-link</td>
<td>div</td>
<td>id=&quot;/writings&quot; jotId=&quot;wuid:gx:4da851d18a38cc16&quot; class=&quot;sites-navigation-link&quot;</td>
</tr>
<tr>
<td>sitemap/hierarchy</td>
<td>div</td>
<td>href=&quot;/system/app/pages/sitemap/hierarchy&quot;</td>
</tr>
<tr>
<td>Sitemap</td>
<td>div</td>
<td>id=&quot;/&quot; ilk class=&quot;topLevel&quot;</td>
</tr>
<tr>
<td>Maui Pictures</td>
<td>div</td>
<td>id=&quot;/home&quot; jotId=&quot;wuid:gx:1f8bd6bae67c3bfbb&quot; class=&quot;sites-embed&quot; role=&quot;navigation&quot;</td>
</tr>
<tr>
<td>Maui Pictures</td>
<td>A</td>
<td>href=&quot;/home&quot; dir=&quot;ltr&quot;</td>
</tr>
</tbody>
</table>

**Note:** The HTML code provided represents the structure of the webpage, including navigation links and page layout. It is formatted as a table with rows and columns, and includes various HTML elements such as `div`, `tr`, `td`, and `a`. The code also includes attributes like `id`, `class`, and `role` to define the structure and style of the webpage. The `xmlns` attribute is used to define the XML namespace for the W3C XLink and XPointer modules. The `href` attribute is used to define the URL for navigating to different sections of the webpage. The `dir` attribute is used to specify the direction of text flow, which is set to "ltr" (left to right) to ensure proper rendering of text in English.
<h3 xmlns="http://www.w3.org/1999/xhtml" id="sites-page-title-header" style="display: none;" align="left">
    <span id="sites-page-title" dir="ltr" tabindex="-1" style="outline: none">maui pictures</span></h3>

<table xmlns="http://www.w3.org/1999/xhtml" cellspacing="0" class="sites-layout-name-one-column sites-layout-hbox">
    <tbody><tr><td class="sites-layout-tile sites-tile-name-content-1">
        <div dir="ltr">
            <div style="text-align:center;color:rgb(51,51,51);line-height:48px">M</div>
            <span style="color:rgb(56,118,29)"><font size="5">MAUI: Ka'anapali</font></span>
        </div>
        <p style="text-align:center"><span style="font-size:1em;line-height:1.5">All images created/photographed by author. Copyright 2014.</span></p>
        </font>
    </td>
    </tr>
    <tr><td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/mauifromsea.JPG" alt="maui seen from the ocean"/></div>
    </td></tr>
    <tr><td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/pacific.JPG" alt="the pacific ocean"/></div>
    </td></tr>
    <tr><td style="text-align:center;width:60px">
        <div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/fishy.JPG" alt="some sort of fish at the aquarium"/></div>
    </td></tr>
    </tbody></table>
Results

Running ./tests/test4.js
Running all tests...
  Checking for image alt text
  Number of images = 7
  Number missing alt text = 0
  All images have alt text: true

  Checking for image alt text with conditions
    images/mauifromsea.JPG
    mauli seen from the ocean
    images/pacific.JPG
    the pacific ocean
    images/fishy.JPG
    some sort of fish at the aquarium
    images/sharktank.JPG
    La'haina aquarium shark tank
Number of images = 7
Number missing alt text = 0
Number failing test = 6
All images meet alt text condition: false

Checking for image alt text consistency
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 0
All images are consistent: true

Checking for image alt text with word limits
Number of images = 7
Number missing alt text = 0
Number not within word limit = 3
All images meet alt text word limit: false

Tests complete.
Site 5

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox" cellspacing="0" style="">
<tr class="sites-header-primary-row" id="sites-chrome-userheader">
</tr>
<tr class="sites-header-secondary-row" id="sites-chrome-horizontal-nav">
```
<td id="sites-chrome-header-horizontal-nav-container" role="navigation">
</td>
</tr>
</table>
<table id="sites-chrome-main" class="sites-layout-hbox"
cellspacing="0" cellpadding="{scmCellpadding}" border="0">
<tr>
<td id="sites-chrome-sidebar-left" class="sites-layout-sidebar-left" style="display: none; width: 150px">
<div xmlns="http://www.w3.org/1999/xhtml" id="COMP_2bd"
class="sites-embed" role="navigation">
<div class="sites-embed-content sites-sidebar-nav">
<ul role="navigation" jotId="navList" class="has-expander">
<li class="topLevel nav-first parent" wuid="gx:1f8bd6bae67c3bfb">
<a href="/home" jotId="wuid:gx:1f8bd6bae67c3bfb" class="sites-navigation-link topLevel">Home</a>
</li><li class="">
<a href="/home/gallery" jotId="wuid:gx:4f3d62c52ca985a1" class="sites-navigation-link">Gallery</a>
</li><li class="">
<a href="/home/links" jotId="wuid:gx:299d2a6b1f87e0e1" class="sites-navigation-link">Links</a>
</li><li class="">
<a href="/home/madbot" jotId="wuid:gx:399a0598e9c220e0" class="sites-navigation-link">madbot</a>
</li><li class="">
<a href="/home/quotes" jotId="wuid:gx:6145cb6acf2c8a06" class="sites-navigation-link">Quotes</a>
</li><li class="">
<a href="/home/writings" jotId="wuid:gx:4da851d18a38cc16" class="sites-navigation-link">Writings</a>
</li><li class="">
<a href="/system/app/pages/sitemap/hierarchy" jotId="wuid:gx:41406bcec4ac9cbf" class="sites-navigation-link">Sitemap</a>
</li></ul></div>
</div>
</td>
<td id="sites-canvas-wrapper">
</div>
</div>
</div>
</td>
</tr>
</table>

The page contains a header and a main content area with navigation links and a list of sections. The header includes links to various sections of the website, such as Gallery, Links, madbot, Quotes, and Writings. The main content area seems to be empty or not fully rendered. The navigation links are displayed vertically, and the content is structured with HTML tags and attributes such as `id`, `class`, and `role`.
<span id="sites-page-title" dir="ltr" tabindex="-1" style="outline: none">maui pictures</span>
<h3>
<table xmlns="http://www.w3.org/1999/xhtml" cellspacing="0" class="sites-layout-name-one-column sites-layout-hbox">
<tbody><tr><td class="sites-layout-tile sites-tile-name-content-1">
<div dir="ltr">
<div style="text-align:center;color:rgb(51,51,51);font-size:xx-large;line-height:48px">M</div><div style="color:rgb(56,118,29);font-size:5">UAU: Ka'anapali</div>
<p style="text-align:center"><span style="font-size:1em;line-height:1.5">All images created/photographed by author. Copyright 2014.</span></p>
</div>
</td></tr>
<tr>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" width="350" src="images/mauifromsea.JPG" alt="Maui from sea."/></div>
</td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" width="350" src="images/pacific.JPG" alt="Pacific Ocean"/></div>
</td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" width="350" src="images/fishy.JPG" alt="just keep swimming"/></div>
</td>
</tr>
<tr>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" width="350" src="images/sharktank.JPG" alt="A shark."/></div>
</td>
</tr>
</tbody></table>
</h3>
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/steelshark.JPG" alt="A shark."/></div>
</td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/rainbowmaui.JPG" alt="Rainbow over Maui"/></div>
</td>
</tr>
</tbody>
</table>
<center style="color:rgb(51,51,51)"><br /></center><center style="color:rgb(51,51,51)"><div style="display:block;text-align:center;margin-right:auto;margin-left:auto"><img alt="blanky blank blank" border="0" width="175" height="100" src="images/lightbox.JPG" /></div><br /></center>
</center>
</font></div></td></tr></tbody></table>

Results

Running ./tests/test5.js
Running all tests...

Checking for image alt text
Number of images = 7
Number missing alt text = 0
All images have alt text: true

Checking for image alt text with conditions
images/pacific.JPG
Pacific Ocean
images/fishy.JPG
just keep swimming
images/rainbowmaui.JPG
Rainbow over Maui
images/lightbox.JPG
blanky blank blank
Number of images = 7
Number missing alt text = 0
Number failing test = 4
All images meet alt text condition: false

Checking for image alt text consistency
images/sharktank.JPG
images/steelshark.JPG
  A shark.
images/steelshark.JPG
images/sharktank.JPG
  A shark.
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 1
All images are consistent: false

Checking for image alt text with word limits
Number of images = 7
Number missing alt text = 0
Number not within word limit = 0
All images meet alt text word limit: true

Tests complete.

Site 6

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox"
cellspacing="0" style="">
<tr class="sites-header-primary-row" id="sites-chrome-userheader">
</tr>
<tr class="sites-header-secondary-row" id="sites-chrome-horizontal-nav">
</tr>
All images created/photographed by author. Copyright 2014.

Maui as seen from a boat on the coast.

View of clouds over the Pacific Ocean.

An ocean fish.
Results

Running ./tests/test6.js
Running all tests...
  Checking for image alt text
  Number of images = 7
  Number missing alt text = 0
  All images have alt text: true

  Checking for image alt text with conditions
  Number of images = 7
  Number missing alt text = 0
  Number failing test = 0
  All images meet alt text condition: true
Checking for image alt text consistency
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 0
All images are consistent: true

Checking for image alt text with word limits
images/mauifromsea.JPG
Maui as seen from a boat on the coast.
images/pacific.JPG
View of clouds over the Pacific Ocean.
Number of images = 7
Number missing alt text = 0
Number not within word limit = 2
All images meet alt text word limit: false

Tests complete.

Site 7

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox"
cellspacing="0" style="">
<tr class="sites-header-primary-row" id="sites-chrome-
userheader">
</tr>
<tr class="sites-header-secondary-row" id="sites-chrome-
horizontal-nav">
<td colspan="2" id="sites-chrome-header-horizontal-nav-
container" role="navigation">
</td>
</tr>
</table>
<table id="sites-chrome-main" class="sites-layout-hbox"
cellspacing="0" cellpadding="{scmCellpadding}" border="0">
<tr>
<td id="sites-chrome-sidebar-left" class="sites-layout-
sidebar-left" style="display: none; width: 150px">
AUI: Ka'anapali

All images created/photographed by author. Copyright 2014.

<center>
<table border="0" bgcolor="#888" cellspacing="0" width="0px">
<tbody>
<tr>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/mauifromsea.JPG" alt="Maui from sea."/></div></td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/pacific.JPG" alt="The Pacific Ocean."/></div></td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/fishy.JPG" alt="A big fish."></div></td>
</tr>
<tr>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/sharktank.JPG" alt="The shark tank."></div></td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/sharktank.JPG" alt="A steel shark."></div></td>
<td style="text-align:center;width:60px">
<div style="margin-right:auto;margin-left:auto"><img border="0" height="200" width="350" src="images/rainbowmaui.JPG" alt="A rainbox over Maui."></div></td>
</tr>
</tbody></table>
</center>
Results

Running .tests/test7.js

Running all tests...

Checking for image alt text
Number of images = 7
Number missing alt text = 0
All images have alt text: true

Checking for image alt text with conditions
Number of images = 7
Number missing alt text = 0
Number failing test = 0
All images meet alt text condition: true

Checking for image alt text consistency
images/sharktank.JPG
The shark tank.
A steel shark.
images/sharktank.JPG
A steel shark.
The shark tank.
Number of images = 7
Number of images missing alt text = 0
Number images inconsistent = 1
All images are consistent: false

Checking for image alt text with word limits
Number of images = 7
Number missing alt text = 0
Tests complete.

**Site 8**

**Code**

```xml
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" 
'http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd'>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us">
<head>
<title>site2</title>
</head>
<body>
<table id="sites-chrome-header" class="sites-layout-hbox"
cellspacing="0" style="">
<tr class="sites-header-primary-row" id="sites-chrome-userheader">
</tr>
<tr class="sites-header-secondary-row" id="sites-chrome-horizontal-nav">
<td colspan="2" id="sites-chrome-header-horizontal-nav-container" role="navigation">
</td>
</tr>
</table>
<table id="sites-chrome-main" class="sites-layout-hbox"
cellspacing="0" cellspacing="{scmCellpadding}" border="0">
<tr>
<td id="sites-chrome-sidebar-left" class="sites-layout-sidebar-left" style="display: none; width: 150px">
<div xmlns="http://www.w3.org/1999/xhtml" id="COMP_2bd"
class="sites-embed" role="navigation"><div class="sites-embed-content sites-sidebar-nav"><ul role="navigation" jotId="navList" class="has-expander"><li class="topLevel nav-first parent" wuid="gx:1f8bd6bae67c3bfb"><a href="/home" jotId="wuid:gx:1f8bd6bae67c3bfb" class="sites-navigation-link topLevel">Home</a></li><li class="topLevel nav-first parent" wuid="gx:1f8bd6bae67c3bfb"><a href="/home/gallery" jotId="wuid:gx:4f3d62c52ca985a1" class="sites-navigation-link">Gallery</a></li></ul></div></div>
</td>
</tr>
</table>
</body>
</html>
```
<h3 xmlns="http://www.w3.org/1999/xhtml" id="sites-page-title-header" style="display: none;" align="left">
<span id="sites-page-title" dir="ltr" tabindex="-1" style="outline: none">maui pictures</span>
</h3>
<table xmlns="http://www.w3.org/1999/xhtml" cellspacing="0" class="sites-layout-name-one-column sites-layout-hbox"><tbody><tr><td class="sites-layout-tile sites-tile-name-content-1"><div dir="ltr"><div style="text-align:center;color:rgb(51,51,51);line-height:48px"><span style="color:rgb(56,118,29);font-size:xx-large;line-height:48px">M</span><span style="color:rgb(56,118,29)"><font size="5">AUI: Ka'anapali</font></span></div></div><p style="text-align:center"><span style="font-size:1em;line-height:1.5">All images created/photographed by author. Copyright 2014.</span><br></p></td></tr></tbody></table>
<table>
<thead>
<tr>
<th>Image Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maui from sea</td>
</tr>
<tr>
<td>The Pacific Ocean</td>
</tr>
<tr>
<td>Some fish</td>
</tr>
<tr>
<td>Big shark tank</td>
</tr>
<tr>
<td>Metal shark sculpture</td>
</tr>
<tr>
<td>A pretty rainbow</td>
</tr>
</tbody>
</table>

(blank image)

(blank image)

(blank image)
Results

Running ./tests/test8.js
Running all tests...
   Checking for image alt text
   Number of images = 7
   Number missing alt text = 0
   All images have alt text: true

   Checking for image alt text with conditions
      images/mauifromsea.JPG
         Maui from sea
      images/fishy.JPG
         some fish
      images/steelshark.JPG
         metal shark sculpture
      images/rainbowmaui.JPG
         a pretty rainbow
      images/lightbox.JPG
         blanky blank blank
   Number of images = 7
   Number missing alt text = 0
   Number failing test = 5
   All images meet alt text condition: false

   Checking for image alt text consistency
   Number of images = 7
   Number of images missing alt text = 0
   Number images inconsistent = 0
   All images are consistent: true

   Checking for image alt text with word limits
   Number of images = 7
   Number missing alt text = 0
   Number not within word limit = 0
   All images meet alt text word limit: true

Tests complete.

Site 9

Code
This is how we say hello in many languages:

- Hello
- Hola
- Bonjour
- Ciao
- Hallo
- Dia duit
- Kumusta
- Salve
- Hei
- Kon'nichiwa

Results

Running ./tests/test9.js
Running all tests...
Checking for page language...
Language found on https://csel.cs.colorado.edu/~erdu8260/site9: en-us

Checking for some section languages...
Language not found for any sections on https://csel.cs.colorado.edu/~erdu8260/site9

Checking for lang on all <p> tags...
Number of <p>: 11
Number of <p> missing lang: 11
All <p> have lang attribute: false
Tests complete.

Site 10

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang='en-us'>
  <p> This is how we say hello in many languages: </p>
  <p>Hello</p>
  <p lang='sp'>Hola</p>
  <p lang='fr'>Bonjour</p>
  <p lang='it'>Ciao</p>
</HTML>
```
<p lang='de'>Hallo</p>
<p lang='ga'>Dia duit</p>
<p>Kumusta</p>
<p lang='la'>Salve</p>
<p lang='fi'>Hei</p>
<p lang='ja'>Kon'nichiwa</p>
</HTML>

Results

Running ./tests/test10.js
Running all tests...
  Checking for page language...
  Language found on https://csel.cs.colorado.edu/~erdu8260/site10: en-us

  Checking for some section languages...
  Language found for some sections on https://csel.cs.colorado.edu/~erdu8260/site10

  Checking for lang on all <p> tags...
  Number of <p>: 11
  Number of <p> missing lang: 3
  All <p> have lang attribute: false

Tests complete.

Site 11

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<body>
<h1>A page of lovely links</h1>
Results

Running ./tests/test11.js
Running all tests...
  Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site11
  No 404 pages discovered: true

  Checking for [0,4] redirect pages...
  Number of redirects found: 5
  Number of redirects within [0,4]: false

  Checking for bad certificates...
  [CERT_UNTRUSTED]:https://csel.cs.colorado.edu/~erdu8260/site11
  No bad certificates discovered: false

Tests complete.
Site 12

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
  <body>
    <h1>A page of lovely links</h1>
    <p><a href="http://www.google.com/imgres?imgurl=http://www.colorado.edu/sites/default/styles/medium-feature-image/public/slideshow-photos/RalphieRunning.jpg%253Fitok%253DIhxRc1J-4&imgrefurl=http://www.colorado.edu/about&amp;h=360&amp;w=680&amp;tbnid=9m2nXq6hLzzRrM:&zoom=1&amp;docid=LWarP3NKOUrNWM&amp;ei=agMGVczxHurIIsQTQ0YCAAg&amp;tbm=isch&amp;ved=0CB0QMygBMAE" >Ralphie</a></p>
    <p><a href="http://cs.colorado.edu/" >CU Boulder CS Homepage</a></p>
    <p><a href='http://google.com/' >Google Search Page</a></p>
    <p><a href="http://www.sandia.gov/" >SNL Homepage</a></p>
    <p><a href="http://cs.nmt.edu" >New Mexico Tech CS Homepage</a></p>
  </body>
</html>
Results

Running ./tests/test12.js
Running all tests...
Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site12
404: http://www-ia.hiof.no/~linettev/html4U/form.htm
No 404 pages discovered: false

Checking for [0,4] redirect pages...
Number of redirects found: 4
Number of redirects within [0,4]: true

Checking for bad certificates...
[CERT_UNTRUSTED]:https://csel.cs.colorado.edu/~erdu8260/site12
No bad certificates discovered: false

Tests complete.

Site 13

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<body>
<h1>A page of lovely links</h1>
9m2nXq6hLzzRrM: &zoom=1&docid=LWarP3NKOURNWM&ei=agMGVczxHurISQTQ0YCAAg&tbm=isch&ved=0CB0QMygBMAE" >Ralphie</a></p>
<p><a href="http://cs.colorado.edu/" >CU Boulder CS Homepage</a></p>
<p><a href='http://google.com/' >Google Search Page</a></p>
<p><a href="http://translate.google.com">Google Translate</a></p>
<p><a href="http://www.sandia.gov/" >SNL Homepage</a></p>
<p><a href="http://cs.nmt.edu" >New Mexico Tech CS Homepage</a></p>
<p><a href="http://99waysnotto.tumblr.com/welcome" >99 Ways Not To...</a></p>
</body>
</HTML>

Results

Running ./tests/test13.js
Running all tests...
  Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site13
    404: http://www-ia.hiof.no/~linettev/html4U/form.htm
    No 404 pages discovered: false
  Checking for [0,4] redirect pages...
    Number of redirects found: 5
    Number of redirects within [0,4]: false
  Checking for bad certificates...
    No bad certificates discovered: true
Tests complete.

**Site 14**

**Code**

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<body>
<h1>A page of lovely links</h1>
<p><a href="http://www.google.com/imgres?imgurl=http://www.colorado.edu/sites/default/files/styles/medium-feature-image/public/slideshow-photos/RalphieRunning.jpg%253Fitok%253DIhxRc1j- &imgrefurl=http://www.colorado.edu/about&h=360&w=680&tbnid=9m2nXq6hLzzRrM:&zoom=1&docid=LWarP3NKOURNWM&ei=agMGVczxHurI sQTQ0YCAA&tbm=isch&ved=0CB0QM2gBMAY" >Ralphie</a></p>
<p><a href="http://cs.colorado.edu/" >CU Boulder CS Homepage</a></p>
<p><a href='http://google.com/'>Google Search Page</a></p>
<p><a href="http://www.sandia.gov/" >SNL Homepage</a></p>
```
Results

Running ./tests/test14.js
Running all tests...
Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site14
  No 404 pages discovered: true

Checking for [0,4] redirect pages...
  Number of redirects found: 4
  Number of redirects within [0,4]: true

Checking for bad certificates...
  [CERT_UNTRUSTED]:https://csel.cs.colorado.edu/~erdu8260/site14
  No bad certificates discovered: false

Tests complete.

Site 15

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
  <body>
    <h1>A page of lovely links</h1>
  </body>
</html>
Results

Running ./tests/test15.js
Running all tests...
  Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site15
  404: http://www-ia.hiof.no/~linettev/html4U/form.htm
  No 404 pages discovered: false
  Checking for [0,4] redirect pages...
  Number of redirects found: 4
  Number of redirects within [0,4]: true
  Checking for bad certificates...
  No bad certificates discovered: true

Tests complete.
<!DOCTYPE HTML PUBLIC "//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
  <body>
    <h1>A page of lovely links</h1>
    <p><a href="http://www.google.com/imgres?imgurl=http://www.colorado.edu/sites/default/styles/medium-feature-image/public/slideshow-photos/RalphieRunning.jpg%253Fitok%253D1hxRc1J- &imgrefurl=http://www.colorado.edu/about/h=360&w=680&tnid=9m2nXg6hLzzRrM:&zoom=1&docid=LWarP3NKOURWfM&ei=agMGVczxHurISQT0YCAAg&tbm=isch&ved=0CB0QMygBMAE" >Ralphie</a></p>
    <p><a href="http://cs.colorado.edu/" >CU Boulder CS Homepage</a></p>
    <p><a href='http://google.com/' >Google Search Page</a></p>
    <p><a href="http://translate.google.com" >Google Translate</a></p>
    <p><a href="http://www.sandia.gov/" >SNL Homepage</a></p>
  </body>
</HTML>
<p><a href="http://cs.nmt.edu" >New Mexico Tech CS Homepage</a></p>
<p><a href="http://99waysnotto.tumblr.com/welcome" >99 Ways Not To...</a></p>
</body>
</html>

Results

Running ./tests/test16.js
Running all tests...
  Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site16
  No 404 pages discovered: true

  Checking for [0,4] redirect pages...
  Number of redirects found: 5
  Number of redirects within [0,4]: false

  Checking for bad certificates...
  No bad certificates discovered: true

Tests complete.

Site 17

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<head>
  <!--Form code taken from http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3-->!
  <style type="text/css">
  div.text,
  div.textinvalid,
  div.button
  {
    margin: 0;
    padding: 0;
    padding-left: 20px;
    padding-bottom: .5em;
    display: block;
  }
</style>
</head>
div.text label,
div.textinvalid label {
  margin: 0;
  padding: 0;
  display: block;
  padding-top: .25em;
}
div.text input,
div.textinvalid input {
  margin: 0;
  padding: 0;
  display: inline;
}
span.inst {
  font-size: 75%;
  color: blue;
  padding-left: .25em;
}
div.text input:active,
div.text input:focus,
div.text input:hover {
  border-color: gray;
  background-color: #E0E0E0;
}
div.textinvalid input:active,
div.textinvalid input:focus,
div.textinvalid input:hover {
  border-color: gray;
  background-color: #FF8080;
}
form div.error {
  display: none;
  border: thin solid red;
  padding: .25em;
  color: red;
  font-size: 80%;
}
.offscreen {
  position: absolute;
  top: -30em;
  left: -300em;
}

</style>
<script type="text/javascript"/>
function validateForm() {
    var errorMessage = "Please complete the following fields:";
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;
    // check required fields
    // Check first name
    node = document.getElementById("fname");
    if (node && node.value =="") {
        result = "You must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }
    // Check last name
    node = document.getElementById("lname");
    if (node && node.value =="") {
        result = "You must enter your last name";
        errorElements.push(result);
        showError(node, result);
        if (node_focus == null)
            node_focus = node;
    }
    // Check date
    node = document.getElementById("date");
    if (node) {
        result = validateDate(node);
        if (result !="") {
            errorElements.push(result);
            showError(node, result);
            if (node_focus == null)
                node_focus = node;
        } // end if
    } // end if
    // Check SSN
    node = document.getElementById("ssn");
    if (node) {
        result = validateSSN(node);
        if (result !="") {
            errorElements.push(result);
            showError(node, result);
            if (node_focus == null)
                node_focus = node;
        } // end if
    } // end if
// Check Phone
node = document.getElementById("phone");
if( node && node.value == "") {  
    result = "\nYou must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        }  // end if
    }  // end if
}  // end if else

if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++ ) {
        errorMessage += "\n" + errorElements[i];
    }  // end for
    alert(errorMessage);
    if( node_focus )
        node_focus.focus();
    return false;
} else {
    // suppress submit for test; for a real form, return true
    alert("If this were a real form, you would have just submitted it.");
    return false;
}

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    }  // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkDate(event) {
    var node = getTarget(event);
    if( node ) {
        var result = validateDate( node);
        showError(node, result);
    }  // endif
    return stopPropagation( event);
}

function validateDate(node) {
    var errorMessage = "\nDate needs to be in date format, such as 1/31/2001.";
    if (node.value != "") {  // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
            if (node.value.match(/\d{1,2}$/) &&
                dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
                dateValue.setFullYear(dateValue.getFullYear() + 100);
            }
            // format as mm/dd/yyyy
            node.value = (dateValue.getMonth() + 1) + "/" +
            dateValue.getDate() + "/" + dateValue.getFullYear();
            return ";
        }
        else {
            return errorMessage;
        }
    }
    return "";
}
function checkPhone(event) {
    var node = getTarget(event);
    if (node) {
        var result = validatePhone(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validatePhone(node) {
    var errorMessage = "Phone needs to be a phone number with area code, such as (555) 123-4567."
    if (node.value != "") {
        // replace all non-digit characters, then match string starting with 2-9 (US phone numbers can't start with 0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/D/g, "").match(/\([2-9]\d\d\\d)\d\d\d\d/)) {
            // use sub-matches captured above to format as (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "+ RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "You must enter a phone number, such as (555) 123-4567";
}

function checkSSN(event) {
    var node = getTarget(event);
    if (node) {
        var result = validateSSN(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validateSSN(node) {
    var errorMessage = "Social Security Number needs to be a nine-digit number, such as 123-45-6789."
    if (node.value != "") {
        // replace all non-digit characters, then match 3 digits, 2 digits, 4 digits
        if (node.value.replace(/D/g, "").match(/\d\d\d\d\d\d\d\d/)) {
            return "";
        } else {
            return errorMessage;
        }
    }
    return "You must enter a Social Security Number, such as 123-45-6789.";
}
// use sub-matches captured above to format as

```javascript
node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
return "";
```

```javascript
else {
    return errorMessage;
}
```

```javascript
return "";
```

```javascript
function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById(id);
    if(node_error)
        node_error.innerHTML = "<span class='offscreen'>Invalid: </span>" + message;
    if(message == "") {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
    }
}
```

```javascript
function getTarget(event) {
    var e = event || window.event;
    if(e.target)
        return e.target;
    else
        return e.srcElement;
}
```

```javascript
function stopPropagation(event) {
    if(event.stopPropagation)
        event.stopPropagation();
    if(event.preventDefault)
        event.preventDefault();
    if(event.cancelBubble)
        event.cancelBubble = true;
    if(event.returnValue)
        event.returnValue = false;
    return false;
}
```

```javascript
//--&gt;&lt;!&gt;

&lt;/script&gt;
&lt;/head&gt;```
<body>
  <nav id="primaryNav">
    <h2>Navigation Menu</h2>
    <ul class="nav-list-primary">
      <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1' >Site 1</a></li>
      <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17' >Site 17</a></li>
      <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20' >Site 20</a></li>
      <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22' >Site 22</a></li>
    </ul>
  </nav>
  <div class="example">
    <div>
      <h2>Sample Form</h2>
      <p>Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.</p>
      <form id="SampleForm" action="#" method="post">
        <div class="text">
          <label for="fname">
            First Name
            <img src="images/required.png" alt="" />
          </label>
          <input type="text" name="fname" id="fname" size="12" onblur="checkFirst(event)"
            aria-required="true" aria-invalid="false" />
          <div class="error" id="fname_error" role="alert">"</div>
        </div>
        <div class="text">
          <label for="lname">
            Last Name
            <img src="images/required.png" alt="" />
          </label>
          <input type="text" name="lname" id="lname" size="12" onblur="checkLast(event)"
            aria-required="true" aria-invalid="false" />
          <div class="error" id="lname_error" role="alert"/>
        </div>
      </form>
    </div>
  </div>
</body>
<div class="error" id="lname_error"></div>
</div>
</div>
<div class="text">
<label for="date">Birth Date</label>
<input type="text"
    name="date"
    id="date"
    size="12"
    onblur="checkDate(event)"
    aria-required="false"
    aria-invalid="false"
    aria-describedby="date_error"/>
<div class="error" id="date_error"></div>
</div>
</div>
<div class="text">
<label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
<input type="text"
    name="ssn"
    id="ssn"
    size="10"
    onblur="checkSSN(event)"
    aria-required="false"
    aria-invalid="false"
    aria-describedby="ssn_error"/>
<div class="error" id="ssn_error"></div>
</div>
</div>
</div>
<div class="text">
<label for="phone">Phone</label>
<input type="text"
    name="phone"
    id="phone"
    size="14"
    onblur="checkPhone(event)"
    aria-required="true"
    aria-invalid="false"/>
Results

Running ./tests/test17.js
Running all tests...
   Checking for skip to main content link...
      Skip to main link found: false
   Checking for nav menu consistency...
      Sites checked: 4
      Sites inconsistent: 0
      Consistent navigation menu: true
   Checking for form ARIA compliance...
      Tests complete.
         Error class tags missing "role: 'alert'":
            id = lname_error
            id = ssn_error
      All form input ARIA compliant: false

Tests complete.

Site 18

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
```
<!--Form code taken from http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3-->
<head>
<style type="text/css">
div.text,
div.textinvalid,
div.button
{
  margin: 0;
  padding: 0;
  padding-left: 20px;
  padding-bottom: .5em;
  display: block;
}
div.text label,
div.textinvalid label {
  margin: 0;
  padding: 0;
  display: block;
  padding-top: .25em;
}
div.text input,
div.textinvalid input {
  margin: 0;
  padding: 0;
  display: inline;
}
span.inst {
  font-size: 75%;
  color: blue;
  padding-left: .25em;
}
div.text input:active,
div.text input:focus,
div.text input:hover
{
  border-color: gray;
  background-color: #E0E0E0;
}
div.textinvalid input:active,
div.textinvalid input:focus,
div.textinvalid input:hover
{
  border-color: gray;
  background-color: #FF8080;
}
function validateForm() {
    var errorMessage = "Please complete the following fields:";
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;
    // check required fields
    // Check first name
    node = document.getElementById("fname");
    if( node && node.value == "" ) {
        result = "You must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }
    // Check last name
    node = document.getElementById("lname");
    if( node && node.value == "" ) {
        result = "You must enter your last name";
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }
    // Check date
    node = document.getElementById("date");
    if( node ) {
        result = validateDate( node );
        if( result != "" ) {
            errorElements.push(result);
        }
    }
}
showError(node, result);
if( node_focus == null )
    node_focus = node;
} // end if
} // end if
// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if
// Check Phone
node = document.getElementById("phone");
if( node && node.value == "") {
    result = "\nYou must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        } // end if
    } // end if
} // end if else
if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++ ) {
        errorMessage += "\n" + errorElements[i];
    } // end for
    alert(errorMessage);
    if( node_focus )
        node_focus.focus();
    return false;
} else {
    // suppress submit for test; for a real form, return true
alert("If this were a real form, you would have just submitted it.");
    return false;
}

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkDate(event) {
    var node = getTarget(event);
    if( node ) {
        var result = validateDate( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateDate(node) {
    var errorMessage = "Date needs to be in date format, such as 1/31/2001.";
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
if (node.value.match(/\D\d{1,2}$/) &&
dateValue.getFullYear() < (new Date().getFullYear() - 96))
{
    dateValue.setFullYear(dateValue.getFullYear() +
    100);
}

// format as mm/dd/yyyy
node.value = (dateValue.getMonth() + 1) + "/" +
dateValue.getDate() + "/" + dateValue.getFullYear();
return "";

} else {
    return errorMessage;
}

return "";

} function checkPhone(event) {
    var node = getTarget(event);
    if (node) {
        var result = validatePhone(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validatePhone(node) {
    var errorMessage = "Phone needs to be a phone
number with area code, such as (555) 123-4567.";
    if (node.value != "") {
        // replace all non-digit characters, then match
        string starting with 2-9 (US phone numbers can't start with
        0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/([2-9]\d{2})(\d{3})(\d{4})/)) {
            // use sub-matches captured above to format as
            (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "You must enter a phone number, such as
(555) 123-4567";
function checkSSN(event) {
    var node = getTarget(event);
    if (node) {
        var result = validateSSN(node);
        showError(node, result);
    }  // endif
    return stopPropagation(event);
}

function validateSSN(node) {
    var errorMessage = "Social Security Number needs to be a nine-digit number, such as 123-45-6789.";
    if (node.value != ") {
        // replace all non-digit characters, then match 3 digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/(^\d{3})(\d{2})(\d{4})/) ) {
            // use sub-matches captured above to format as ###-##-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById(id);
    if (node_error )
        node_error.innerHTML = "<span class='offscreen'>Invalid: </span>" + message;
    if (message == "") {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
    }
}

function getTarget(event) {
    var e = event || window.event;
    if (e.target )
        return e.target;
    else
return e.srcElement;
}

function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble )
        event.cancelBubble = true;
    if( event.returnValue )
        event.returnValue = false;
    return false;
}

// --></!]>
</script>
</head>
<body>
<h2>Navigation Menu</h2>
<ul class="nav-list-primary" role='navigation'>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1'">Site 1</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17'">Site 17</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20'">Site 20</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22'">Site 22</a></li>
</ul>
<div class="example">
    <ul class="nav-secondary">
    </ul>
</div>
<h2>Sample Form</h2>
<p>Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.</p>
<form id="SampleForm" action="#" method="post">
    <div class="text">
        <label for="fname">
            First Name
            <img src="images/required.png" alt=""/>
        </label>
        <input type="text" name="fname">
    </div>
id="fname"
size="12"
onblur="checkFirst(event)"
aria-required="true"
aria-invalid="false"
aria-describedby="fname_error"/>
</div>
</div>

<div class="text">
<label for="lname">
  Last Name
  <img src="images/required.png" alt="" />
</label>
<input type="text"
  name="lname"
  id="lname"
  size="20"
onblur="checkLast(event)"
  aria-required="true"
  aria-invalid="false"
  aria-describedby="lname_error"/>
</div>

<div class="text">
<label for="date">Birth Date</label>
<input type="text"
  name="date"
  id="date"
  size="12"
onblur="checkDate(event)"
  aria-required="false"
  aria-invalid="false"
  aria-describedby="date_error"/>
</div>

<div class="text">
<label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
<input type="text"
  name="ssn"
  id="ssn"
  size="10"
onblur="checkSSN(event)"
  aria-required="false"
  aria-invalid="false"
<div class="error" id="ssn_error" role="alert"></div>
</div>
<div class="text">
<label for="phone">
  Phone
  <img src="images/required.png" alt="Required"
</label>
<input type="text"
   name="phone"
   id="phone"
   size="14"
   onblur="checkPhone(event)"
   aria-required="true"
   aria-invalid="false"
   aria-describedby="phone_error"/>
<div class="error" id="phone_error" role="alert"></div>
</div>
<div class="button">
<input type="button"
   value="Check Form"
   onclick=" validateForm()"
   onclick=" validateForm()"/>
</div>
</form>
<p><a href="http://www.msfw.com/accessibility/tests/ClientSideValidationTest.HTML">Example was developed from MSF&W Accessibility Tests</a></p>
</body>
</html>

Results

Running ./tests/test18.js
Running all tests...
  Checking for skip to main content link...
  Skip to main link found: false

  Checking for nav menu consistency...
https://csel.cs.colorado.edu/~erdu8260/site1
https://csel.cs.colorado.edu/~erdu8260/site17
https://csel.cs.colorado.edu/~erdu8260/site20
https://csel.cs.colorado.edu/~erdu8260/site22
Sites checked: 5
Sites inconsistent: 4
Consistent navigation menu: false

Checking for form ARIA compliance...
All form input ARIA compliant: true

Tests complete.

Site 19

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<!--Form code taken from http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3--!>
<head>
<style type="text/css">
div.text, 
div.textinvalid, 
div.button 
{
    margin: 0;
    padding: 0;
    padding-left: 20px;
    padding-bottom: .5em;
    display: block;
}
div.text label, 
div.textinvalid label 
{
    margin: 0;
    padding: 0;
    display: block;
    padding-top: .25em;
}
div.text input, 
div.textinvalid input 
{
    margin: 0;
    padding: 0;
    display: inline;
}
span.inst 
{
    font-size: 75%;
}
color: blue;
  padding-left: .25em;
}
div.text input:active,
div.text input:focus,
div.text input:hover
{
  border-color: gray;
  background-color: #E0E0E0;
}
div.textinvalid input:active,
div.textinvalid input:focus,
div.textinvalid input:hover
{
  border-color: gray;
  background-color: #FF8080;
}
form div.error {
  display: none;
  border: thin solid red;
  padding: .25em;
  color: red;
  font-size: 80%;
}
.offscreen {
  position: absolute;
  top: -30em;
  left: -300em;
}
</style>
<script type="text/javascript">
<!--//--><![CDATA[<!--

function validateForm() {
  var errorMessage = "Please complete the following fields:";
  var errorElements = new Array();
  var node;
  var result;
  var node_focus = null;
  // check required fields
  // Check first name
  var node = document.getElementById("fname");
  if( node && node.value == "") {
    result = "You must enter your first name";
    errorElements.push(result);
    showError(node, result);
node_focus = node;
}

// Check last name
node = document.getElementById("lname");
if( node && node.value == "" ) {
    result = "You must enter your last name";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
}

// Check date
node = document.getElementById("date");
if( node ) {
    result = validateDate( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if

// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if

// Check Phone
node = document.getElementById("phone");
if( node && node.value == "" ) {
    result = "You must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
        } // end if
    } // end if
} // end if
errorElements.push(result);
showError(node, result);
if( node_focus == null )
    node_focus = node;
} // end if
} // end if
} // end if else
if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++) {
        errorMessage += "\n" + errorElements[i];
    } // end for
    alert(errorMessage);
    if( node_focus )
        node_focus.focus();
    return false;
} else {
    // supress submit for test; for a real form, return true
    alert("If this were a real form, you would have just submitted it.");
    return false;
}

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}
function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}
function checkDate(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        result = "You must enter your date of birth";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}
if( node ) {
    var result = validateDate( node);
    showError(node, result);
} // endif
return stopPropagation( event);

function validateDate(node) {
    var errorMessage = "\nDate needs to be in date format, \nsuch as 1/31/2001."
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
            if (node.value.match(/D\d{1,2}$/) &&
                dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
                dateValue.setFullYear(dateValue.getFullYear() + 100);
            }
            // format as mm/dd/yyyy
            node.value = (dateValue.getMonth() + 1) + "/" +
                dateValue.getDate() + "/" + dateValue.getFullYear();
            return "";
        } else {
            return errorMessage;
        }
    }
return "";
}

function checkPhone(event ) {
    var node = getTarget( event );
    if( node ) {
        var result = validatePhone( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validatePhone( node ) {
    var errorMessage = "\nPhone needs to be a phone number with area code, \nsuch as (555) 123-4567."
    if (node.value != "") {
// replace all non-digit characters, then match
string starting with 2-9 (US phone numbers can't start with
0 or 1) followed by 2 digits, 3 digits, 4 digits
if (node.value.replace(/\D/g, "").match(/([2-9]\d{2})\d{3}\d{4}/)) {
    // use sub-matches captured above to format as
    (###) ###-####
    node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
    return "";
} else {
    return errorMessage;
}
return "\nYou must enter a phone number, \nsuch as
(555) 123-4567";
}
function checkSSN(event) {
    var node = getTarget( event);
    if( node ) {
        var result = validateSSN( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
} function validateSSN(node) {
    var errorMessage = "\nSocial Security Number needs to
be a nine-digit number, \nsuch as 123-45-6789.";
    if (node.value != "") {
        // replace all non-digit characters, then match 3
digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g,
            "").match(/([\d{3}])(\d{2})(\d{4})/)) ) {
            // use sub-matches captured above to format as
            ###-###-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" +
            RegExp.$3;
            return "";
        }
    else {
            return errorMessage;
    }
    return "";
}
function showError(node, message) {
var id = node.id + "_error";
var node_error = document.getElementById( id );
if( node_error )
    node_error.innerHTML = "<span class='offscreen'>Invalid: </span>" + message;
if( message == "" ) {
    node.setAttribute('aria-invalid', 'false');
    node_error.style.display = "none";
} else {
    node.setAttribute('aria-invalid', 'true');
    node_error.style.display = "inline";
}
}

function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
        return e.srcElement;
}

function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble)
        event.cancelBubble = true;
    if(event.returnValue)
        event.returnValue = false;
    return false;
}

//---></![endif]
</script>
</head>
<body>

<div id="skip-link">
    <a href="#main-content" class="element-invisible element-focusable">Skip to main content</a>
</div>

<h2>Navigation Menu</h2>
<ul class="nav-list-primary" role='navigation'>
    <li class="nav-list-item"><a href='https://csei.cs.colorado.edu/~erdu8260/site1' >Site 1</a></li>
    <li class="nav-list-item"><a href='https://csei.cs.colorado.edu/~erdu8260/site17' >Site 17</a></li>
</ul>
Sample Form

Fields marked with an <img src="images/required.png" class="required" alt="Required"/> are required.

<form id="SampleForm" action="#" method="post">
  <div class="text">
    <label for="fname">
      First Name
      <img src="images/required.png" alt=""/>
    </label>
    <input type="text"
      name="fname"
      id="fname"
      size="12"
      onblur="checkFirst(event)"
      aria-required="true"
      aria-invalid="false"
      aria-describedby="fname_error"/>
    <div class="error" id="fname_error"></div>
  </div>
  <div class="text">
    <label for="lname">
      Last Name
      <img src="images/required.png" alt=""/>
    </label>
    <input type="text"
      name="lname"
      id="lname"
      size="20"
      onblur="checkLast(event)"
      aria-required="true"
      aria-invalid="false"/>
    <div class="error" id="lname_error" role="alert"></div>
  </div>
  <div class="text">
    <label for="date">
      Birth Date
    </label>
  </div>
</form>
<input type="text"
    name="date"
    id="date"
    size="12"
    onblur="checkDate(event)"
    aria-required="false"
    aria-invalid="false"
    aria-describedby="date_error"/>
</div>
    role="alert">
</div>
<div class="text">
    <label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
    <input type="text"
        name="ssn"
        id="ssn"
        size="10"
        onblur="checkSSN(event)"
        aria-required="false"
        aria-invalid="false"
        aria-describedby="ssn_error"/>
</div>
    role="alert">
</div>
    <div class="text">
        <label for="phone">Phone</label>
        <img src="/images/required.png" alt="Required" />
    </div>
        <input type="text"
            name="phone"
            id="phone"
            size="14"
            onblur="checkPhone(event)"
            aria-required="true"
            aria-invalid="false"
        />
</div>
    role="alert">
</div>
<div class="button">
    <input type="button"
        value="Check Form"
        onclick=" validateForm()"/>
</div>
</form>
Results

Running ./tests/test19.js
Running all tests...
  Checking for skip to main content link...
  Skip to main link found: true

  Checking for nav menu consistency...
  https://csel.cs.colorado.edu/~erdu8260/site1
  https://csel.cs.colorado.edu/~erdu8260/site17
  https://csel.cs.colorado.edu/~erdu8260/site20
  https://csel.cs.colorado.edu/~erdu8260/site22
  Sites checked: 6
  Sites inconsistent: 4
  Consistent navigation menu: false

  Checking for form ARIA compliance...
  Error class tags missing "role: 'alert'":
    id = fname_error
    id = phone_error
  All form input ARIA compliant: false

Tests complete.

Site 20

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<head>
<style type="text/css">
div.text,
div.textinvalid,
div.button
{ margin: 0;
 padding: 0;
 padding-left: 20px;
 padding-bottom: .5em;
 display: block;
 }
div.text label,
div.textinvalid label {
 margin: 0;
 padding: 0;
 display: block;
 padding-top: .25em;
 }
div.text input,
div.textinvalid input {
 margin: 0;
 padding: 0;
 display: inline;
 }
span.inst {
 font-size: 75%;
 color: blue;
 padding-left: .25em;
 }
div.text input:active,
div.text input:focus,
div.text input:hover {
 border-color: gray;
 background-color: #E0E0E0;
 }
div.textinvalid input:active,
div.textinvalid input:focus,
div.textinvalid input:hover {
 border-color: gray;
 background-color: #FF8080;
 }
form div.error {
 display: none;
 border: thin solid red;
 padding: .25em;
 color: red;
 font-size: 80%;
 }
.offscreen {
function validateForm() {
    var errorMessage = "Please complete the following fields:"
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;
    // check required fields
    // Check first name
    var node = document.getElementById("fname");
    if( node && node.value == "" ) {
        result = "You must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }
    // Check last name
    node = document.getElementById("lname");
    if( node && node.value == "" ) {
        result = "You must enter your last name";
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }
    // Check date
    node = document.getElementById("date");
    if( node ) {
        result = validateDate( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        } // end if
    } // end if
    // Check SSN
    node = document.getElementById("ssn");
    if( node ) {

result = validateSSN(node);
if (result != "") {
    errorElements.push(result);
    showError(node, result);
    if (node_focus == null)
        node_focus = node;
} // end if
} // end if
// Check Phone
node = document.getElementById("phone");
if (node && node.value == "") {
    result = "You must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if (node_focus == null)
        node_focus = node;
} else {
    if (node) {
        result = validatePhone(node);
        if (result != "") {
            errorElements.push(result);
            showError(node, result);
            if (node_focus == null)
                node_focus = node;
        } // end if
    } // end if
} // end if else
if (node_focus != null) {
    for (i = 0; i < errorElements.length; i++) {
        errorMessage += 
        + errorElements[i];
    } // end for
    alert(errorMessage);
    if (node_focus)
        node_focus.focus();
    return false;
} else {
    // suppress submit for test; for a real form, return true
    alert("If this were a real form, you would have just submitted it.");
    return false;
}
}
function checkFirst(event) {
    var node = getTarget(event);
    var result;
if( node.value == "" ) {
    var result = "You must enter your first name";
} else {
    result = "";
} // endif
showError(node, result);
return stopPropagation( event);
}

function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkDate(event) {
    var node = getTarget(event);
    if( node ) {
        var result = validateDate( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateDate(node) {
    var errorMessage = "Date needs to be in date format, such as 1/31/2001.";
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
            if (node.value.match(/\D\d{1,2}$/) &&
                dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
                dateValue.setFullYear(dateValue.getFullYear() + 100);
            }
            // format as mm/dd/yyyy
            node.value = (dateValue.getMonth() + 1) + "/" +
            dateValue.getDate() + "/" + dateValue.getFullYear();
            return "";
        }
    }
}
else {
    return errorMessage;
}
}

function checkPhone(event) {
    var node = getTarget(event);
    if (node) {
        var result = validatePhone(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validatePhone(node) {
    var errorMessage = "Phone number needs to be a phone number with area code, such as (555) 123-4567."
    if (node.value != ")") {
        // replace all non-digit characters, then match string starting with 2-9 (US phone numbers can't start with 0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/\D/g, ")").match(/(\d{2})(\d{3})(\d{4})/)) {
            // use sub-matches captured above to format as (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "You must enter a phone number, such as (555) 123-4567";
}

function checkSSN(event) {
    var node = getTarget(event);
    if (node) {
        var result = validateSSN(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validateSSN(node) {
var errorMessage = "\nSocial Security Number needs to be a nine-digit number, such as 123-45-6789.";
    if (node.value != "") {
        // replace all non-digit characters, then match 3 digits, 2 digits, 4 digits
        if (node.value.replace(/[\D]/g, "").match(/([\d]{3})([\d]{2})([\d]{4})/)) {
            // use sub-matches captured above to format as ###-##-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    } else {
    return "";
}
function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById( id );
    if( node_error )
        node_error.innerHTML = "<span class='offscreen'>Invalid: </span>" + message;
    if( message == "" ) {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
    }
}
function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
        return e.srcElement;
}
function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble )
        event.cancelBubble = true;
if(event.returnValue)
    event.returnValue = false;
return false;

//--><![>]
</script>
</head>
<body>
<nav id="primaryNav">
    <h2>Navigation Menu</h2>
    <ul class="nav-list primary">
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1' ">Site 1</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17' ">Site 17</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20' ">Site 20</a></li>
        <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22' ">Site 22</a></li>
    </ul>
</nav>
<div class="example">
    <div>
        <h2>Sample Form</h2>
        <p>Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.</p>
        <form id="SampleForm" action="#" method="post">
            <div class="text">
                <label for="fname">First Name
                    <img src="images/required.png" alt=""/>
                </label>
                <input type="text" name="fname" id="fname" size="12"
                    onblur="checkFirst(event)"
                    aria-required="true"
                    aria-invalid="false"
                    aria-describedby="fname_error"/>
            </div>
        </form>
    </div>
</div>
<div class="text">
<label for="lname">
  Last Name
  <img src="images/required.png" alt="" />
</label>
<input type="text"
  name="lname"
  id="lname"
  size="20"
  onblur="checkLast(event)"
  aria-required="true"
  aria-invalid="false"
  aria-describedby="lname_error"/>
<div class="error" id="lname_error" role="alert"></div>
</div>
<div class="text">
<label for="date">Birth Date</label>
<input type="text"
  name="date"
  id="date"
  size="12"
  onblur="checkDate(event)"
  aria-required="false"
  aria-invalid="false"
  aria-describedby="date_error"/>
<div class="error" id="date_error" role="alert"></div>
</div>
<div class="text">
<label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
<input type="text"
  name="ssn"
  id="ssn"
  size="10"
  onblur="checkSSN(event)"
  aria-required="false"
  aria-invalid="false"
  aria-describedby="ssn_error"/>
<div class="error" id="ssn_error" role="alert"></div>
</div>
<div class="text">
<label for="phone">Phone</label>
</div>
<img src="images/required.png" alt="Required"/>

</label>
<input type="text"
    name="phone"
    id="phone"
    size="14"
    onblur="checkPhone(event)"
    aria-required="true"
    aria-invalid="false"
    aria-describedby="phone_error"/>

<div class="error" id="phone_error"
    role="alert"></div>

</div>
</div>

<div class="button">
<input type="button"
    value="Check Form"
    onclick=" validateForm()"/>
</div>
</form>

<p><a href="http://www.msfw.com/accessibility/tests/ClientSideValidationTest.HTML">Example was developed from MSFW Accessibility Tests</a></p>

<Results>

Running ./tests/test20.js
Running all tests...
Checking for skip to main content link...
Skip to main link found: false

Checking for nav menu consistency...
Sites checked: 4
Sites inconsistent: 0
Consistent navigation menu: true

Checking for form ARIA compliance...
All form input ARIA compliant: true

Tests complete.
Site 21

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<!--Form code taken from http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3--!>
<head>
<style type="text/css">
  div.text, div.textinvalid, div.button
  {
    margin: 0;
    padding: 0;
    padding-left: 20px;
    padding-bottom: .5em;
    display: block;
  }
  div.text label, div.textinvalid label {
    margin: 0;
    padding: 0;
    display: block;
    padding-top: .25em;
  }
  div.text input, div.textinvalid input {
    margin: 0;
    padding: 0;
    display: inline;
  }
  span.inst {
    font-size: 75%;
    color: blue;
    padding-left: .25em;
  }
  div.text input:active, div.text input:focus, div.text input:hover
  {
    border-color: gray;
    background-color: #E0E0E0;
  }
</head>
function validateForm() {
    var errorMessage = "Please complete the following fields:"
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;
    // check required fields
    // Check first name
    var node = document.getElementById("fname");
    if( node && node.value == "" ) {
        result = "You must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }
    // Check last name
    node = document.getElementById("lname");
    if( node && node.value == "" ) {
        result = "You must enter your last name";
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }
}

<script type="text/javascript">
</script>
node_focus = node;
}

// Check date
node = document.getElementById("date");
if( node ) {
    result = validateDate( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }  // end if
}  // end if

// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }  // end if
}  // end if

// Check Phone
node = document.getElementById("phone");
if( node && node.value == "") {
    result = \nYou must enter a phone number, \nsuch as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        }  // end if
    }  // end if
}  // end if else
if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++ ) {
        errorMessage += \n" + errorElements[i];
function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkLast(event) {
    var node = getTarget(event);
    var result;
    if( node && node.value == "" ) {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkDate(event) {
    var node = getTarget(event);
    if( node ) {
        var result = validateDate( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateDate(node) {
    var errorMessage = "\nDate needs to be in date format, \nsuch as 1/31/2001.";
}
if (node.value != "") {
    // try parsing as date using JavaScript Date constructor
    var dateValue = new Date(node.value.replace(/-/g, "/"));
    if (isFinite(dateValue)) {
        // if two-digit year, guess at correct century
        if (node.value.match(/\D\d{1,2}$/) &&
            dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
            dateValue.setFullYear(dateValue.getFullYear() + 100);
        }
        // format as mm/dd/yyyy
        node.value = (dateValue.getMonth() + 1) + "/" +
            dateValue.getDate() + "/" + dateValue.getFullYear();
        return "";
    } else {
        return errorMessage;
    }
} else {
    return "";
}

function checkPhone(event ) {
    var node = getTarget( event );
    if( node ) {
        var result = validatePhone( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validatePhone( node ) {
    var errorMessage = "\nPhone needs to be a phone number with area code, \nsuch as (555) 123-4567.";
    if (node.value != "") {
        // replace all non-digit characters, then match string starting with 2-9 (US phone numbers can't start with 0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/\D/g, ").match(/([2-9]\d{2})(\d{3})(\d{4})/)) {
            // use sub-matches captured above to format as (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
            return "";
        }
    }
else {
    return errorMessage;
}

return "\nYou must enter a phone number, \nsuch as (555) 123-4567";
}

function checkSSN(event) {
    var node = getTarget( event );
    if( node ) {
        var result = validateSSN( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateSSN(node) {
    var errorMessage = "\nSocial Security Number needs to
be a nine-digit number, \nsuch as 123-45-6789.";
    if (node.value != "") {
        // replace all non-digit characters, then match 3
digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/(^\d{3})(\d{2})(\d{4})/)) {
            // use sub-matches captured above to format as
            ###-##-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" +
RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById( id );
    if( node_error )
        node_error.innerHTML = "<span
class='offscreen'>Invalid: </span>" + message;
    if( message == "" ) {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
        return e.srcElement;
}

function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble )
        event.cancelBubble = true;
    if( event.returnValue )
        event.returnValue = false;
    return false;
}

//---</![CDATA[
</script>
</head>
<body>
<div id="skip-link">
    <a href="#main-content" class="element-invisible element-focusable">Skip to main content</a>
</div>
<h2>Navigation Menu</h2>
<ul class="nav-list-primary" role='navigation'>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1'">Site 1</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17'">Site 17</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20'">Site 20</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22'">Site 22</a></li>
</ul>
</body>
<div class="example">
    <h2>Sample Form</h2>
Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.

<form id="SampleForm" action="#" method="post">
  <div class="text">
    <label for="fname">
      First Name
    </label>
    <input type="text"
      name="fname"
      id="fname"
      size="12"
      onblur="checkFirst(event)"
      aria-required="true"
      aria-invalid="false"
      aria-describedby="fname_error"/>
    <div class="error" id="fname_error" role="alert"></div>
  </div>
  <div class="text">
    <label for="lname">
      Last Name
    </label>
    <input type="text"
      name="lname"
      id="lname"
      size="20"
      onblur="checkLast(event)"
      aria-required="true"
      aria-invalid="false"
      aria-describedby="lname_error"/>
    <div class="error" id="lname_error" role="alert"></div>
  </div>
  <div class="text">
    <label for="date">Birth Date</label>
    <input type="text"
      name="date"
      id="date"
      size="12"
      onblur="checkDate(event)"
      aria-required="false"
      aria-invalid="false"
      aria-describedby="date_error"/>
    <div class="error" id="date_error" role="alert"></div>
  </div>
</form>
<div class="text">
  <label for="ssn"><abbr title="Social Security Number">SSN</abbr></label><input type="text" name="ssn" id="ssn" size="10" onblur="checkSSN(event)" aria-required="false" aria-invalid="false" aria-describedby="ssn_error"/>
</div>

<div class="error" id="ssn_error" role="alert"></div>

<div class="text">
  <label for="phone">Phone</label><input type="text" name="phone" id="phone" size="14" onblur="checkPhone(event)" aria-required="true" aria-invalid="false" aria-describedby="phone_error"/>
</div>

<div class="error" id="phone_error" role="alert"></div>

<div class="button">
  <input type="button" value="Check Form" onclick="validateForm()"/>
</div>

</form>

<p><a href="http://www.msfw.com/accessibility/tests/ClientSideValidationTest.HTML">Example was developed from MSFW Accessibility Tests</a></p>
</body>
</HTML>
Results

Running ./tests/test21.js
Running all tests...
  Checking for skip to main content link...
  Skip to main link found: true

  Checking for nav menu consistency...
  https://csel.cs.colorado.edu/~erdu8260/site1
  https://csel.cs.colorado.edu/~erdu8260/site17
  https://csel.cs.colorado.edu/~erdu8260/site20
  https://csel.cs.colorado.edu/~erdu8260/site22
  Sites checked: 6
  Sites inconsistent: 4
  Consistent navigation menu: false

  Checking for form ARIA compliance...
  All form input ARIA compliant: true

Tests complete.

Site 22

Code

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
  <!--Form code taken from
http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3-->!
  <head>
  <style type="text/css">
  div.text, 
  div.textinvalid, 
  div.button 
  { 
    margin: 0;
    padding: 0;
    padding-left: 20px;
    padding-bottom: .5em;
    display: block;
  }
  div.text label, 
  div.textinvalid label { 
    margin: 0;
  }
```
function validateForm() {
}
var errorMessage = "Please complete the following fields;"
var errorElements = new Array();
var node;
var result;
var node_focus = null;
// check required fields
// Check first name
// Check last name
node = document.getElementById("fname");
if( node && node.value == """) { 
    result = "You must enter your first name";
    errorElements.push(result);
    showError(node, result);
    node_focus = node;
}
// Check last name
node = document.getElementById("lname");
if( node && node.value == """) { 
    result = "You must enter your last name";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
}
// Check date
node = document.getElementById("date");
if( node ) {
    result = validateDate( node );
    if( result != """) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if
// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != """) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if
// Check Phone
node = document.getElementById("phone");
if( node && node.value == "") {
    result = "\nYou must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        } // end if
    } // end if
} // end if else
if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++ ) {
        errorMessage += "\n" + errorElements[i];
    } // end for
    alert(errorMessage);
    if( node_focus )
        node_focus.focus();
    return false;
} else {
    // supress submit for test; for a real form, return true
    alert("If this were a real form, you would have just submitted it.");
    return false;
}

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if( node.value == "" ) {
        var result = "You must enter your first name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}
function checkLast(event) {
    var node = getTarget(event);
    var result;
    if(node && node.value == "") {
        result = "You must enter your last name";
    } else {
        result = "";
    } // endif
    showError(node, result);
    return stopPropagation( event);
}

function checkDate(event) {
    var node = getTarget(event);
    if(node) {
        var result = validateDate( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateDate(node) {
    var errorMessage = "\nDate needs to be in date format, such as 1/31/2001.";
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
            if (node.value.match(/\D{1,2}$/) &&
                dateValue.getFullYear() < (new Date().getFullYear() - 96))
                {  
                dateValue.setFullYear(dateValue.getFullYear() + 100);
            }
            // format as mm/dd/yyyy
            node.value = (dateValue.getMonth() + 1) + "/" +
            dateValue.getDate() + "/" + dateValue.getFullYear();
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function checkPhone(event) {
var node = getTarget( event );
    if( node ) {
        var result = validatePhone( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validatePhone( node ) {
    var errorMessage = "\nPhone needs to be a phone
number with area code, \nsuch as (555) 123-4567.";
    if (node.value != "") {
        // replace all non-digit characters, then match
        string starting with 2-9 (US phone numbers can't start with
        0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/([2-9]\d{2})\d{3}\d{4}/)) {
            // use sub-matches captured above to format as
            (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 + "-" + RegExp.$3;
            return "";
        }
        else {
            return errorMessage;
        }
    }
    return "\nYou must enter a phone number, \nsuch as
(555) 123-4567";
}

function checkSSN(event) {
    var node = getTarget( event );
    if( node ) {
        var result = validateSSN( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateSSN(node) {
    var errorMessage = "\nSocial Security Number needs to
be a nine-digit number, \nsuch as 123-45-6789.";
    if (node.value != "") {
        // replace all non-digit characters, then match 3
digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g, ") .match(/((\d{3})(\d{2})(\d{4}))/) ) {
            // use sub-matches captured above to format as
            ###-###-####
            return "";
        }
        else {
            return errorMessage;
        }
    }
    return "\nYou must enter a Social Security Number, \nsuch as 123-45-6789";
node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
    return ""
} else {
    return errorMessage;
}
return "";
}
function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById( id );
    if( node_error )
        node_error.innerHTML = "<span class='offscreen'>Invalid: </span>" + message;
    if( message == "" ) {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
    }
}
function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
        return e.srcElement;
}
function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble )
        event.cancelBubble = true;
    if(event.returnValue)
        event.returnValue = false;
    return false;
}
//--><!]></script></head></body><div id="skip-link"
<a href="#main-content" class="element-invisible element-focusable">Skip to main content</a>
</div>
<nav id="primaryNav">
<h2>Navigation Menu</h2>
<ul class="nav-list-primary">
<li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1' >Site 1</a></li>
<li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17' >Site 17</a></li>
<li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20' >Site 20</a></li>
<li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22' >Site 22</a></li>
</ul>
</nav>

<div class="example">
<h2>Sample Form</h2>
<p>Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.</p>
<form id="SampleForm" action="#" method="post">
<div class="text">
<label for="fname">
First Name
<img src="images/required.png" alt=""/>
</label>
<input type="text" name="fname" id="fname" size="12" onblur="checkFirst(event)"
aria-required="true"
aria-invalid="false"/>
</div>
</form>

</div>

</div>
<input type="text"
    name="lname"
    id="lname"
    size="20"
    onblur="checkLast(event)"
    aria-required="true"
    aria-invalid="false"
    aria-describedby="lname_error"/>
<div class="error" id="lname_error"></div>
</div>
<div class="text">
    <label for="date">Birth Date</label>
    <input type="text"
        name="date"
        id="date"
        size="12"
        onblur="checkDate(event)"
        aria-required="false"
        aria-invalid="false"
        aria-describedby="date_error"/>
    <div class="error" id="date_error"></div>
</div>
<div class="text">
    <label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
    <input type="text"
        name="ssn"
        id="ssn"
        size="10"
        onblur="checkSSN(event)"
        aria-required="false"
        aria-invalid="false"
        aria-describedby="ssn_error"/>
    <div class="error" id="ssn_error" role="alert"></div>
</div>
<div class="text">
    <label for="phone">Phone</label>
    <input type="text"
        name="phone"
        id="phone"
        size="14"
        onblur="checkPhone(event)"
/>
aria-required="true"
aria-invalid="false"
aria-describedby="phone_error"/>

<div class="error" id="phone_error" role="alert"></div>
</div>
</div>
<div class="button">
<input type="button" value="Check Form" value="Check Form"
onclick=" validateForm()"/>
</div>
</form>
<p><a href="http://www.msfw.com/accessibility/tests/ClientSideValidationTest.HTML">Example was developed from MSF&amp;W Accessibility Tests</a></p>
</body>
</HTML>

Results

Running ./tests/test22.js
Running all tests...
Checking for skip to main content link...
Skip to main link found: true

Checking for nav menu consistency...
Sites checked: 5
Sites inconsistent: 0
Consistent navigation menu: true

Checking for form ARIA compliance...
Tests complete.
Error class tags missing "role: 'alert'":
  id = fname_error
  id = lname_error
  id = date_error
All form input ARIA compliant: false

Tests complete.

Site 23

Code

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
<HTML xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb">
<!--Form code taken from http://HTML.cita.illinois.edu/nav/form/aria/index.php?example=3--!>
<head>
<style type="text/css">
div.text, 
div.textinvalid, 
div.button 
{
    margin: 0;
    padding: 0;
    padding-left: 20px;
    padding-bottom: .5em;
    display: block;
}
div.text label, 
div.textinvalid label { 
    margin: 0;
    padding: 0;
    display: block;
    padding-top: .25em;
}
div.text input, 
div.textinvalid input { 
    margin: 0;
    padding: 0;
    display: inline;
}
span.inst {
    font-size: 75%;
    color: blue;
    padding-left: .25em;
}
div.text input:active, 
div.text input:focus, 
div.text input:hover 
{
    border-color: gray;
    background-color: #E0E0E0;
}
div.textinvalid input:active, 
div.textinvalid input:focus, 
div.textinvalid input:hover 
{
    border-color: gray;
}
function validateForm() {
    var errorMessage = "Please complete the following fields:";
    var errorElements = new Array();
    var node;
    var result;
    var node_focus = null;

    // Check required fields
    // Check first name
    node = document.getElementById("fname");
    if( node && node.value == "" ) {
        result = "You must enter your first name";
        errorElements.push(result);
        showError(node, result);
        node_focus = node;
    }

    // Check last name
    node = document.getElementById("lname");
    if( node && node.value == "" ) {
        result = "You must enter your last name";
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    }

    // Check date
    node = document.getElementById("date");
    if( node )
        result = validateDate( node );
}
if( result != "" ) {
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} // end if
} // end if
// Check SSN
node = document.getElementById("ssn");
if( node ) {
    result = validateSSN( node );
    if( result != "" ) {
        errorElements.push(result);
        showError(node, result);
        if( node_focus == null )
            node_focus = node;
    } // end if
} // end if
// Check Phone
node = document.getElementById("phone");
if( node && node.value == "") {
    result = "You must enter a phone number, such as (555) 123-4567 ";
    errorElements.push(result);
    showError(node, result);
    if( node_focus == null )
        node_focus = node;
} else {
    if( node ) {
        result = validatePhone( node );
        if( result != "" ) {
            errorElements.push(result);
            showError(node, result);
            if( node_focus == null )
                node_focus = node;
        } // end if
    } // end if
} // end if else
if( node_focus != null ) {
    for(i=0; i < errorElements.length; i++ ) {
        errorMessage += "\n" + errorElements[i];
    } // end for
    alert(errorMessage);
    if( node_focus )
        node_focus.focus();
    return false;
} else {

function checkFirst(event) {
    var node = getTarget(event);
    var result;
    if (node.value == "") {
        result = "You must enter your first name";
    } else {
        result = ""
    } // endif
    showError(node, result);
    return stopPropagation(event);
}

function checkLast(event) {
    var node = getTarget(event);
    var result;
    if (node && node.value == "") {
        result = "You must enter your last name";
    } else {
        result = ""
    } // endif
    showError(node, result);
    return stopPropagation(event);
}

function checkDate(event) {
    var node = getTarget(event);
    if (node) {
        var result = validateDate(node);
        showError(node, result);
    } // endif
    return stopPropagation(event);
}

function validateDate(node) {
    var errorMessage = "Date needs to be in date format, such as 1/31/2001.";
    if (node.value != "") {
        // try parsing as date using JavaScript Date constructor
        var dateValue = new Date(node.value.replace(/-/g, "/"));
        if (isFinite(dateValue)) {
            // if two-digit year, guess at correct century
if (node.value.match(/\D\d{1,2}$/) &&
dateValue.getFullYear() < (new Date().getFullYear() - 96)) {
    dateValue.setFullYear(dateValue.getFullYear() + 100);
}
    // format as mm/dd/yyyy
    node.value = (dateValue.getMonth() + 1) + "/" +
dateValue.getDate() + "/" + dateValue.getFullYear();
    return "";
} else {
    return errorMessage;
}
}
return "";
} 

function checkPhone(event ) {
    var node = getTarget( event );
    if( node ) {
        var result =  validatePhone( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validatePhone( node ) {
    var errorMessage = "\nPhone needs to be a phone
number with area code, \nsuch as (555) 123-4567.";
    if (node.value != ") {
        // replace all non-digit characters, then match
        string starting with 2-9 (US phone numbers can't start with
        0 or 1) followed by 2 digits, 3 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/\([2-9]\d{2})\(\d{3}\)\(\d{4}\)/)) {
            // use sub-matches captured above to format as
            (###) ###-####
            node.value = "(" + RegExp.$1 + ") " + RegExp.$2 +
            "+" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    } else {
        return errorMessage;
    }
}
return "\nYou must enter a phone number, \nsuch as
(555) 123-4567";
function checkSSN(event) {
    var node = getTarget( event );
    if( node ) {
        var result = validateSSN( node);
        showError(node, result);
    } // endif
    return stopPropagation( event);
}

function validateSSN(node) {
    var errorMessage = "Social Security Number needs to be a nine-digit number, e.g., 123-45-6789."
    if (node.value != "") {
        // replace all non-digit characters, then match 3 digits, 2 digits, 4 digits
        if (node.value.replace(/\D/g, "").match(/(^\d{3})(\d{2})(\d{4})/) ) {
            // use sub-matches captured above to format as ###-##-####
            node.value = RegExp.$1 + "-" + RegExp.$2 + "-" + RegExp.$3;
            return "";
        } else {
            return errorMessage;
        }
    }
    return "";
}

function showError(node, message) {
    var id = node.id + "_error";
    var node_error = document.getElementById( id );
    if( node_error )
        node_error.innerHTML = "In valid: " + message;
    if( message == "" ) {
        node.setAttribute('aria-invalid', 'false');
        node_error.style.display = "none";
    } else {
        node.setAttribute('aria-invalid', 'true');
        node_error.style.display = "inline";
    }
}

function getTarget( event ) {
    var e = event || window.event;
    if( e.target )
        return e.target;
    else
return e.srcElement;
}  

function stopPropagation( event ) {
    if( event.stopPropagation )
        event.stopPropagation();
    if( event.preventDefault )
        event.preventDefault();
    if( event.cancelBubble )
        event.cancelBubble = true;
    if( event.returnValue )
        event.returnValue = false;
    return false;
}

// --></script>
</head>
<body>
<h2>Navigation Menu</h2>
<ul class="nav-list-primary" role=navigation>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site1'">Site 1</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site17'>Site 17</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site20'>Site 20</a></li>
    <li class="nav-list-item"><a href='https://csel.cs.colorado.edu/~erdu8260/site22'>Site 22</a></li>
</ul>
<div class="example">
    <h3>404 and Redirects</h3>
    <p><a href='http://www-ia.hiof.no/~linettev/html4U/form.htm'>I'm a 404 Page!</a></p>
    <p><a href='http://cs.colorado.edu/' >CU Boulder CS Homepage</a></p>
    <p><a href='http://google.com/' >Google Search Page</a></p>
    <p><a href='http://translate.google.com'>Google Translate</a></p>
    <p><a href='http://cs.nmt.edu' >New Mexico Tech CS Homepage</a></p>
    <p><a href='http://99waysnotto.tumblr.com/welcome'>99 Ways Not To...</a></p>
<div>
<h2>Sample Form</h2>
<p>Fields marked with an <img src="images/required.png" class="required" alt="Required" /> are required.</p>
<form id="SampleForm" action="#" method="post">
<div class="text">
<label for="fname">
  First Name
  <img src="images/required.png" alt=""/>
</label>
<input type="text"
  name="fname"
  id="fname"
  size="12"
  onblur="checkFirst(event)"
  aria-required="true"
  aria-invalid="false"
  aria-describedby="fname_error"/>
<div class="error" id="fname_error"></div>
</div>
<div class="text">
<label for="lname">
  Last Name
  <img src="images/required.png" alt=""/>
</label>
<input type="text"
  name="lname"
  id="lname"
  size="20"
  onblur="checkLast(event)"
  aria-required="true"
  aria-invalid="false"/>
<div class="error" id="lname_error" role="alert"></div>
</div>
<div class="text">
<label for="date">
  Birth Date
</label>
<input type="text"
  name="date"
  id="date"
  size="12"
  onblur="checkDate(event)"
  aria-required="false"
  aria-invalid="false"
  aria-describedby="date_error"/>
</div>
</form>
</div>
<form>
  <div class="error" id="date_error" role="alert"></div>
  <div class="text">
    <label for="ssn"><abbr title="Social Security Number">SSN</abbr></label>
    <input type="text"
      name="ssn"
      id="ssn"
      size="10"
      onblur="checkSSN(event)"
      aria-required="false"
      aria-invalid="false"
      aria-describedby="ssn_error"/>
    <div class="error" id="ssn_error" role="alert"></div>
  </div>
  <div class="text">
    <label for="phone">Phone</label>
    <input type="text"
      name="phone"
      id="phone"
      size="14"
      onblur="checkPhone(event)"
      aria-required="true"
      aria-invalid="false"/>
    <div class="error" id="phone_error" ></div>
  </div>
  <div class="button">
    <input type="button"
      value="Check Form"
      onclick=" validateForm()"/>
  </div>
</form>
<p><a href="http://www.msfw.com/accessibility/tests/ClientSideValidationTest.HTML">Example was developed from MSF&amp;W Accessibility Tests</a></p>
Results

Running ./tests/test23.js
Running all tests...
  Checking for image alt text
  images/required.png
  images/required.png
  Number of images = 4
  Number missing alt text = 2
  All images have alt text: false

Checking for image alt text with conditions
  images/required.png
  Required
  images/required.png
  images/required.png
  images/required.png
  images/required.png
  Required
  Number of images = 4
  Number missing alt text = 2
  Number failing test = 2
  All images meet alt text condition: false

Checking for image alt text consistency
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Missing: images/required.png
  Number of images = 4
  Number of images missing alt text = 8
  Number images inconsistent = 0
  All images are consistent: false

Checking for image alt text with word limits
  images/required.png
  Required
  images/required.png
  images/required.png
  images/required.png
  images/required.png
  images/required.png
  images/required.png
  images/required.png
images/required.png
Required
Number of images = 4
Number missing alt text = 2
Number not within word limit = 4
All images meet alt text word limit: false

Checking for page language...
Language not found on https://csel.cs.colorado.edu/~erdu8260/site23

Checking for some section languages...
Language not found for any sections on https://csel.cs.colorado.edu/~erdu8260/site23

Checking for lang on all <p> tags...
Number of <p>: 8
Number of <p> missing lang: 8
All <p> have lang attribute: false

Checking for skip to main content link...
Skip to main link found: false

Checking for 404 pages: https://csel.cs.colorado.edu/~erdu8260/site23
404: http://www-ia.hiof.no/~linettev/html4U/form.htm
No 404 pages discovered: false

Checking for [0,4] redirect pages...
Number of redirects found: 5
Number of redirects within [0,4]: false

Checking for bad certificates...
[CERT_UNTRUSTED]:https://csel.cs.colorado.edu/~erdu8260/site23
No bad certificates discovered: false

Checking for nav menu consistency...
https://csel.cs.colorado.edu/~erdu8260/site1
https://csel.cs.colorado.edu/~erdu8260/site17
https://csel.cs.colorado.edu/~erdu8260/site20
https://csel.cs.colorado.edu/~erdu8260/site22
Sites checked: 5
Sites inconsistent: 4
Consistent navigation menu: false

Checking for form ARIA compliance...
Tests complete.
Error class tags missing "role: 'alert'":

id = fname_error
id = phone_error
All form input ARIA compliant: false

Tests complete.

Site A

Results

Running ./tests/testA.js
Running all tests...
Checking for image alt text /
/profiles/cu_homepage/themes/cu_960_responsive/images/print-logo.png
Number of images = 7
Number missing alt text = 1
All images have alt text: false

Checking for image alt text with conditions /
/profiles/cu_homepage/themes/cu_960_responsive/images/print-logo.png
CU ski team at the NCAA Championships
Publicity still of CU Opera production of Cosi Fan Tutte
http://www.colorado.edu/sites/default/files/styles/square_thumbnail/public/promos/x355_LR.jpg?itok=Itw9AN6F
Old Main with spring blossoms
Number of images = 7
Number missing alt text = 1
Number failing test = 3
All images meet alt text condition: false

Checking for image alt text consistency
Missing: /profiles/cu_homepage/themes/cu_960_responsive/images/print-logo.png
  Missing:
http://www.colorado.edu/sites/default/files/styles/square_thumbnail/public/promos/x355_LR.jpg?itok=Itw9AN6F
  Number of images = 7
  Number of images missing alt text = 7
  Number images inconsistent = 0
  All images are consistent: false

Checking for image alt text with word limits
/profiles/cu_homepage/themes/cu_960_responsive/images/print-logo.png
/undefined
CU ski team at the NCAA Championships
Publicity still of CU Opera production of Cosi Fan Tutte
NASA graphic rendition of MMS LASP satellites.
Portrait of CU-Boulder Asian Studies student Leslie Dong.
http://www.colorado.edu/sites/default/files/styles/large-feature-image/public/features/feature/matusik_banner.jpg?itok=ETQPB00i
Photo of CU-Boulder business professor Sharon Matusik.

http://www.colorado.edu/sites/default/files/styles/square_thumbnail/public/promos/x355_LR.jpg?itok=Itw9AN6F
Old Main with spring blossoms
Number of images = 7
Number missing alt text = 1
Number not within word limit = 7
All images meet alt text word limit: false

Checking for page language...
Language not found on http://www.colorado.edu

Checking for some section languages...
Language not found for any sections on http://www.colorado.edu

Checking for lang on all <p> tags...
Number of <p>: 13
Number of <p> missing lang: 13
All <p> have lang attribute: false

Checking for skip to main content link...
Skip to main link found: true

Checking for 404 pages: http://www.colorado.edu
No 404 pages discovered: true

Checking for [0,4] redirect pages...
Number of redirects found: 70
Number of redirects within [0,4]: false

Checking for bad certificates...
No bad certificates discovered: true

Checking for nav menu consistency...
http://www.colorado.edu/
http://www.colorado.edu/#main-content
http://www.colorado.edu/atoz
http://www.colorado.edu/campusmap
http://www.colorado.edu/#
http://www.colorado.edu/audience/students
http://www.colorado.edu/audience/faculty-staff
http://www.colorado.edu/parents
http://www.colorado.edu/alumni
http://www.colorado.edu/about
http://www.colorado.edu/admissions
http://www.colorado.edu/academics
http://www.colorado.edu/research
http://www.colorado.edu/outreach
http://www.colorado.edu/international
http://www.colorado.edu/sustainability
http://www.colorado.edu/athletics
http://www.colorado.edu/news
http://www.colorado.edu/news/features/mms-spacecraft-blasts-students-ready-to-take-controls
http://www.colorado.edu/news/features/students-vision-becomes-global-student-initiative
http://www.colorado.edu/news/features/faculty-focus-no-10-business-strategist
http://www.colorado.edu/news/releases/2015/03/18/cu-boulders-conference-world-affairs-now-accessible-anywhere
http://www.colorado.edu/news/releases/2015/03/18/cu-boulder-study-beetles-beat-out-extinction
http://www.colorado.edu/news/releases/2015/03/17/cu-boulders-patty-limerick-review-nearly-40-years-university-fool-april-1
http://www.colorado.edu/events
http://www.colorado.edu/home?qt-main=0#qt-main
http://www.colorado.edu/home?qt-main=1#qt-main
http://www.colorado.edu/home?qt-main=2#qt-main
http://www.colorado.edu/about/visiting-campus/directions
http://www.colorado.edu/about/visiting-campus/parking-transportation
http://www.colorado.edu/campusmap/
http://www.colorado.edu/about/visiting-campus/planning-your-visit
http://www.colorado.edu/cwa/
http://www.colorado.edu/atlas/newatlas/events/
http://www.colorado.edu/atoz/letter/a
http://www.colorado.edu/atoz/letter/b
http://www.colorado.edu/atoz/letter/c
http://www.colorado.edu/atoz/letter/d
http://www.colorado.edu/atoz/letter/e
http://www.colorado.edu/atoz/letter/f
http://www.colorado.edu/atoz/letter/g
http://www.colorado.edu/atoz/letter/h
http://www.colorado.edu/atoz/letter/i
http://www.colorado.edu/atoz/letter/j
http://www.colorado.edu/atoz/letter/k
http://www.colorado.edu/atoz/letter/l
http://www.colorado.edu/atoz/letter/m
http://www.colorado.edu/atoz/letter/n
http://www.colorado.edu/atoz/letter/o
http://www.colorado.edu/atoz/letter/p
http://www.colorado.edu/atoz/letter/q
http://www.colorado.edu/atoz/letter/r
http://www.colorado.edu/atoz/letter/s
http://www.colorado.edu/atoz/letter/t
http://www.colorado.edu/atoz/letter/u
http://www.colorado.edu/atoz/letter/v
http://www.colorado.edu/atoz/letter/w
http://www.colorado.edu/atoz/letter/x
http://www.colorado.edu/atoz/letter/y
http://www.colorado.edu/atoz/letter/z
http://www.colorado.edu/atoz/show/u3a7d872abe219de2dfd5607a6bd9936a
http://www.colorado.edu/atoz/show/u05b57443f3f8831ad2d52c54b0a946b4
http://www.colorado.edu/atoz/show/u33ca8da233ca6fc7d7542cc170d5a48d
http://www.colorado.edu/atoz/show/u1871ffe25a842f5770d93eb62ed8955
http://www.colorado.edu/atoz/show/u71eece41407691596fd88220aedeca5d
http://www.colorado.edu/atoz/show/u8fede7ba4913c8c14187e6ea10888dd4
Test B

Results

Running ./tests/testB.js
Running all tests...
  Checking for image alt text
  Number of images = 2
  Number missing alt text = 0
  All images have alt text: true

Checking for image alt text with conditions
  Front of Cramer Hall, viewed from the parking lot off Bullock Blvd
   Accredited-CAC-Web
  Number of images = 2
  Number missing alt text = 0
  Number failing test = 2
  All images meet alt text condition: false

Checking for image alt text consistency
  Number of images = 2
  Number of images missing alt text = 0
  Number images inconsistent = 0
  All images are consistent: true

Checking for image alt text with word limits
  Front of Cramer Hall, viewed from the parking lot off Bullock Blvd
   Accredited-CAC-Web
  Number of images = 2
  Number missing alt text = 0
  Number not within word limit = 2
  All images meet alt text word limit: false

Checking for page language...
Language found on https://cs.nmt.edu: en-US

Checking for some section languages...
Language not found for any sections on https://cs.nmt.edu

Checking for lang on all <p> tags...
Number of <p>: 11
Number of <p> missing lang: 11
All <p> have lang attribute: false

Checking for skip to main content link...
Skip to main link found: false

Checking for 404 pages: https://cs.nmt.edu
No 404 pages discovered: true

Checking for [0,4] redirect pages...
Number of redirects found: 8
Number of redirects within [0,4]: false

Checking for bad certificates...
No bad certificates discovered: true

Checking for nav menu consistency...
https://cs.nmt.edu/
https://cs.nmt.edu/academics/
https://cs.nmt.edu/academics/degree-programs/
https://cs.nmt.edu/academics/degree-programs/cs-phd/
https://cs.nmt.edu/academics/degree-programs/cs-masters/
https://cs.nmt.edu/academics/degree-programs/bs-in-cs/
https://cs.nmt.edu/academics/degree-programs/bs-in-it/
https://cs.nmt.edu/academics/course-catalog/
https://cs.nmt.edu/academics/course-catalog/cse/
https://cs.nmt.edu/academics/course-catalog/it/
https://cs.nmt.edu/academics/class-homepages/
https://cs.nmt.edu/academics/special-topics-offerings/
https://cs.nmt.edu/academics/special-topics-offerings/spring-2014-special-topics-offerings/
https://cs.nmt.edu/academics/special-topics-offerings/special-topics-offerings-spring-2013/
https://cs.nmt.edu/academics/special-topics-offerings/special-topics-offerings-fall-2012/
https://cs.nmt.edu/academics/special-topics-offerings/spring-2012-special-topics-offerings/
https://cs.nmt.edu/academics/special-topics-offerings/spring-2011-special-topics-offerings/
https://cs.nmt.edu/academics/special-topics-offerings/fall-2011-special-topics-offerings/
https://cs.nmt.edu/academics/special-topics-offerings/fall-2010-special-topics-offerings/
https://cs.nmt.edu/people/
https://cs.nmt.edu/people/faculty/
https://cs.nmt.edu/people/staff/
https://cs.nmt.edu/people/graduate-students/
Number missing alt text = 30
All images have alt text: false

Checking for image alt text with conditions
http://i.cdn.turner.com/cnn/e/img/4.0/logos/wonderlist_bw.png
The Wonder List

Number of images = 83
Number missing alt text = 30
Number failing test = 23
All images meet alt text condition: false

Checking for image alt text consistency

Number of images = 83
Number of images missing alt text = 30
Number images inconsistent = 418
All images are consistent: false

Checking for image alt text with word limits

Number of images = 83
Number missing alt text = 30
Number not within word limit = 70
All images meet alt text word limit: false

Checking for page language...
Language not found on http://www.cnn.com

Checking for some section languages...
Language not found for any sections on http://www.cnn.com

Checking for lang on all <p> tags...
Number of <p>: 2
Number of <p> missing lang: 2
All <p> have lang attribute: false

Checking for skip to main content link...
Skip to main link found: false

Checking for 404 pages: http://www.cnn.com
404: http://offers.lendingtree.com/splitter/splitter.ashx?id=RefisavingsJDT1
No 404 pages discovered: false

Checking for [0,4] redirect pages...
Number of redirects found: 23
Number of redirects within [0,4]: false

Checking for bad certificates...
No bad certificates discovered: true

Checking for nav menu consistency...
Sites checked: 134
Sites inconsistent: 0
Consistent navigation menu: true

Checking for form ARIA compliance...
Tests complete.
Input tags missing aria-describedBy:
  id = edition-picker--edition-header
  id = edition-picker--www-header
  id = searchInputNav
  id = searchInputFooter
  id = edition-picker--edition-footer
  id = edition-picker--www-footer
All form input ARIA compliant: false

Tests complete.

Site D

Results

Running ./tests/testD.js
Running all tests...
Checking for image alt text
  /images/icons/product/chrome-48.png
  Number of images = 2
  Number missing alt text = 1
  All images have alt text: false

Checking for image alt text with conditions
  /images/icons/product/chrome-48.png
  /logos/doodles/2015/first-day-of-spring-2015solar-eclipse-5662379358552064-hp.gif
    1er jour du printemps et eclipse solaire
  Number of images = 2
  Number missing alt text = 1
Number failing test = 1
All images meet alt text condition: false

Checking for image alt text consistency
Missing: /images/icons/product/chrome-48.png

Number of images = 2
Number of images missing alt text = 2
Number images inconsistent = 0
All images are consistent: false

Checking for image alt text with word limits
/missing.png
/missing.png
Undefined
/missing.png

Number of images = 2
Number missing alt text = 1
Number not within word limit = 2
All images meet alt text word limit: false

Checking for page language...
Language found on https://www.google.fr: fr

Checking for some section languages...
Language not found for any sections on https://www.google.fr

Checking for lang on all <p> tags...
Number of <p>: 1
Number of <p> missing lang: 1
All <p> have lang attribute: false

Checking for skip to main content link...
Skip to main link found: false

Checking for 404 pages: https://www.google.fr
No 404 pages discovered: true

Checking for [0,4] redirect pages...
Number of redirects found: 18
Number of redirects within [0,4]: false

Checking for bad certificates...
No bad certificates discovered: true

Checking for nav menu consistency...
https://www.google.fr/
https://www.google.fr/webhp?tab=ww
https://www.google.fr/imghp?hl=fr&tab=wi
http://www.google.fr/intl/fr/options/
http://www.google.fr/shopping?hl=fr&tab=wf
http://www.google.fr/preferences?hl=fr
https://www.google.fr/preferences?hl=fr
http://www.google.fr/history/optout?hl=fr
https://www.google.fr/search?site=&q=%C3%89quinoxe+de+printemps+%26+printemps+%26+printemps+&oi=ddle&ct=first-day-of-spring-2015solar-eclipse-566227935852064-hp&hl=fr&sa=X&ei=y5AMVejUDZHroAToqYBQ&ved=0CAMQNg
https://www.google.fr/advanced_search?hl=fr&authuser=0
https://www.google.fr/language_tools?hl=fr&authuser=0
https://www.google.fr/setprefs?sig=0_wxxf_Gio6AGjjLHMKmu95V8TteU%3D&hl=en&source=homepage
https://www.google.fr/intl/fr/ads/
https://www.google.fr/services/
https://www.google.fr/intl/fr/about.HTML
https://www.google.fr/setprefdomain?prefdom=US&sig=0_BopB-xsb1LoDi8BjyKH5piLPht8%3D
https://www.google.fr/intl/fr/policies/privacy/
https://www.google.fr/intl/fr/policies/terms/
Sites checked: 19
Sites inconsistent: 19
Consistent navigation menu: false

Checking for form ARIA compliance...
Tests complete.
Input tags missing aria-describedby:
   id =
All form input ARIA compliant: false
Tests complete.