



Automated Testing of Web Applications Coursework

Author: Matthew Herman

Supervisor: Dr Martin Caminada

Moderator: Dr Bailin Deng

CM3203 – Individual Project

Academic Year: 2021/22

Word Count: 13,330

Abstract

One of the modules studied in the first year of Computer Science is the module Web Applications. For this one of the first assignments that students are to complete is a task where they have to build two sites. Firstly, students are to build a static site that must satisfy various requirements such as having a certain number of paragraphs, external CSS, and a navigation menu. The other site is a dynamic site that allows for the input of a year and a format, and the site should return the date that Easter is or was on the given year.

As Student numbers are rising, the University is interested in making the marking of these two tasks as automated as possible, to assist the marker so they won't have to look through every element of the site to make sure it meets the requirements. The focus of the report will be the creation of a tool that will assist markers by making the process more automated.

Acknowledgements

I would like to thank my supervisor Dr Martin Caminada for the opportunity to undertake the completion of this project, and for their support and feedback throughout its development. As well as this I would like to thank my family and friends who helped and supported me throughout the completion, as well as kept me motivated to complete it

Table of Contents

Abstract.....	1
Acknowledgements	1
Table of Figures	4
Introduction	5
Project Aims	5
Background	5
Current System	5
Previous Solutions	7
Proposed Solution	7
Stakeholders.....	7
Approach and Methodology	8
Software Development Approaches.....	8
Tools and Programming Languages	8
Specification & Design	10
Functional Requirements	10
Non – Functional Requirements	12
Use Cases.....	13
Implementation.....	15
Tool File Structure.....	15
Part 1 Testing – Static Site.....	16
Main Program.....	16
Part 1 Tests.....	18
Part 2 Test - Dynamic Site	22
Main Program.....	22
Part 2 Tests.....	23
Common Tests	26
Reading from Console and Config File	28
Results and Evaluation	29
Testing Strategy	29
Testing Full System with Previous Submissions – Site 1	29
Testing Full System with Previous Submissions – Site 2	33
Testing based on Requirements	36
Testing of Non-Functional Requirements	43
Future Work	44
Conclusion	45

<i>Reflection</i>	<i>46</i>
<i>References.....</i>	<i>47</i>

Table of Figures

Figure 1 - File structure	15
Figure 2 - main program site 1	16
Figure 3 - runCheck function	17
Figure 4 - returnResults function.....	18
Figure 5 - initialSiteHtmlPages function	18
Figure 6 - checkForTable function	19
Figure 7 - checkNavMenu function	20
Figure 8 - checkClickPicture function	20
Figure 9 - checkThreeParagraphs function.....	21
Figure 10 - main program site 2	22
Figure 11 - outputOptions dictionary	23
Figure 12 - getEasterResponses.....	23
Figure 13 - getEasterOutput function.....	24
Figure 14 - getCorrectEasterOut function	25
Figure 15 - checkOutput function.....	26
Figure 16 - checkURL function.....	26
Figure 17 - getParsedHTML function.....	27
Figure 18 - checkLastModified function	27
Figure 19 – checkValidHTML function.....	28
Figure 20 - checForCSS function	28
Figure 21 - config.json file	28

Introduction

All first-year students at the School of Computer Science at Cardiff University are required to complete the Web Applications Module, which takes them through various internet and web technologies used around the world, these include things such as Front – End development, like HTML, CSS, and JavaScript as well as Backend and Server frameworks, such as Linux Webservers as well as Django. As part of the assessment for this module, students are required to complete a coursework assessment which involves building two different sites. One is a static site which should be about an aspect of technology and meet various requirements such as having a navigation menu, having a certain number of pages, and containing an external CSS document. The second site should take a year as an input and dynamically show the date that Easter is on that year, based on the year inputted into the form.

Currently, the project is marked by manually opening the website in a web browser and checking that the website meets each of the requirements. As the coursework has several requirements for each site, this can take some time, especially as for some this involves going into the inspector and looking at the code as well as doing the front-end checks. With the rising number of students starting at university and a possible need to add more requirements, this process is becoming very awkward and time-consuming. As a result, there is a need to move this to a more automated process. The goal of my project is to build a tool for a marker to use to test many aspects of the coursework automatically without the need to go through the web browser.

Project Aims

The project aims are to create a tool to assist the marker, this tool would likely be a console / terminal application that can easily be run and will take input which will be the URL of each student's website and return results on whether the given requirements have been met or not. It should provide feedback on what has been passed or not passed and in certain cases what has caused the test to fail. The tool also may be given to students, so they can use it to be able to test their code to check it passes all tests before submitting their work. As the tool will most likely be run in a terminal window, there will not have to be a consideration for user Interface, however, the tool should be easy enough and at least well laid out to be able to use and figure out without much training or work.

Background

Current System

Currently, both websites for the students are marked by the marker, the process for marking is currently about 5-10 minutes for each student and requires the students to be

either in a call or in-person with the marker. For the first site (the static site about an aspect of technology), the required processes for each part of the marking criteria are:

1. "At Least 3 pages of HTML Code" – The marker must check the site manually to see if there are at least 3 pages. It also needs to check each page for any errors using the source code viewer (Control + U).
2. "Navigation with a menu on each page that provides access to each other page" – The marker should check every page is accessible from the menu, as well as if the menu is visible on each page.
3. "A table that has at least two columns and three rows, containing information relevant to the website" – Again the marker should check each page to see if there is a table, making sure there are 3 rows and 2 columns.
4. An external style sheet that includes rules that at least set the font family, size, and colour of text" – Again the marker needs to open the inspector in the browser to get the HTML source, find the CSS most likely in the header, open the CSS and check the font family, size, and colour selectors.
5. "Informative content about a key person or a technology, including a clickable picture" – Again the marker should check each page to ascertain that there is enough informative content about the aspect as well as check each page if there is a clickable picture, to be able to get the full marks for this section.

For the second site (dynamic site to find easter) the required processes are:

1. "An HTML page with a form that accepts user input of both the year and the option" – The marker should check the form to make sure the form contains an input for the year as well as check the form contains radio buttons for options (being verbose, numeric and both), as well as make sure the radio buttons are not broken.
2. "The above page is styled with an external CSS style sheet for formatting the output" – The marker should view the source using the inspector, check there is a link to external CSS and then check if at least one property is set by the given CSS document.
3. "Python + CGI program at least echoes (prints out) the year that the user has input into the form, as an HTML/CSS web page" – This involves the marker checking whether the form works, the user should at least check if the output echoes the year, as well as check if it uses an external CSS stylesheet
4. "Python + CGI processing of user input to generate and output (in HTML/CSS) the date of Easter for the year that the user has entered" – The marker has to check whether the script gets the date of easter. This has to be checked for multiple dates and make sure the correct date is given for each date.
5. "Correct implementation of the user options for formatting the date as both 'dd/mm/year' and 'day named_month year'" – The marker has to check the output is the correct output based on the input on the form, this only has to be for one date for each output

6. "The superscript for the day (st, nd, rd, or th) is correctly computed and displayed" – The marker should check if the correct subscript has been used, and used correctly based on the date, for the verbose output.

Previous Solutions

A previously implemented system for this task was using a web driver. This is an interface that interacts with a web browser and can control it in the same way that a user would. This system used a web driver to interact with the student's site to be able to get the outputs. The issue with web drivers, however, is that they have dependencies on certain web browsers and depend on the setup that the user of the tool has on their specific computer. Also, if the browser setup changes this may be enough to cause the tool to not work as the dependencies are no longer there. This was the case with this solution; the tool ceased to work as it relied on dependencies in the Firefox Web Browser which didn't exist.

Proposed Solution

As seen above in the current system is currently the process that is being used to mark the coursework, as well as a solution that has been tried before. This in combination with the fact that it must be combined with the marker being in the room with the student creates a very complicated way of marking the coursework. With a tool that could automatically check most of these elements, this process would be easier. The proposed solution of the project would be a tool that would take in a URL of the student website and run the checks on it automatically, instead of the marker having to check each criterion manually. The proposed solution would be able to take each URL of the student, check the site against the marking criteria and return the results of whether each criterion is met or not

Stakeholders

The stakeholders for this project include several groups of people. The main group of people who will benefit from an automated marking tool are the markers of the coursework, as this group of people will be able to mark the coursework more efficiently, and as the number of students being marked is large, the tool will allow them to get through the process faster. Another stakeholder will be the module leader, as this again will enable the marking process to be faster whilst allowing module leader to release marks quicker along with making the process of marking students a lot smoother. Finally, the last group of stakeholders will be the students themselves as the tool has the potential to be released to the students as the marking team. The tool being released to the students would allow them to test their work with the tool. This would mean that they could see if their sites meet the marking criteria and if all the elements are working correctly. This would mean students are more likely to get better marks as they have a way of checking if their work is correct before they submit their code.

Approach and Methodology

Software Development Approaches

Waterfall Method

This method is generally considered to be the traditional software development method. It involves a model that has linear steps that each have distinct goals, for example – requirements, design, implementation, verification, and maintenance. Each step is sequential and leads to the next. Before moving on to the next step the current step must be fully completed. There is usually no way to go back on each step if the whole step must be undone first before being allowed to go backwards. The fact that each step goes into each other makes this very easy to understand and manage, however, it is generally quite slow as well as inflexible if any changes need to be made when through the development cycle.

Agile Methods

There are many agile methods to decide between when developing software. They mainly consist of various steps; however, they are iterated over again and again. This allows the user to receive feedback from their clients and make any changes to the code before iterating the process again with the new changes. This is a good method to use as it allows any issues, fixes, and changes to be sorted constantly and not just to be found out at the end and then have to change the whole system then. However, when using this there needs to be regular communication between the developer and the customer. It also can take more time if there are always changes to be made, as there would need to be a new iteration every time that this happens.

My Chosen Solution

My solution to develop the tool was to use an Agile methodology. As the tool would require constant improvement and would need to be viewed by the client as they know the marking criteria as well as what was needed best, it would be best to be able to use a method that can make use of feedback given from the client (in this case my supervisor). As it is only me that would be developing this tool, I would need a method that would allow me to test and check for issues during the development, as the waterfall method only allows for this at the end of the development process, I know that would also not be ideal.

Tools and Programming Languages

Python

I will be writing this tool using the programming language Python. I have decided to use Python as it contains the libraries and modules that can capture various webpages and parse and analyse the HTML and CSS so that I can check the HTML against the marking criteria that it is met. Python code is also able to be run on all operating systems and on

most devices, that have a desktop operating system, which would be needed as markers will most likely be using a variety of operating systems and devices. This is unlike some other programming languages that do not have the support for some of these libraries or some of them are not compatible on all platforms, so therefore I have decided to use Python.

HTTP Requests (Python Requests Library)

I have decided to use HTTP requests to capture any web pages. This is opposed to the previous use of a Web Driver to do this. Using a web driver as discussed above would depend on the markers/students' web browser setup and could cause future issues if the web browser were to change. As the web pages can be directly accessed without having to use a university login screen when on the university network, there is no need to have to use a web driver to get past this section, therefore it is better to use direct HTTP requests as it will always get the HTML, as it is just requesting from the server and does not depend on the setup that the marker has.

To perform HTTP requests, I have decided to use the Python requests module. This module allows one to send HTTP requests in Python. Once sent it will return the results of the HTTP request. The module allows for various requests including POST, GET, DELETE, and PUT, however for this I will mainly be using POST and GET. I have chosen to use this module instead of the urllib module on which the requests module is based as the requests module is easier to understand and does not require as much code to perform the same action, which means less code would be needed. Also, I am only using the requests part of the requests library and would not need any of the features of urllib.

Beautiful Soup

To check the HTML of a students site, I will need to be able to parse the HTML from the HTTP requests. Beautiful Soup is a python library that parses any HTML and organises it into its relevant tags and makes it easily traversable. This would make my code better as without this I would have to use string matching, which would not be as efficient or as ideal as it may not work as intended. This tool also allows me to search any given HTML for specific tags within. This is great for the tool as I will have to check for specific tags to check the marking criteria, such as img, a, and table.

Visual Studio Code

This is a graphical code editor that will be used to create and edit the code for this project. I have used this as it supports Python and its libraries quite well. It is quite helpful as it highlights any errors and can show developers what needs to be changed based on the error in the code. It also supports auto-filling commands, which makes developing code much easier. This is better than using a text-based editor, or IDLE as these functions are not available and thus would make this harder to develop.

Specification & Design

Functional Requirements

1. The tool should allow for user input in the console for the URL of the site to be inputted
Acceptance Criteria:
 - The user will run the script from the console with the URL as an argument
 - The code will run with that site as the one being checked
2. The tool should allow users the option to opt for a verbose mode which shows more options
Acceptance Criteria:
 - The user will be able to add a verbose argument after the URL when running the script to add verbose mode when running.
 - When added the verbose mode will show a more detailed output than when just running the script without
3. The tool should check if the URL is valid and actually returns an HTTP response
Acceptance Criteria:
 - The tool should run if a valid URL is given.
 - If no URL is given or if the URL does not return anything an error should be shown
4. Site 1 - The tool should check Marking Criterion 1 – Whether there are at least 3 pages of HTML code.
Acceptance Criteria:
 - The tool should show the marker whether there are or are at least 3 pages and whether the test has been passed, and failed if not
 - In verbose mode, the marker should be shown whether the test has been passed as well as what URL's have been seen.
5. Site 1 - The tool should check Marking Criterion 2 – Whether there is a Navigation menu that points to each page of the site
Acceptance Criteria:
 - The tool should show the marker whether there is a navigation menu on the site and whether the test has been passed, and failed if not
 - In Verbose mode, the marker should be shown whether the test has been passed or not.
6. Site 1 - The tool should check Marking Criterion 3 – Whether there is a table and whether the table has at least two columns and three rows

Acceptance Criteria:

- The tool should show whether it has found a table and whether the test has been passed or whether it has failed.
- When in verbose mode the tool should show whether the table has been found, and on what page the table has been found.

7. Site 1 - The tool should check Marking Criterion 4 – Whether the site contains an external CSS that sets the font family, font size and colour.

Acceptance Criteria:

- The tool should show whether the site contains the external CSS sheet and whether the test has passed or failed
- The verbose should show whether the site contains the external CSS and should show a link to the CSS document

8. The site should read the submission date from a config file

Acceptance Criteria:

- The tool should check the site against the submission date given in the config file

9. The site should read the chosen test years from a config file

Acceptance Criteria:

- The tool should check the easter dates with the years given in the test file

10. Site 2 – The tool should check Marking Criterion 1 – Whether the site contains a form that accepts both the year as well as the format of the output.

Acceptance Criteria:

- The tool should show whether the form and format inputs exist on the site and whether the test has passed or failed

11. Site 2 – The tool should check Marking Criterion 2 – Whether the site contains an external CSS stylesheet.

Acceptance Criteria:

- The tool should show whether the site contains an external CSS stylesheet
- In the verbose mode, the tool should show whether the CSS stylesheet has been found as well as a link to the CSS stylesheet

12. Site 2 – The tool should check Marking Criterion 3 – Whether the site at least echoes the year, that has been inputted into the form

Acceptance Criteria:

- The tool should check whether the year given is outputted in the site output and should show how many years passed
- The tool should check this for all years in the config file

13. Site 2 – The tool should check Marking Criterion 4 – Whether the site has correctly processed the date for Easter of the given year.

Acceptance Criteria:

- The tool should check whether the correct date has been processed and should show how many years this has passed.
- The tool should check this for all years in the config file

14. Site 2 – The tool should check Marking Criterion 5 - Whether the site has correctly outputted for both format options, this being dd/mm/year, and day named month year.

Acceptance Criteria:

- The tool should check whether the output is correct for all options and should show how many years passed this has passed.
- The tool should check this for all years in the config file

15. Site 2 – The tool should check Marking criterion 6 – Whether the site has correctly outputted the correct superscript for the date.

Acceptance Criteria:

- The tool should check if the correct superscript has been used and should show how many years this has passed.
- The tool should check this for all years in the config file

16. The tool should check the Modification date, to make sure that the site has not been modified after the submission date.

Acceptance Criteria:

- The tool should check if the modification is not after the given submission date.
- The verbose mode should show the last modified date and if this is after the submission date or not

17. The tool should check the site for any errors as well, and present them to the user

Acceptance Criteria:

- The tool should check if there are any errors on the site and show them to the user when using the verbose mode.

Non – Functional Requirements

1. When checking the sites, the tool should not have any false positives

Acceptance Criteria:

- The tool should be checked against various sites and should always show a failed test if does not meet the marking criteria.
- The tool can show a failed test if it is actually passed, as this can always be looked into further

2. The tool should be able to check the sites and load data in a reasonable amount of time

Acceptance Criteria:

- The tool should not take longer than a few seconds to perform testing on each site in the case of a reasonable internet connection.

3. The interface while being a console app, should be relatively easy for the user to understand what is going on

Acceptance Criteria:

- All users should be able to use the tool once shown how to use it and shouldn't have to struggle with understanding the output of the tool.

Use Cases

Use Case 1 – Test Site 1 with Verbose

Flow:

- The user opens the command line in the folder for the tool
- The user runs the command for site 1 with the URL for site 1 and the Verbose mode flag, as arguments
- The user is presented with the verbose output, for each marking criterion
- The user is presented with the results for each criterion saying whether each test has been passed or failed

Use Case 2 – Test Site 1 without Verbose

Flow:

- The user opens the command line in the folder for the tool
- The user runs the command for site 1 with the URL for site 1, as an argument
- The user is presented with the results for each criterion saying whether each test has been passed or failed

Use Case 3 – Test Site 2 with Verbose

Flow:

- The user opens the command line in the folder for the tool
- The user runs the command for site 2 with the URL for site 2 and the Verbose mode flag as arguments
- The user is presented with the verbose output, for each marking criterion
- The user is presented with the results for each criterion saying whether each test has been passed or failed

User Case 4 – Test Site 3 without Verbose

Flow:

- The user opens the command line in the folder for the tool
- The user runs the command for site 2 with the URL for site 2 as

- The user is presented with the results for each criterion saying whether each test has been passed or failed

Implementation

Tool File Structure

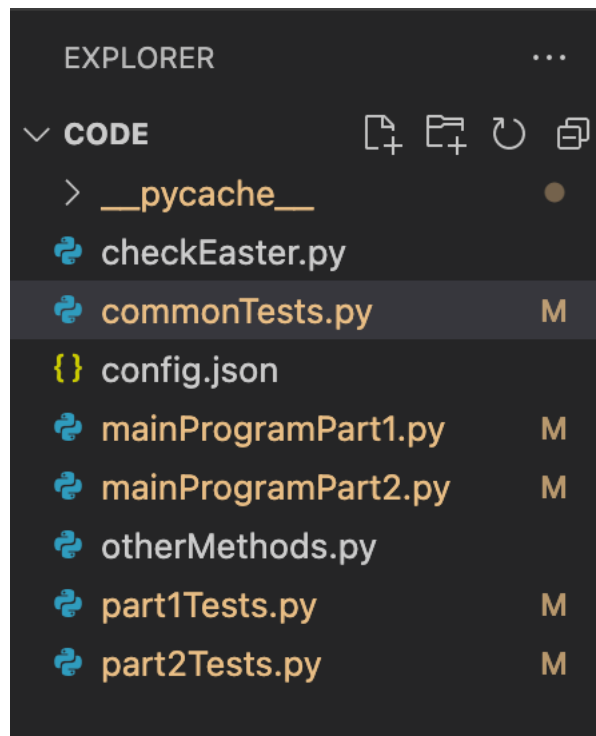


Figure 1 - File structure

This is the file structure for the tool. The “mainProgramPart1.py” contains all the functions that involve the outputs for verbose mode and normal output for the testing of site 1. This file will get the input from the command line as well as from the config file, as well as run the functions that will test the site. At the end of this file, the results of the tests will be shown to the user. Similarly, the file “mainProgramPart2.py” performs the same tasks for site 2.

The files “part1Tests.py” and “part2Tests.py” contain all the functions that perform the tests for site 1 and site 2 respectively. These are all imported and run in mainProgramPart1 and mainProgramPart2 when they are needed. I have the tests in a separate file as it means the main part can be kept shorter. Also, a lot of tests are used more than once in the main program so calling them from another file. Similarly, “commonTests.py” contain any tests that are common to both sites, such as testing for submission dates and any HTML errors. This is again so they don’t have to be in both files for site 1 and the site 2 tests and can be reused in each site, which makes the code more concise again, and prevents unnecessary duplicate code.

The file “otherMethods.py” contain any code that is not related directly to testing of the site, but the code needs to run, and “checkEaster.py” contains the methods used to calculate the actual date of Easter so the easter site can be checked against this.

Finally, the “config.json” file contains the config data for testing. This is in a JSON format and contains any config data, such as the list of years for Easter to be tested as well as the Submission date, this will be able to be edited by the user to be changed for each marking session.

Part 1 Testing – Static Site

Main Program

```
def main():
    getVerb = ""
    verbMode = False

    if len(sys.argv) > 2:
        siteURL = sys.argv[1]
        getVerb = sys.argv[2]
        if getVerb == "-v":
            verbMode = True
            urlCheck = (checkURL(siteURL))
            if urlCheck[0] != True:
                print(cs("Error Getting URL \n" + str(urlCheck[1]), "Red"))
                exit()
        elif len(sys.argv) > 1:
            siteURL = sys.argv[1]
            urlCheck = (checkURL(siteURL))
            if urlCheck[0] != True:
                print(cs("Error getting URL \n" + str(urlCheck[1]), "Red"))
                exit()
        else:
            print("No Site URL Provided, Please run again with site URL as a Variable")
            exit()

    if getVerb == "-v":
        print(cs("Running in Verbose Mode", "red"))
        verbMode = True

    if not verbMode:
        with HiddenPrints():
            checks = runChecks(siteURL)
            returnResults(checks)
    else:
        checks = runChecks(siteURL)
        returnResults(checks)
```

Figure 2 - main program site 1

The program for site 1 is run using the main function in the mainProgramPart1.py. It uses sys.argv to get any arguments from the command line. It first checks if the length is more than 2 to make sure there are at least 2 arguments when running. It then gets the site URL from the first argument and the verbose mode flag from the second argument. It then checks the URL to make sure that it returns a valid response, and that there are no connection errors when connecting to the URL. If there are any connection errors or if the URL doesn't exist, then the program will exit and return an error to the user. This is checked before the rest of the program, because if the URL is not correct then the rest of the tool cannot run at all as it would not have a valid URL to test hence this is done at the start. The verbose mode is also set here if the second argument exists and is equal to a “-v” then the

program will set the verbose mode on, again this is set at the start as any outputs that just relate to the verbose mode will need to be disabled if it is not enabled. If verbose mode is set to off, it will use the HiddenPrints function, which disables the print function inside the runChecks function. This means that any print statements inside the runChecks will not run, which means no outputs will be shown when running, if it is set to on then it will not use the function so no statement will be hidden. I chose to do it this way as it means I would not have to write the same code without the print statements which makes the code more concise and will take less time to run.

```
def runChecks(siteURL):
    listOfPages = getInitialSiteHtmlPages(siteURL)

    checks = {
        "ThreePage": False,
        "Table": [False, ""],
        "ClickablePicture": False,
        "ExternalCss": [False, False],
        "NavMenu": False,
        "AllSitesValid": False,
        "ThreeParagraphs": False,
        "ValidHTML": False
    }

    print (bold(underline("Finding Pages")))
    if len(listOfPages) >= 3:
        checks['ThreePage'] = [True, "\n".join(listOfPages)]
        print(bold(cs("Pages Found:", "green")))
        print(cs(checks["ThreePage"][1], "blue"))
    else:
        checks['ThreePage'] = [False, "\n".join(listOfPages)]
        print(bold(cs("Pages Found:", "red")))
        print(cs(checks["ThreePage"][1], "blue"))
```

Figure 3 - runCheck function

The runChecks function is the function that runs all the checks on the site. I have used a dictionary that will store whether each check has passed or failed or not. This is in the form of False if failed and True if passed. The table entry has both the Boolean variable and another variable which contains the link to the site where the table is found. The External CSS value also contains two variables, the Boolean value for whether the test has passed, and another Boolean for whether it is the same CSS document for the whole site. All these values are set to false initially to make sure that they are all false, and all tests fail unless they are found to have met the criteria. I have used a dictionary for all these values as it can easily be updated, for each value.

The program then runs all the test functions for each of the marking criteria. It starts with running the test for the first marking criteria, which is that there are at least 3 pages of HTML. This is the first test to run as it involves adding the found sites to a list which is saved in a variable. If this list is larger than 3 elements the dictionary value is marked as True. This

list is then used throughout the rest of the tests as they mostly involve checking all the pages. For the rest of the marking criteria, this being the table, Clickable picture, External CSS, Nav Menu, and the Submission date. Each of the sites in this list is iterated through to make sure each site meets the test. The submission date is received from the getSubmission function which loads the submission date from the JSON configuration file.

```
def returnResults(checks):
    resultOut1 = lambda x: f"The site should have at least 3 Pages of HTML\nPassed - Found sites listed below\n{x[1]}\n" if x[0] else "The site should have at least 3 Pages of HTML\nFailed\n"
    resultOut2 = lambda x: f"The site should contain a table with at least 2 columns and 3 rows\nPassed - Valid table found on following site:\n{x[1]}\n" if x[0] else "The site should contain a table with at least 2 columns and 3 rows\nFailed\n"
    resultOut3 = lambda x: f"The site should contain a clickable picture\nPassed - Valid picture found on following site:\n{x[1]}\n" if x[0] else "The site should contain a clickable picture\nFailed\n"
    resultOut4 = lambda x: "Each Page should contain a link to an external style sheet, with the rules that set at least the font family, size, and the colour of the text\nPassed\n" if x[0] else "Each Page should contain a link to an external style sheet, with the rules that set at least the font family, size, and the colour of the text\nFailed\n"
    resultOut4_5 = lambda x: "The Stylesheet should be the same for each page.\nPassed\n" if x[0] else "The Stylesheet should be the same for each page.\nFailed\n"
    resultOut5 = lambda x: "Each Page should have a navigation menu that can access each of the pages\nPassed\n" if x[0] else "Each Page should have a navigation menu that can access each of the pages\nFailed\n"
    resultOut6 = lambda x: "The site should not be modified after the submission date\nPassed\n" if x[0] else "The site should not be modified after the submission date\nFailed\n"
    resultOut7 = lambda x: "2 pages should have at least 3 Reasonably Sized Paragraphs\nPassed\n" if x[0] else "2 Sites should have at least 3 Reasonably Sized Paragraphs\nFailed\n"

    print("\n" + resultOut1(checks['ThreePage']))
    print(resultOut2(checks['Table']))
    print(resultOut3(checks['ClickablePicture']))
    print(resultOut4(checks['ExternalCss'][0]))
    print(resultOut4_5(checks['ExternalCss'][1]))
    print(resultOut5(checks['NavMenu']))
    print(resultOut6(checks['AllSitesValid']))
    print(resultOut7(checks['ThreeParagraphs']))
```

Figure 4 - returnResults function

Finally, the returnResults function is run, which takes the checks dictionary as a parameter. For each value in the dictionary, the user is shown a message for each marking criterion and other tests, and whether it has passed or not. This is based on the Boolean value of the dictionary value. I have also used the string colour library in Python which allows me to change the colour of the output, which allows the user to distinguish between the different types of output, such as a failed test in red, and a passed test in green.

Part 1 Tests

All the tests are stored in the file "Part1Tests.py", the individual functions for each test will be discussed below.

```
def getInitialSiteHtmlPages(linkToSite):
    listOfPages = []
    parsedHTML = getParsedHTML(linkToSite, "GET")
    for link in parsedHTML.find_all('a'):
        try:
            if link.get('href').startswith("http"):
                code = rq.get(link.get('href')).status_code
            else:
                code = rq.get("http://" + link.get('href')).status_code
            if code == 200:
                listOfPages.append(link.get('href'))
        except:
            if rq.get(rq.compat.urljoin(linkToSite, link.get('href'))).status_code == 200:
                listOfPages.append(rq.compat.urljoin(linkToSite, link.get('href')))
    for page in listOfPages:
        if page.__contains__("project.cs.cf.ac.uk") != True:
            listOfPages.remove(page)
    return list(dict.fromkeys(listOfPages))
```

Figure 5 - initialSiteHtmlPages function

The function getInitialHtmlPages takes a parameter of the link to the site. It starts by taking this link to the site and using a get request using the python request library and then using

the BeautifulSoup module to parse the HTML from the get request. Using the BeautifulSoup find_all function, which finds all of the 'a' tags in the HTML document. It then iterates through all the 'a' tags found and finds the href attribute which is the link for the given 'a' tag. It then checks performs a get request with the link of the given 'a' tag and checks if the status code is 200. This has been done as if the site returns a code of 200, then the site is valid and not broken. If all these passes, then the site is added to the list of pages list. As only these are added to the list there will only be links that return this status code, and no other links should be added making sure that all links are not broken.

After this, the list of pages is iterated over and removed if they do not contain the project server URL, as this would mean that no links are included that are not part of the student's project. Any duplicates are then removed from this list, and it is returned by the function.

```
def checkForTable(linkToPage):
    parsedHTML = getParsedHTML(linkToPage, "GET")
    list = parsedHTML.find_all('table')
    check = False
    if len(list) >= 1:
        for item in list:
            rowList = item.find_all('tr')
            if len(rowList) >= 3:
                for row in rowList:
                    thRow = row.find_all('th')
                    tdRow = row.find_all('td')
                    if len(thRow) >= 2 or len(tdRow) >= 2:
                        check = True
                    else:
                        check = False
            return check
    return check
```

Figure 6 - checkForTable function

The checkForTable function takes a parameter of the URL of the page that is being checked. Using a get request and the BeautifulSoup library to parse the HTML, it is then checked for any table tags, on the site. The check variable is initialised and set to false, as there is currently no table found with the required criteria. Each table in the list of table tags is iterated over, and checked for 'tr' tags, if this list is more than 3 then the row is checked for any 'th' or 'td' tags and if is more than 2. If both are true, then the check is set to True and is returned by the function if any of these are false then the next Table tag is iterated over. If none is found, then the function will return False as no table with 3 rows and 2 columns has been found.

```

def checkNavMenu(linkToPage):
    parsedHTML = getParsedHTML(linkToPage, "GET")
    count = 0
    ulTags = parsedHTML.find_all('ul')
    olTags = parsedHTML.find_all('ol')
    for ol in olTags:
        ulTags.append(ol)
    for ul in ulTags:
        liTags = ul.find_all('li')
        for li in liTags:
            aTag = li.find('a')
            if aTag != None:
                href = aTag.get('href')
                if href != "" or href != None:
                    count += 1
        if len(liTags) == count:
            return [True, ul]
    return [False, []]

```

Figure 7 - checkNavMenu function

The checkNavMenu function takes a link to the page as a parameter and then like the other functions, it parses the HTML from the get request using the link. It then checks the page to find any ol or ul tags. It then merges the list of ol and ul tags. Using the Beautiful soup library, it checks the list of tags for any li tags, which contain 'a' tags. It then increments a counter by 1 the 'li' tag contains an 'a' tag with a link. When this is done it checks if the counter is equal to the length of the li tags. If this is true, then the function returns True as the test is passed as well as the list of 'ul' or 'ol' tag, which is returned so that the main program can check against the other pages if the same nav menu is used. 'ol' and 'ul' tags are checked for as they are how a nav menu is constructed, and if they contain links it is likely to be a nav menu so if this is true then the test is passed.

```

def checkClickPicture(linkToPage):
    parsedHTML = getParsedHTML(linkToPage, "GET")
    aTags = parsedHTML.find_all('a')
    for a in aTags:
        imgTag = a.find_all('img')
        try:
            if a.get('href').startswith("http"):
                code = rq.get(a.get('href')).status_code
            else:
                code = rq.get("http://" + a.get('href')).status_code
            if code == 200:
                if len(imgTag) == 1:
                    return True
        except:
            if rq.get(rq.compat.urljoin(linkToPage, a.get('href'))).status_code == 200:
                if len(imgTag) == 1:
                    return True
    imgTags = parsedHTML.find_all('img')
    for img in imgTags:
        aTag = img.find_all('a')
        if len(aTag) > 0:
            try:
                if aTag[0].get('href').startswith("http"):
                    code = rq.get(a.get('href')).status_code
                else:
                    code = rq.get("http://" + aTag[0].get('href')).status_code
                if code == 200:
                    return True
            except:

```

Figure 8 - checkClickPicture function

The checkClickPicture function again takes a URL of the page to be checked as a parameter and then uses a get request and beautiful soup to parse the HTML of the page. It then uses the find_all function to look for any 'a' tags within the HTML document. It then iterates over the list of 'a' tags and uses the find_all function to find any 'img' tags within the 'a' tag. It then checks the href of the 'a' tag and performs a get request using the requests library to return a status code and checks if the status code is 200. If the status code is 200, then it means the link is not broken and the link is valid. If this is the case, then the function returns true. If nothing is returned here the function performs the same thing but iterates over all the 'img' tags and checks for any 'a' tags inside them. This function works as if there is an 'a' tag that contains an image or an image that contains an 'a' tag then the picture will be clickable. If the link returns a valid HTTP response, then the test is valid and can be passed.

```
def checkThreeParagraphs(linkToPage):
    parsedHTML = getParsedHTML(linkToPage, "GET")
    pTags = parsedHTML.find_all('p')
    count = 0
    for p in pTags:
        paragraph = p.get_text()
        wordCount = len(paragraph.split())
        if wordCount > 40:
            count += 1
    if count >= 3:
        return True
    else:
        return False
```

Figure 9 - checkThreeParagraphs function

The final test in this python file is the function checkThreeParagraphs, which also takes a link to a page, which is used in a get request HTML in the response is pared in beautiful soup. This HTML document is then checked for any 'p' tags. The list of 'p' tags is then iterated over, and the text of the 'p' tag is then split into words. If the word count of this 'p' tag is more than 40 then it increases a counter by 1. After all, 'p' tags have been checked then the counter is checked if it is at least 3 then the function returns True, else it will return False. The word count is set to 40 as this is an average size of a paragraph in the sample of students who passed this test. If the count is greater than 3 then there are at least 3 'p' tags with a word count greater than 40 so this means that the test can be passed. There is no feasible way to check the contents of these paragraphs so this would be something markers would need to be aware of, as it could be paragraphs irrelevant.

Part 2 Test - Dynamic Site

Main Program

The program for site 2 is run from the similarly named mainProgramPart2.py. This uses the same method to get the arguments for verbose mode and the URL of the site. It also gets the list of years from the config.json file and stores them in a variable. The runChecks method in this section is also similar to the part 1 program. It initialises a checks dictionary which also contains a Boolean value for each of the tests. However, the format input contains a Boolean value for each of the format inputs (Verbose, Numeric, and Both). The Correct Output value contains a list of Boolean values for each year. The first is for correct numeric output, the second two values are for both verbose and correct superscript for the verbose, the third is for correct output for both and the final is for processing the correct year. A dictionary is again used here as each value can be accessed easily with the key and updated and changed individually.

Similarly, to the site 1 code, each test is run from this function, including testing the easter outputs. The easter outputs are checked with each year given in the configuration file and are checked for each output (verbose, Numeric and Both). The Boolean value for each of these tests is stored in their respective dictionary value.

Again, similar to the site 1 testing, the returnResults function takes the checks dictionary as a parameter and then outputs whether each test has passed or failed. When outputting for the years, for each marking criterion, (Verbose, numeric, superscript, and both), a counter is initialised. Each year is iterated over and if the counter is iterated for each true value. The user is shown how many times each criterion is passed, out of the given years. Again the strung colour library was used throughout to make the messages easier to distinguish, for example, the tests that were passed in green and the failed tests in red.

```
print("\nSite should process user input to correctly output the date of easter as a HTML page")
atLeastCount = 0
numericCount = 0
verboseCount = 0
verboseAndNumericCount = 0
superscriptCount = 0
for i in range(len(years)):
    if(checks["CorrectOutput"][i][3]):
        atLeastCount += 1
    if(checks["CorrectOutput"][i][0]):
        numericCount += 1
    if(checks["CorrectOutput"][i][1][0]):
        verboseCount += 1
    if(checks["CorrectOutput"][i][2]):
        verboseAndNumericCount += 1
    if(checks["CorrectOutput"][i][1][1]):
        superscriptCount += 1

print("\nSite at least echoes the year, the user has input")
print ("Passed for", atLeastCount, "of", len(years), "years")
print("Site shows correct numeric output")
print ("Passed for", numericCount, "of", len(years), "years")
print("Site shows correct Verbose output")
print ("Passed for", verboseCount, "of", len(years), "years")
print("Site shows both verbose and numeric outputs correctly")
print ("Passed for", verboseAndNumericCount, "of", len(years), "years")
print("Site shows correct Superscript with Verbose output")
print ("Passed for", superscriptCount, "of", len(years), "years")
```

Figure 10 - main program site 2

Part 2 Tests

```
outputOptionsDictionary = {
    "Numerically": ["numerical", "numerically", "dd", "num", "dd/mm/yyyy", "ddmmyyyy", "numeric", "1", "number", "dd-mm-yyyy", "date"],
    "Verbosely": ["verbose", "verbosely", "verb", "named_month", "text", "str", "2", "word", "verbal", "daymonthyear", "day_month_year", "day-month-year", "full"],
    "Both": ["both", "3"]
}
```

Figure 11 - outputOptions dictionary

The outputOptions dictionary contains a list of all the names that previous students have used for their radio button values for the numeric, verbose and both outputs. This will be used to compare any options that future students will use and allow the tool to work out what each radio button should be used for. I have used this method as it can be added I have checked against many student submissions in the last year and most of the submissions use a variant of these options. Likely, a student using an option that is not on this list would be an infrequent occurrence and if so, their test would not pass, however, this could easily be checked by a marker manually. Also, there is another option name used it can easily be added to the dictionary which will improve the tool in future. I decided to use the output of the radio buttons to determine the function as this could be not as accurate as students could map a button to the wrong output which would make the test inaccurate.

```
def getEasterResponses(siteURL):
    parsedHTML = getParsedHTML(siteURL, "GET")
    findForm = parsedHTML.find("form")
    requestMethod = findForm.get("method")
    action = rq.compat.urljoin(siteURL, findForm.get('action'))
    #action = + "/" + findForm.get('action')
    getInputs = findForm.find_all("input")
    selectOptions = [0,0,0]
    optionVar = ""
    for inputs in getInputs:
        if inputs.get("type") == "text" or inputs.get("type") == "number":
            findYearInput = inputs.get("name")
            break
    for inputs in getInputs:
        if inputs.get("type") == "radio":
            if optionVar == "" or inputs.get("name") == optionVar:
                if([option for option in outputOptionsDictionary["Numerically"] if(option in inputs.get("value").lower())]):
                    selectOptions[0] = inputs.get("value")
                    optionVar = inputs.get("name")
                elif([option for option in outputOptionsDictionary["Verbosely"] if(option in inputs.get("value").lower())]):
                    selectOptions[1] = inputs.get("value")
                    optionVar = inputs.get("name")
                elif([option for option in outputOptionsDictionary["Both"] if(option in inputs.get("value").lower())]):
                    selectOptions[2] = inputs.get("value")
                    optionVar = inputs.get("name")
    getVariables = {
        "getURL": action,
        "request": requestMethod,
        "findYearVariable": findYearInput,
        "outputOptions": selectOptions,
        "optionVar": optionVar
    }
    return getVariables
```

Figure 12 - getEasterResponses

The function is the getEasterResponses function which takes the site URL and parses using the BeautifulSoup library. It then finds any form tags in the HTML document using the find

function. This is used to get the form that the user would input the year and format options into. For this form, I have used BeautifulSoup again to find the method attribute, which returns the link to the action for the form. Using the `urljoin` function in the requests library I have combined this with the relative URL to get the link for the action to get the student's Easter output. Then again, I have used the BeautifulSoup library to find any input tags. Each input tag is checked for whether it contains a "type" attribute which is either text or number. If it is either of them then it is stored in the year input variable and the loop is broken as the program has found the input for the year.

The select options list is initialised as an array of three values. The first is for the 'numeric' input variable, the second is for the 'Verbose' input variable and the third is for the 'both' input option. I have used this as a simple way to store the output options for each of the required outputs and if one isn't there it can easily be recognised.

The program then loops through all the inputs found and makes sure that they are of type radio, meaning they are radio buttons. It also makes sure the name of the radio button is the same as the previous one checked. Using multiple if statements it checks if the input value attribute matches any of the outputOptions dictionary mentioned earlier. If it matches or contains one of the values, the input attribute name is placed in the relevant index in the select options array, so for example, if it contains one of the values in the numeric dictionary value then it would be placed in the first position in the select options list. This means that if all input options are found, the list should contain a list of all the format options. If any are not found, then it will be a blank element in the list.

The function then returns a dictionary containing the action link, the request method, the name of the year input variable, the select options list, and the name of the radio buttons.

```
def getEasterOutput(action, method, yearInputVar, year, optionVar, option):
    if method == "GET":
        getRequest = action + "?" + yearInputVar + "=" + str(year) + "&" + optionVar + "=" + str(option)
        print (getRequest)
        return rq.get(getRequest).text
    elif method == "POST":
        postRequestData = {yearInputVar: year, optionVar: option}
        postRequest = rq.post(action, postRequestData)
        return postRequest.text
    else:
        return None
```

Figure 13 - *getEasterOutput* function

The `getEasterOutput` function takes the parameters for the action, the request method, the year being tested, the format option variable and the format option name. Depending on whether a POST or GET request is being used the tool will construct the string for the GET request or create a dictionary for a POST request. It will then perform the relevant request and return the text data for this. This data will be used to pass on to the function that will compare the HTML to the correct HTML, which will include superscript. Therefore, it is not parsed as it would be easier to compare the strings directly as they both will have superscript HTML tags. If there is no method or it is not GET or POST then the code will return None, as there is no relevant method to use.

```
def getCorrectEasterOut(year):
    getEaster = findEaster(year)
    correctDates = {
        "Numeric": "",
        "Verbose": "",
        "VerboseNoSup": ""
    }
    correctDates["Numeric"] = (getEaster.strftime('%d/%m/%y'))
    correctDates["VerboseNoSup"] = getEaster
    if getEaster.day == 1 or getEaster.day == 21 or getEaster.day == 31:
        correctDates["Verbose"] = (getEaster.strftime('%d<sup>st</sup> of %B %Y'))
    elif getEaster.day == 2 or getEaster.day == 22:
        correctDates["Verbose"] = (getEaster.strftime('%d<sup>nd</sup> of %B %Y'))
    elif getEaster.day == 3 or getEaster.day == 23:
        correctDates["Verbose"] = (getEaster.strftime('%d<sup>rd</sup> of %B %Y'))
    else:
        correctDates["Verbose"] = (getEaster.strftime('%d<sup>th</sup> %B %Y'))
    return correctDates
```

Figure 14 - `getCorrectEasterOut` function

The `getCorrectEasterOut` function takes a parameter of the year being tested. It calls the `findEaster` function with that year, which returns a `DateTime` value for the date of Easter for that given year. The `correctDates` dictionary contains the correct values for Easter for the Numeric option, the Verbose option as well as the Verbose option with no superscript. The code then sets the correct Numeric output to the string which is dd/mm/yy. Based on what number day of the month is given, the correct output with superscript is stored in the Verbose value. For the non-superscript verbose output, the `DateTime` value is stored here. This is because the student output just needs to contain the relevant numbers and month names so this will be processed in a different function. In the case of the Numeric and Verbose with superscript, the output just needs to be compared and checked if the output contains the correct output. The function then returns the dictionary of the correct outputs.

```

def checkOutput(correctOutput, studentOutput):
    check = {
        "Numerically": False,
        "Verbosely": [False, False],
        "Both": False,
        "Year": False,
        "NumOutput": ""
    }

    if(studentOutput[5][0] != 0):
        numOutput = getEasterOutput(studentOutput[0], studentOutput[1], studentOutput[2], studentOutput[3], studentOutput[4], studentOutput[5][0])
        if numOutput.__contains__(correctOutput["Numeric"]):
            check["Numerically"] = True
            check["Year"] = True
        elif numOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%y')):
            check["Year"] = True

    if(studentOutput[5][1] != 0):
        verbOutput = getEasterOutput(studentOutput[0], studentOutput[1], studentOutput[2], studentOutput[3], studentOutput[4], studentOutput[5][1])
        if verbOutput.__contains__(correctOutput["Verbose"]):
            check["Verbosely"] = [True, True]
            check["Year"] = True
        elif (verbOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%d')) and verbOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%B')) and verbOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%y'))):
            check["Verbosely"] = [True, False]
            check["Year"] = True
        elif verbOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%y')):
            check["Year"] = True

    if(studentOutput[5][2] != 0):
        bothOutput = getEasterOutput(studentOutput[0], studentOutput[1], studentOutput[2], studentOutput[3], studentOutput[4], studentOutput[5][2])
        if (bothOutput.__contains__(correctOutput["Numeric"])) and ((bothOutput.__contains__(correctOutput["Verbose"])) or (bothOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%y')))):
            check["Both"] = True
            check["Year"] = True
        elif bothOutput.__contains__(correctOutput["VerboseNoSup"].strftime('%y')):
            check["Year"] = True

    check["NumOutput"] = numOutput
    return check

```

Figure 15 - checkOutput function

The function check output takes the correct output and the student output data. It creates a dictionary called 'check'. It is a Boolean value for Numeric, Two Boolean values for Verbose – one for superscript and one for correct verbose values, and A Boolean value for Both output, one for Year only. It first checks the easter output with the numeric format. To check this, it checks if the Student Easter output, which is retrieved by running the getEasterOutput function, contains the correct output for the numeric format given in the correctOutput dictionary. I have chosen this approach as it means the data can be anywhere in the student's response, if it contains the correctOutput then the test will pass. The same is done for the Verbose mode, however, an additional if statement is used to check if there is the correct superscript, again this is done by checking if the student output contains the correct output. For the 'both' output option it is checked that the student output contains both the verbose and numeric correct outputs and will only pass if both are present in the output

Common Tests

```

def checkURL(siteURL):
    try:
        siteRequest = (rq.get(siteURL))
        statusCode = siteRequest.status_code
        if statusCode == 200:
            return [True, 200]
        else:
            return [False, statusCode]
    except:
        return [False, "Connection Error or URL does not exist"]

```

Figure 16 - checkURL function

This checkURL function takes a URL as a parameter and checks it returns a status code of 200. This means that the URL is valid. If the URL does not return any response or doesn't connect a connection error will be shown. This code is to check any URLs that are entered are valid before using the code.

```
def getParsedHTML(request, method):
    if method == "GET":
        getSite = rq.get(request)
    elif method == "POST":
        getSite = request
    else:
        return None
    html = getSite.text
    parsedHTML = BeautifulSoup(html, 'html.parser')
    return parsedHTML
```

Figure 17 - getParsedHTML function

The getParsedHTML function takes a request and a method and returns the HTML for that URL. As this happens a lot in the tests, I have decided to write this as a separate function so it can be easily reused in the code.

```
def checkLastModified(listOfPages, submissionDate):
    deadlineDict = {}
    for pages in listOfPages:
        lastModified = rq.get(pages).headers['Last-Modified'][:4]
        lastModifiedDate = datetime.strptime(lastModified, "%a, %d %b %Y %H:%M:%S")
        submissionDateDate = datetime.strptime(submissionDate, "%d-%m-%Y %H:%M:%S")
        if lastModifiedDate < submissionDateDate:
            deadlineDict[pages] = [True, lastModified]
        else:
            deadlineDict[pages] = [False, lastModified]
    return deadlineDict
```

Figure 18 - checkLastModified function

The checkLastModified function takes a list of pages as well as the submission date. It iterates over each page and gets the last modified date and time from the header, using the requests library which can view the headers of a response. It then converts this into a date-time value. It adds the page to a dictionary, with each page having a Boolean value and a last modified value. The Boolean value represents whether the page has been modified after the submission date or not. The dictionary of all pages is then returned by the function.

```
def checkValidHTML(siteURL):
    getSite = rq.get(siteURL)
    siteHtml = getSite.text
    document, errors = tidy_document(siteHtml,
    options={'numeric-entities':1})
    return errors
```

Figure 19 – checkValidHTML function

The checkValidHTML function takes the site URL as a parameter and then sends the text returned by the request through a library called tidylib, which lists any errors in the document. This list of errors is then returned by the function.

```
def checkForCss(linkToPage):
    parsedHTML = getParsedHTML(linkToPage, "GET")
    check = False
    head = parsedHTML.find('head')
    if head != None or head != "":
        linkTag = head.find('link')
        if linkTag.get('href') != None or linkTag != "":
            cssText = rq.get(rq.compat.urljoin(linkToPage, linkTag.get('href'))).text
            if cssText.__contains__("font-family:") and cssText.__contains__("font-size:") and cssText.__contains__("color:"):
                check = True
    return [check, linkTag.get('href')]
```

Figure 20 - checForCSS function

The final function checkForCSS contains a link to the page, and again using a get request and BeautifulSoup checks the head tag for any link attributes which would contain the CSS stylesheet. It then performs a get request on the link for the stylesheet and checks if the text contains the font family, font size and colour attributes, which makes sure that the CSS sets these values.

Reading from Console and Config File

```
{
    "years": [1975, 2156, 1945, 1889, 2004, 2078, 2000, 1961, 1962],
    "submission_date": "20-03-2022 23:59:59"
}
```

Figure 21 - config.json file

The two pieces of data that is needed to be retrieved from a config file are the Submission date and the list of years that need to be tested. To implement this, I have used a JSON file. I have used a format of a list to store the years to be tested as well as the submission date after that. This is easily editable by the user and is also easily read in python. They can also be in any order if the title of the data is correct. Therefore, JSON was preferable to using another form of storage. It can also be adapted to add more config data in future if more were to be needed.

Results and Evaluation

Testing Strategy

Testing was mostly performed during the implementation of the project, unlike most traditional methods where testing is performed at the end of the implementation. This allowed me to see any errors that were present at the time of implementing them and then be able to fix them when they came up. Each function for each test of the site, for example, the check for the table, was tested to make sure it passed with a valid site as well as failed for an invalid site. This made sure each of the functions worked properly when written, as well as performed the task they were supposed to without errors.

I performed dedicated testing after the implementation, to make sure the entire program worked as expected. Even though the individual functions may have worked for each test, it was important to check the program works as intended. I built my version of the sites and installed them on the University Project server. This allowed me to change the site to allow the test to fail or pass. This allowed me to check if the testing failed when the site did not meet the marking criteria. I also had access to the previous years' student sites, which allowed me to test on various other projects that may have been in a different format or taken a different approach. It also allowed the system to be tested in an environment that is similar to the way it would be used when actually deployed.

Testing Full System with Previous Submissions – Site 1

For each site tested I used the tool to test the site and used then also checked the site manually, by opening the site and checking in the way the marker currently does without the tool. I checked whether it passed/failed for the automated test for each site and did the same for the manual test.

There were some inconsistencies when testing on different submissions, this was mainly down to the approach that the students used for the various marking criteria. Importantly, there were no False-positive tests, if the test wasn't supposed to pass and didn't pass the manual test just by looking over it, then the test never passed. There were, however, some false negatives where, which in these circumstances the marker would have to look over that specific section manually. Mostly, the reason for false-negative tests was when a student used a different approach to the one that was used by the automated tool. These were:

- Using Nav instead of ol, ul for the nav menu
- Using a different approach for creating a table
- Using smaller paragraphs than the 40-word count set by the tool.

Below are the results for each of the sites.

Site 1 - Test 1

Test URL: https://project.cs.cf.ac.uk/BeavisJ1/static/index.html

Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Failed – Couldn't find nav menu	Passed
Table with at least 3 rows and 2 columns	Failed – Couldn't find table	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Failed	Failed
Submission Date	Passed	Passed

Site 1 - Test 2

Test URL: https://project.cs.cf.ac.uk/DaviesJ164/home.html		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Failed – only one site with adequate paragraphs	Failed
Submission Date	Passed	Passed

Site 1 – Test 3

Test URL: https://project.cs.cf.ac.uk/GreerC1/part1/index.html		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed

Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Failed – no table found	Failed – no table seen
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Failed	Failed
Submission Date	Passed	Passed

Site 1 – Test 4

Test URL: https://project.cs.cf.ac.uk/HowarthA1/part1/Home.html		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Failed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Failed - Couldn't find enough sites with adequate paragraphs	Passed
Submission Date	Passed	Passed

Site 1 – Test 5

Test URL: https://project.cs.cf.ac.uk/ThakrarA/Part1/		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Failed - Couldn't find enough sites with adequate paragraphs	Passed

Submission Date	Passed	Passed
-----------------	--------	--------

Site 1 – Test 6

Test URL: https://project.cs.cf.ac.uk/SwarbrickAO/CM1102/part%201/index.html		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Passed	Passed
Submission Date	Passed	Passed

Site 1 – Test 7

Test URL: https://project.cs.cf.ac.uk/StodolnicB/INDEX-PT-1.html		
Marking Criterion	Automated Test	Manual Test
At least 3 pages of code	Passed	Passed
Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Passed	Passed
Submission Date	Passed	Passed

Site 1 – Test 8

Test URL: https://project.cs.cf.ac.uk/PhillipsA36/Part%201/		
Marking Criterion	Automated Test	Manual Test

At least 3 pages of code	Passed	Passed
Nav menu on each page	Passed	Passed
Table with at least 3 rows and 2 columns	Passed	Passed
External Style Sheet that sets font-family etc.	Passed	Passed
Number of paragraphs of content	Passed	Passed
Submission Date	Passed	Passed

Testing Full System with Previous Submissions – Site 2

Similar to testing site 1 I also used the tool to check the easter output and performed a manual check to make sure the tests passed correctly and failed correctly.

During the testing of Site 2, I found that once again there were no false-positive tests. Even though there were some inconsistencies between the manual test and the automated test. These again would have to be checked manually by the marker in the case of a false-negative test.

I found that most of the failed tests were in the checks for the numeric output for the Easter output. I found that this was because some students outputted as “12/4/2020” where the program is looking for “12/04/2020”. As Python doesn’t have a way to format this, I would have to look into this more thoroughly to make sure this is fixed in the final release. The other reason was mostly the Superscript which was failing because some students used a different format to the way the tool was looking for. For example, the code needed the “of” in-between the date so it wouldn’t mark as passed. This would also need to be fixed in the final release. Below are the tests performed on each of the sites

Site 2 – Test 1

Test URL: https://project.cs.cf.ac.uk/BeavisJ1/dynamic/cgi-bin/Easter.py		
Marking Criterion	Automated Test	Manual Test – input of 2020
HTML page that accepts user input	Passed	Passed
Page is styled with external CSS	Passed	Passed
Program that prints year based on input	Passed	Passed
Correctly outputs date of easter	Passed – For Verbose and Numeric, however Failed for both	Passed

Verbose output contains correct superscript	Failed	Passed
Submission Date	Passed	Passed

Site 2 – Test 2

Test URL: https://project.cs.cf.ac.uk/DaviesJ164/easter_calc.html		
Marking Criterion	Automated Test	Manual Test – input of 2020
HTML page that accepts user input	Failed – no both option	Failed – no both option
Page is styled with external CSS	Passed	Passed
Program that prints year based on input	Passed	Passed
Correctly outputs date of easter	Passed – For Verbose mode, but not numeric or both (as there was no both option)	Passed – Failed for both option as there was none
Verbose output contains correct superscript	Failed	Passed
Submission Date	Passed	Passed

Site 2 – Test 3

Test URL: https://project.cs.cf.ac.uk/GreerC1/part2/html/easter_numbers.html		
Marking Criterion	Automated Test	Manual Test – input of 2020
HTML page that accepts user input	Failed – no both option	Failed – no both option
Page is styled with external CSS	Passed	Passed
Program that prints year based on input	Passed	Passed
Correctly outputs date of easter	Passed – For Verbose mode, but not numeric or both (as there was no both option)	Passed – Failed for both option as there was none
Verbose output contains correct superscript	Failed	Passed
Submission Date	Passed	Passed

Site 2 – Test 4

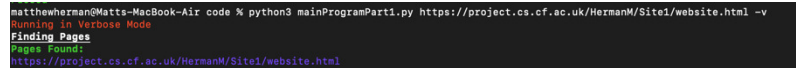
Test URL: https://project.cs.cf.ac.uk/HowarthA1/part2/form.html		
Marking Criterion	Automated Test	Manual Test – input of 2020
HTML page that accepts user input	Passed	Passed
Page is styled with external CSS	Passed	Passed
Program that prints year based on input	Passed	Passed
Correctly outputs date of easter	Passed – For Verbose mode, but not numeric or both	Passed
Verbose output contains correct superscript	Passed	Passed
Submission Date	Passed	Passed

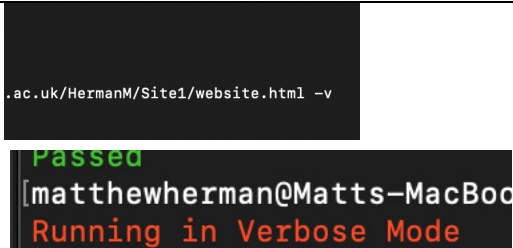
Site 2 – Test 5

Test URL: https://project.cs.cf.ac.uk/PhillipsA36/Part%202/index.html		
Marking Criterion	Automated Test	Manual Test – input of 2020
HTML page that accepts user input	Passed	Passed
Page is styled with external CSS	Passed	Passed
Program that prints year based on input	Passed	Passed
Correctly outputs date of easter	Passed – For Verbose mode, numeric but not both	Passed all
Verbose output contains correct superscript	Passed	Passed
Submission Date	Passed	Passed

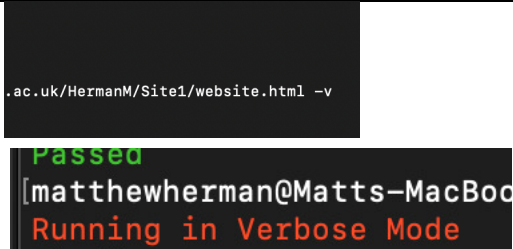
Testing based on Requirements

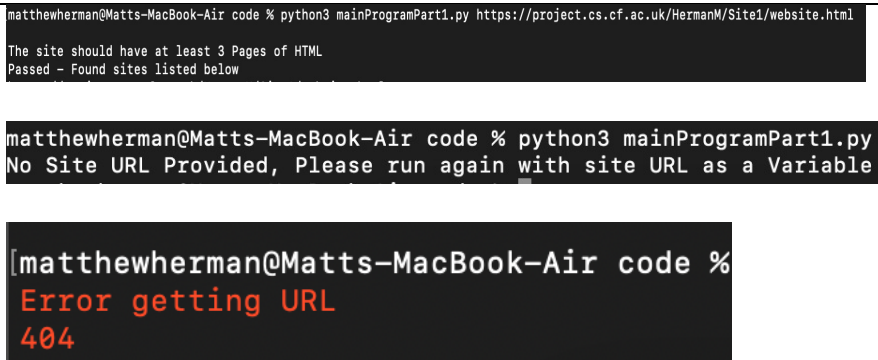
I have also tested the tool based on each of the requirements. For each of the Requirements relating to the marking criteria, I edited the site for each marking criterion that I had created to make sure to include a case for a test that should fail and a test that should pass. The requirements would only be fully satisfied if this was the case. As seen in the tests performed below, all the requirements were at least partially satisfied, with most of them fully meeting the requirements.

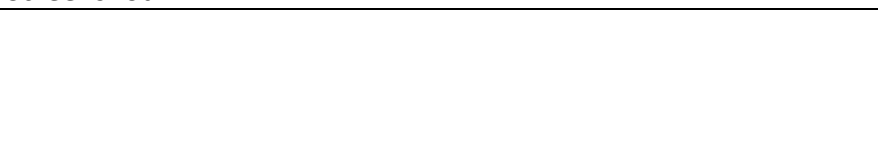
FR1: The tool should allow for user input in the console for the URL of the site to be inputted		
Acceptance Criteria: <ul style="list-style-type: none"> - The user will run the script from the console with the URL as an argument - The code will run with that site as the one being checked 		
Screenshot	Comments	Satisfied
	The users runs the script in the terminal with a URL in the arguments and the code is run with the URL	Yes

FR2: The tool should allow users the option to opt for a verbose mode which shows more options		
Acceptance Criteria: <ul style="list-style-type: none"> - The user will run the script from the console with the URL as an argument - The code will run with that site as the one being checked 		
Screenshot	Comments	Satisfied
	The tool shows more output when the flag "-v" is added as an argument	Yes

FR2: The tool should allow users the option to opt for a verbose mode which shows more options		
Acceptance Criteria <ul style="list-style-type: none"> - The user will be able to add a verbose argument after the URL when running the script to add verbose mode when running. 		

<ul style="list-style-type: none"> - When added the verbose mode will show a more detailed output than when just running the script without 		
Screenshot	Comments	Satisfied
 <pre> project.cs.ac.uk/HermanM/Site1/website.html -v Passed [matthewherman@Matts-MacBook-Air ~]\$ python3 mainProgramPart1.py https://project.cs.ac.uk/HermanM/Site1/website.html -v Running in Verbose Mode - . . . </pre>	The tool shows more output when the flag “-v” is added as an argument	Yes

FR3: The tool should check if the URL is valid and actually returns an HTTP response		
<p>Acceptance Criteria</p> <ul style="list-style-type: none"> - The tool should run if a valid URL is given. - If no URL is given or if the URL does not return anything an error should be shown 		
Screenshot	Comments	Satisfied
 <pre> [matthewherman@Matts-MacBook-Air ~]\$ python3 mainProgramPart1.py https://project.cs.cf.ac.uk/HermanM/Site1/website.html The site should have at least 3 Pages of HTML Passed - Found sites listed below [matthewherman@Matts-MacBook-Air ~]\$ python3 mainProgramPart1.py No Site URL Provided, Please run again with site URL as a Variable [matthewherman@Matts-MacBook-Air ~]\$ python3 mainProgramPart1.py https://project.cs.cf.ac.uk/HermanM/Site1/website.html Error getting URL 404 </pre>	The tool runs if the URL is valid and if not then will show an error. If no site is given it will also present an error	Yes

FR4: Site 1 - The tool should check Marking Criterion 1 – Whether there are at least 3 pages of HTML code.		
<p>Acceptance Criteria</p> <ul style="list-style-type: none"> - The tool should show the marker whether there are or are at least 3 pages and whether the test has been passed, and failed if not - In verbose mode, the marker should be shown whether the test has been passed as well as what sites have been seen. 		
Screenshot	Comments	Satisfied
 <pre> [matthewherman@Matts-MacBook-Air ~]\$ python3 mainProgramPart1.py https://project.cs.cf.ac.uk/HermanM/Site1/website.html The site should have at least 3 Pages of HTML Passed - Found sites listed below </pre>	The tool checks the site to see if there	Yes

<pre> Running in verbose mode Finding Pages Pages Found: https://project.cs.cf.ac.uk/HermanM/Site1/website.html https://project.cs.cf.ac.uk/HermanM/Site1/History.html https://project.cs.cf.ac.uk/HermanM/Site1/technical.html https://project.cs.cf.ac.uk/HermanM/Site1/references.html </pre>	are at least 4 pages of HTML	
--	------------------------------	--

FR5: Site 1 - The tool should check Marking Criterion 2 – Whether there is a Navigation menu that points to each page of the site

Acceptance Criteria

- The tool should show the marker whether there is a navigation menu on the site and whether the test has been passed, and failed if not
- In Verbose mode, the marker should be shown whether the test has been passed or not.

Screenshot

Checking Nav Menu

```

Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/website.html:
Same Page Nav Menu found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/History.html:
Same Page Nav Menu found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/technical.html:
Same Page Nav Menu found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/references.html:
Same Page Nav Menu found
Same Page Nav Menu found on all pages

```

Comments

The tool checks the site to see if there are at least 4 pages of HTML

Satisfied

Yes

FR6: Site 1 - The tool should check Marking Criterion 3 – Whether there is a table and whether the table has at least two columns and three rows

Acceptance Criteria

- The tool should show whether it has found a table and whether the test has been passed or whether it has failed.
- When in verbose mode the tool should show whether the table has been found, and on what page the table has been found.

Screenshot

Checking for Tables

```

Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/website.html:
No Table found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/History.html:
Table found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/technical.html:
No Table found
Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/references.html:
No Table found

```

Comments

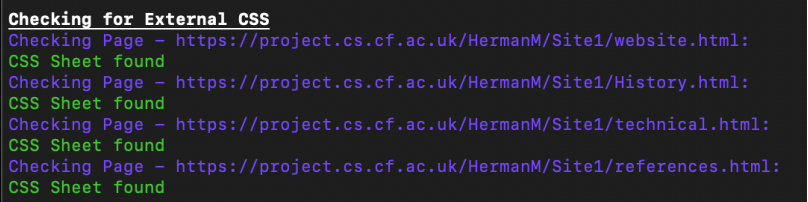
The tool checks the site for a table and returns the page it has been found

Satisfied

Yes

FR7: Site 1 - The tool should check Marking Criterion 4 – Whether the site contains an external CSS that sets the font family, font size and colour.

Acceptance Criteria

<ul style="list-style-type: none"> - The tool should show whether the site contains the external CSS sheet and whether the test has passed or failed - The verbose should show whether the site contains the external CSS and should show a link to the CSS document 		
Screenshot	Comments	Satisfied
 <pre> Checking for External CSS Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/website.html: CSS Sheet found Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/History.html: CSS Sheet found Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/technical.html: CSS Sheet found Checking Page - https://project.cs.cf.ac.uk/HermanM/Site1/references.html: CSS Sheet found </pre>	The tool checks the site for an External style sheet	Partially – as the style sheet is required to be the same and should be flagged if not.

FR8: The site should read the submission date from a config file		
Acceptance Criteria <ul style="list-style-type: none"> - The tool should check the site against the submission date given in the config file 		
Screenshot	Comments	Satisfied
	The tool retrieves the submission date from a JSON file – config.json	Yes

FR9: The site should read the chosen test years from a config file		
Acceptance Criteria <ul style="list-style-type: none"> - The tool should check the easter dates with the years given in the test file 		
Screenshot	Comments	Satisfied
	The tool retrieves the list of years from a JSON file – config.json	Yes

FR10: Site 2 – The tool should check Marking Criterion 1 – Whether the site contains a form that accepts both the year as well as the format of the output.		
Acceptance Criteria <ul style="list-style-type: none"> - The tool should show whether the form and format inputs exist on the site and whether the test has passed or failed 		
Screenshot	Comments	Satisfied

<pre>Running in Verbose Mode Checking for input for year and format Year Input Found: year_input Checking for External CSS</pre>	The tool checks the site for an input box that sets the year.	Yes
--	---	-----

FR11: Site 2 – The tool should check Marking Criterion 2 – Whether the site contains an external CSS stylesheet.

Acceptance Criteria

- The tool should show whether the site contains an external CSS stylesheet
- In the verbose mode, the tool should show whether the CSS stylesheet has been found as well as a link to the CSS stylesheet

Screenshot	Comments	Satisfied
<pre>Checking for External CSS CSS Sheet found css.css</pre>	The tool checks the site contains a CSS externally, in verbose mode the link to the file is shown.	Yes

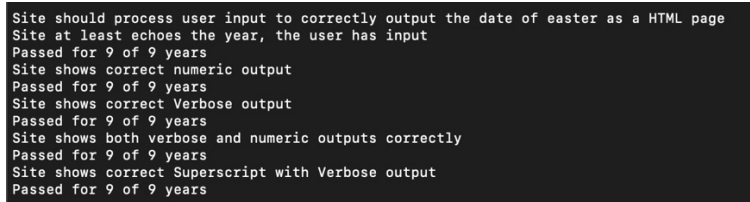
FR12: Site 2 – The tool should check Marking Criterion 3 – Whether the site at least echoes the year, that has been inputted into the form

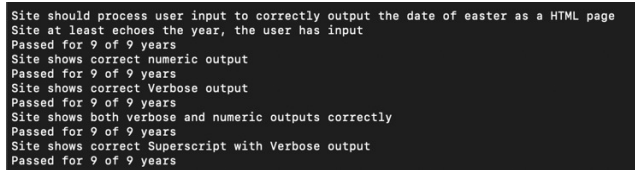
Acceptance Criteria

- The tool should check whether the year given is outputted in the site output and should show how many years passed
- The tool should check this for all years in the config file

Screenshot	Comments	Satisfied
<pre>Site should process user input to correctly output the date of easter as a HTML page Site at least echoes the year, the user has input Passed for 9 of 9 years</pre>	The tool checks the site echoes the year that is given, it checks this for all of the years. These are the years from the config file	Yes

1. FR13: Site 2 – The tool should check Marking Criterion 4 – Whether the site has correctly processed the date for Easter of the given year.

<p>Acceptance Criteria</p> <ul style="list-style-type: none"> - The tool should check whether the correct date has been processed and should show how many years this has passed. - The tool should check this for all years in the config file 		
Screenshot	Comments	Satisfied
 <pre> Site should process user input to correctly output the date of easter as a HTML page Site at least echoes the year, the user has input Passed for 9 of 9 years Site shows correct numeric output Passed for 9 of 9 years Site shows correct Verbose output Passed for 9 of 9 years Site shows both verbose and numeric outputs correctly Passed for 9 of 9 years Site shows correct Superscript with Verbose output Passed for 9 of 9 years </pre>	The tool checks the date of easter, and compares it to the student's calculation and output to make sure the date is the same	Yes

FR14: The tool should check Marking Criterion 5 - Whether the site has correctly outputted for both format options, this being dd/mm/year, and day named month year.		
<p>Acceptance Criteria</p> <ul style="list-style-type: none"> - The tool should check whether the output is correct for all options and should show how many years passed this has passed. - The tool should check this for all years in the config file 		
Screenshot	Comments	Satisfied
 <pre> Site should process user input to correctly output the date of easter as a HTML page Site at least echoes the year, the user has input Passed for 9 of 9 years Site shows correct numeric output Passed for 9 of 9 years Site shows correct Verbose output Passed for 9 of 9 years Site shows both verbose and numeric outputs correctly Passed for 9 of 9 years Site shows correct Superscript with Verbose output Passed for 9 of 9 years </pre>	The tool checks all the output formats and makes sure it is valid for both the Verbose and Numeric outputs, however, sometimes fails the test due to the student outputting slightly differently from the way the tool uses	Partially – The tool sometimes fails the test if the student uses a different valid output

FR14: The tool should check Marking Criterion 5 - Whether the site has correctly outputted for both format options, this being dd/mm/year, and day named month year.		
<p>Acceptance Criteria</p> <ul style="list-style-type: none"> - The tool should check whether the output is correct for all options and should show how many years passed this has passed. - The tool should check this for all years in the config file 		
Screenshot	Comments	Satisfied

<pre> Site should process user input to correctly output the date of easter as a HTML page Site at least echoes the year, the user has input Passed for 9 of 9 years Site shows correct numeric output Passed for 9 of 9 years Site shows correct Verbose output Passed for 9 of 9 years Site shows both verbose and numeric outputs correctly Passed for 9 of 9 years Site shows correct Superscript with Verbose output Passed for 9 of 9 years </pre>	<p>The tool checks all the output formats and makes sure it is valid for both the Verbose and Numeric outputs, however, sometimes fails the test due to the student outputting slightly differently from the way the tool uses</p>	<p>Partially – The tool sometimes fails the test if the student uses a different valid output</p>
--	--	---

FR15: Site 2 – The tool should check Marking criterion 6 – Whether the site has correctly outputted the correct superscript for the date.

Acceptance Criteria

- The tool should check if the correct superscript has been used and should show how many years this has passed.
- The tool should check this for all years in the config file

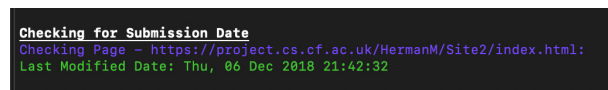
Screenshot	Comments	Satisfied
<pre> Site should process user input to correctly output the date of easter as a HTML page Site at least echoes the year, the user has input Passed for 9 of 9 years Site shows correct numeric output Passed for 9 of 9 years Site shows correct Verbose output Passed for 9 of 9 years Site shows both verbose and numeric outputs correctly Passed for 9 of 9 years Site shows correct Superscript with Verbose output Passed for 9 of 9 years </pre>	<p>The tool checks the verbose mode for the correct superscript. It checks this for all the years from the config file. Similarly, it sometimes fails if the test if the student input is very different from the output the tool requires</p>	<p>Partially – The tool sometimes fails the test if the student uses a different valid output.</p>

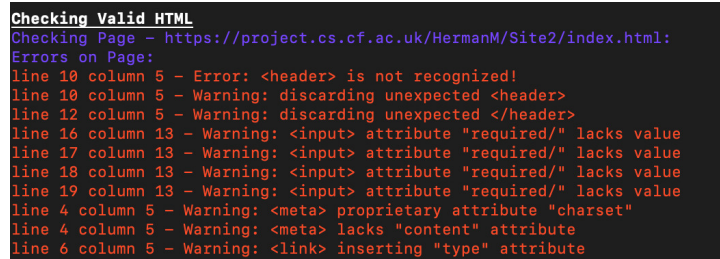
FR16: The tool should check the Modification date, to make sure that the site has not been modified after the submission date.

Acceptance Criteria

- The tool should check if the modification is not after the given submission date.
- The verbose mode should show the last modified date and if this is after the submission date or not

Screenshot	Comments	Satisfied
------------	----------	-----------

 <pre> Checking for Submission Date Checking Page - https://project.cs.cf.ac.uk/HermanM/Site2/index.html: Last Modified Date: Thu, 06 Dec 2018 21:42:32 </pre>	The tool checks the site's last modified date, and makes sure it is not after the submission date given in the config file	Yes
---	--	-----

FR17: The tool should check the site for any errors as well, and present them to the user		
Acceptance Criteria <ul style="list-style-type: none"> - The tool should check if there are any errors on the site and show them to the user when using the verbose mode. 		
Screenshot	Comments	Satisfied
 <pre> Checking Valid HTML Checking Page - https://project.cs.cf.ac.uk/HermanM/Site2/index.html: Errors on Page: line 10 column 5 - Error: <header> is not recognized! line 10 column 5 - Warning: discarding unexpected <header> line 12 column 5 - Warning: discarding unexpected </header> line 16 column 13 - Warning: <input> attribute "required/" lacks value line 17 column 13 - Warning: <input> attribute "required/" lacks value line 18 column 13 - Warning: <input> attribute "required/" lacks value line 19 column 13 - Warning: <input> attribute "required/" lacks value line 4 column 5 - Warning: <meta> proprietary attribute "charset" line 4 column 5 - Warning: <meta> lacks "content" attribute line 6 column 5 - Warning: <link> inserting "type" attribute </pre>	The tool checks the code for any HTML errors and returns these in the verbose mode	Yes

Testing of Non-Functional Requirements

NFR1: When checking the sites, the tool should not have any false positives

- As seen above when checking with the student submissions, there were no false-positive tests, if a test was supposed to fail and did not meet the marking criteria the tool always marked it as failed, based on this the likelihood of the tool causing a failed test to pass is very small. Based on this this requirement has been met.

NFR2: The tool should be able to check the sites and load data in a reasonable amount of time

- When checking the sites, and running the tests on them, the code has not taken an excessively long time, the larger sites with more pages to check have understandably taken longer however they do not take more than 10 seconds to run so this requirement has also been met

NFR3: The interface while being a console app, should be relatively easy for the user to understand what is going on

- The tool uses various colours as well as is clearly laid out when returning results so it should be easy for a user to figure out what is being tested and at what time. When using the verbose mode, there is a lot of text and scrolls very quickly so it may be hard for users to miss a section. Based on this the requirement has been partially met.

Future Work

More Testing and Fixing of Issues

I have performed a lot of testing on the tool with various student approaches to the two sites. However still when testing the site there are several issues that while do not stop the tool from working still cause inconsistencies based on what approach a student has used. So, more testing needs to be performed with more sites to better improve the tool. Also, this needs to be constantly tested as the requirements of the student coursework change and the tool would need to be updated accordingly to fix any issues if the requirements and marking criteria of the task were to change, as if they were not changed this could cause the test to fail unnecessarily

Improvement of Checking Easter Outputs

In the testing of the student submissions, I found that it can fail if the student uses a valid approach to the solution but is just not recognised by the tool yet. An example of this was the tool not recognising the numerical output when the number for the month was a single digit ("12/4/2020" instead of "12/04/2020"), so this was marked as a failed test. Another example was using a nav class instead of an ordered or unordered list to create a navigation menu. As the tool didn't recognise this, it was marked as a failed test. As more students complete the coursework the dictionary of valid names for the format options also should be expanded, allowing more format options to be recognised and marked as passed if used by other students. By doing this, the tool will be able to detect valid tests more accurately and will hopefully result in fewer false-negative tests.

Possible Allowance for User changing the marking Criteria

To improve the usability of the Tool, there could be allowance made for the marking criteria that need to be tested to be changed by the user or the marker. This would mean greater flexibility when the marking criteria are changed or updated to include new sections or changes. This would mean that the tool would not have to be re-written to include the change it would be able to be used again and more quickly without having to go to the developer to change it. However, this may not be feasible in the short term.

Easier Setup and Running of Tool

Currently, the user must have Python installed and be able to navigate the command line to run the tool. In future the tool could be developed to have a Graphical Interface and be able to run from anywhere, this would again improve the usability of the tool and allow more people to be able to use and access it if needed.

Conclusion

The aim of the project was to create a tool that the markers of the Web Applications Coursework can use to automatically test certain elements of the coursework. Overall, I have managed to successfully create this system and based on the requirements set out in the specification initially. Most of the requirements have been met. I have tried to use good programming principles throughout such as the use of reusable code as well as using methods and functions that are more concise and use less resources to complete this task. I was able to recreate a new version of this tool that was previously based on using a web driver and improve on it using a different approach, through using HTTP requests and parsing HTML which hopefully should improve the reliability and make sure the tool continues to work regardless of any change in dependencies. As this tool is currently working and depends only on having Python the requests library, this tool is likely to continue working in future.

However, to improve the project in future, there are still some errors and fixes that need to be made to the tool in order to make it better. Even though there was plenty of testing performed on the tool to make sure it met the requirements, there are still more that could have been done to make sure some of the smaller bugs were found as well as to decrease the likelihood of false negatives, which can still come up a lot. I would also have liked to get more feedback from the project client on what else can be added to make the tool more useful to the markers of the project. It would also have been great to be able to test this with actual users of the tool, so markers and students etc, to be able to see how the tool would perform in the real world and get feedback and improvements from actual users. Overall, I think even though I didn't get to test more thoroughly than I had hoped and better recognition of outputs, would have been beneficial to the project, I believe the tool is still successful in meeting the requirements and working to a degree that will improve the marking of Web Applications Coursework in future. The improvements I have suggested could also be completed afterwards with little work needed to change this.

Reflection

Initially, when taking this project, I looked back on taking the module, Web Applications in my first year, and remember the steps I took to complete the project such as building the sites on Easter and a technology. I was interested in making the marking procedure for this module faster and more efficient which was what this project was designed to do. While this is not fully complete and still needs some work to fix errors and expand some of the data used to check, I believe I have created a tool that will be able to be used in future.

Over the course of completing this project, I was able to use many skills gained in my degree such as software development including the use of good programming principles and improving code reusability. I was also able to follow a software development method to be able to build this tool which gave me more experience in doing this on top of that which my degree has already given me, which while this has been a lot, I haven't had the opportunity to do a full project to completion such as this project has done. On top of this I have been able to use some modules, approaches, and tools that I have never been involved with or used before. These are things such as the use of HTTP requests, as well as parsing and matching HTML in the way that this project has and applying it in a practical and useful way that can be used by real users. The skills and tools I have used are a great addition to my skillset and I feel I will be able to use these approaches in various other projects in future as well as in possible career paths.

In future if I were to do this project again there are a few things that I would improve on. Firstly, I would want to improve my communication with clients. I would work on improving the way I explain the code and what still needs to be done when performing demonstrations of code, as well as when answering any questions. I felt like I could have received more feedback and more input on what was needed in the tool, which would have made the development more effective. While this didn't affect the result it would have made it better. I also would improve on testing and taking more time to fully test and improve the project, as I felt I could have made the product better this way and also improved my skills and knowledge in testing.

To conclude, I found that this project has provided me with many skills and things I can use in future projects, the rest of my degree and in my career path, as these are many things I would need in jobs which involve software development, and when working with developers

References

1. Top 4 software development methodologies
<https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/>
2. Beautiful Soup Documentation
<https://beautiful-soup-4.readthedocs.io/en/latest/>
3. Python Requests Module
https://www.w3schools.com/python/module_requests.asp
4. Automated Testing of Web Applications Coursework
Author – William Bailey