Assignment 2, Part A: What's On?

(21%, due 11:59pm Sunday, May 26th, end of Week 12)

Overview

This is the first part of a two-part assignment. This part is worth 21% of your final grade for IFB104. Part B will be worth a further 4%. Part B is intended as a last-minute extension to the assignment, thereby testing the maintainability of your solution to Part A and your ability to work under time pressure. The instructions for completing Part B will not be released until Week 12. Whether or not you complete Part B you will submit only one solution, and receive only one mark, for the whole 25% assignment.

This a complex and challenging assignment. If you are unable to solve the whole problem, submit whichever parts you can get working. You will receive partial marks for incomplete solutions.

Motivation

In the age of the Internet we're bombarded with entertainment options and new ones emerge every day. It's often observed that we spend more time scrolling through menus than watching programmes! Here you will develop a software application that allows its users to efficiently plan their entertainment options by browsing lists of upcoming events in their preferred entertainment categories and generate a personalised entertainment schedule. Your Python application will have a Graphical User Interface that allows its user to select upcoming events and then export an HTML document describing the events they want to see or attend. This document can be examined in a standard web browser or printed as a hardcopy.

This "capstone" assignment is designed to incorporate all of the concepts taught in IFB104. To complete it you will need to: (a) use Tkinter to create an interactive Graphical User Interface; (b) download web documents using a Python script and use pattern matching to extract specific elements from them; and (c) generate an HTML document integrating the extracted elements, presented in an attractive, easy-to-read format.

Goal

Your aim in this assignment is to develop an interactive "app" which allows its users to preview upcoming entertainment events downloaded from the web. There must be at least three distinct categories of entertainment available, with at least six distinct events in each category at any time. Most importantly, the online web documents from which you collect your events must be ones that are updated on a regular basis, either daily or weekly, so your program needs to be robust to changes in the source documents. To help you develop and debug your code, you will also include the ability to "work offline" using previously downloaded web documents that are never changed.

For the purposes of this assignment you have a free choice of which categories of entertainment events your application will offer, provided there are always at least six items per category, the events are updated frequently, and the information available for each event includes its name, the date/time, and at least one additional piece of information such as an image or a textual description. The upcoming entertainment events could be:

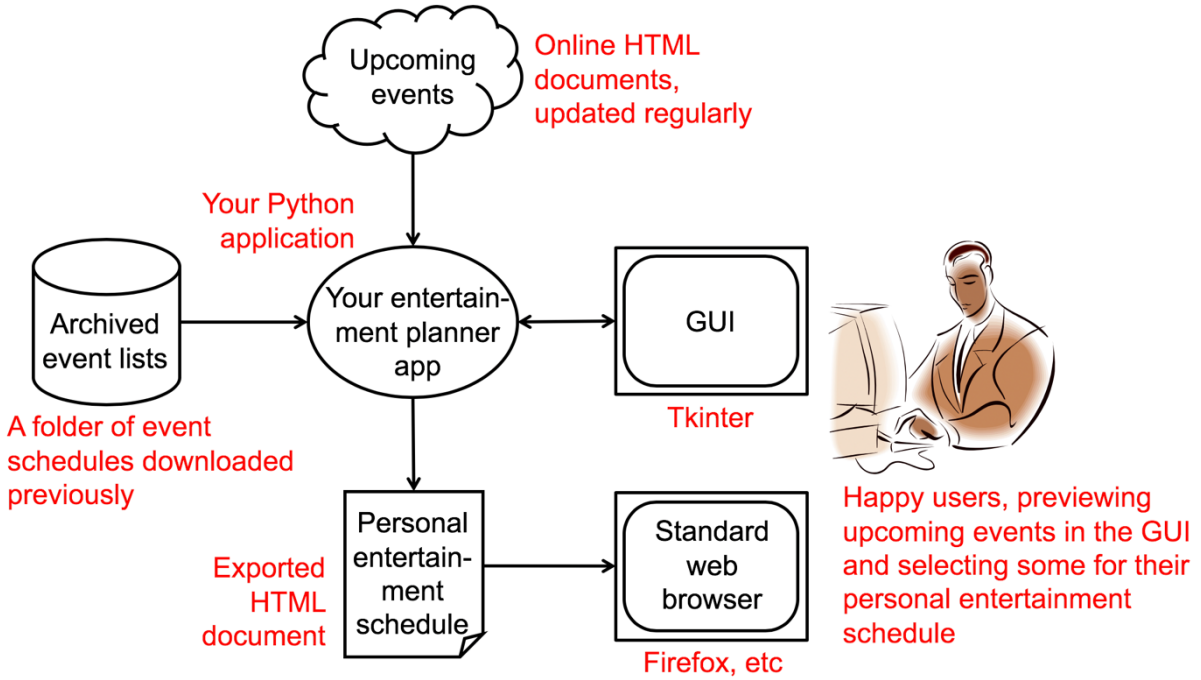
- television programmes,

- radio programmes,
- movies in the cinema,
- live plays,
- community events,
- sporting matches,
- music concerts,
- video game releases,
- streaming media releases,
- home video (DVD and BluRay) releases,
- etc.

However, whichever categories you choose, you must confirm that the online web documents are updated frequently. Appendix A below lists many web sites which *may* be suitable for this assignment, but you are encouraged to find your own of personal interest.

Note: An obvious source for entertainment events is sport. However, you cannot rely on lists of upcoming sporting events being updated "out of season". Therefore, if you choose to use a sports-based category, you must confirm that the sport is being played during the period in which this assignment will be developed and assessed, i.e., mid-April to mid-June 2019!

Using the events appearing in the various categories online you are required to build an IT system with the following general architecture.



Your application will be a Python program with a Graphical User Interface. Under the user's control, it displays lists of upcoming events in the GUI. The user can then select specific events of interest, and export them as an HTML document. This document will contain a de-

tailed itinerary of the selected events which can be studied by the user in any standard web browser at their leisure.

In addition to downloading event lists from online, your program must also provide an "offline" mode in which event categories can be chosen from an "archive" folder of previously-downloaded HTML documents. Although of relatively little value for planning an entertainment schedule, this feature will make it easier for you to develop and debug your code because it allows you to use unchanging web documents as the data source when in this mode.

This is a large and complex project, so its design allows it to be completed in distinct stages. You should aim to complete it incrementally, rather than trying to solve the whole problem at once. A suggested development sequence is:

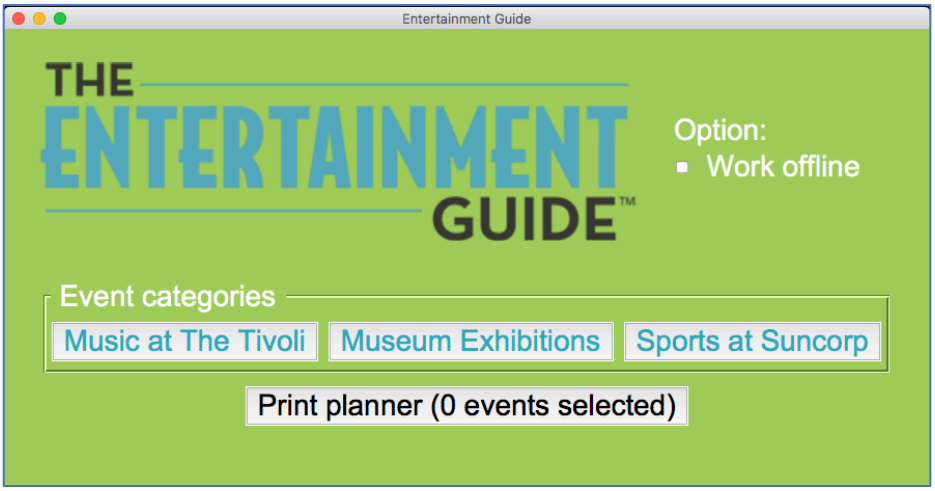
1. Develop code that allows the static, archived event categories to be displayed in the GUI and allows the user to select specific events.
2. Extend your solution so that it allows events from the corresponding "live", online web documents to be displayed.
3. Extend your solution further so that the user's selected events can be exported as an HTML document.

If you can't complete the whole assignment submit whatever parts you can get working. You will get **partial marks for incomplete solutions** (see the marking guide below).

Illustrative example

To demonstrate the idea, below is our own entertainment planner application, which uses information extracted from three different web sites. Our demonstration application allows users to see upcoming live music events at Brisbane's Tivoli Theatre, exhibitions from the Museum of Brisbane, and sporting events at Suncorp Stadium. The application allows users to select which events they wish to see from each category. The program accesses all the necessary information from the three web sites (either "live" or using previously-downloaded copies) and uses it to produce a personalised entertainment schedule in the form of an HTML document which can be viewed in any standard web browser.

The screenshot below shows our example solution's GUI when it first starts. We've called our application *The Entertainment Guide* and have included a suitably evocative image to identify it, but you should choose your own name and GUI design.



The GUI has three push buttons allowing the user to select event categories of interest, another push button for exporting the entertainment schedule, and a check button for choosing whether to work online or offline. Our "*Entertainment Guide*" logo contains text which names the application, but if you choose an image with no text then you must also add a textual label containing your application's name.

You do not need to copy our example and are encouraged to design your own GUI with equivalent functionality. For instance, menus or radio buttons could be used to allow the category selection rather than push buttons.

Selecting upcoming events

When the user selects an entertainment category in "online mode", the application displays a list of upcoming events extracted "live" from the web. For instance, pressing the *Music at the Tivoli* button produced the following display in the GUI on April 17th, 2019.

Our application shows the next ten events coming up in a pop-up window, including the name of the event and the date it will take place. You must display a minimum of six upcoming events for each category, but showing more is obviously better for the user. You do not need to use pop-up windows in your solution; the events could just as easily have been listed in the main GUI window.

For each event our application provides a checkbox allowing the user to choose the events they want to see or attend. In the case above the user has selected three shows they want to go to. Your solution does not need to use checkboxes for this purpose. Any mechanism that allows the user to select events is acceptable, e.g., providing a text entry box that allows the user to type in the number of the event.

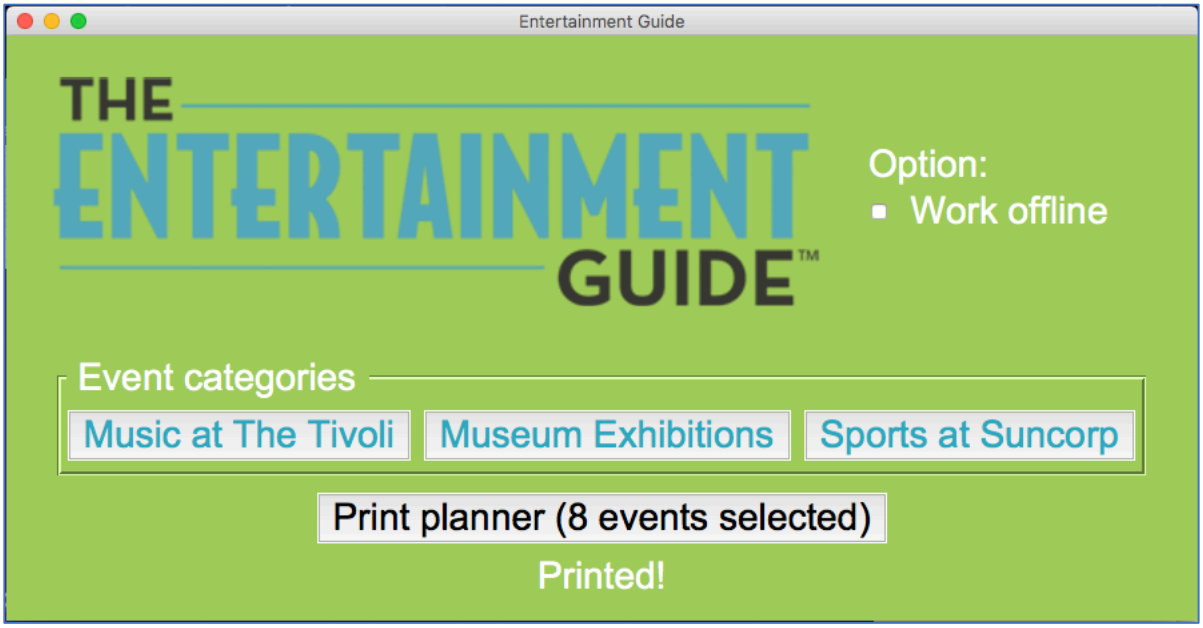
Similarly, pressing the *Museum Exhibitions* button produced the following display in the GUI on April 17th, 2019.

In this case the source web site uses a variety of formats for the dates at which the events occur. Our application simply extracts the date exactly as it appears on the original web site (resulting in a very wide label for the third event in this instance!). Here our imaginary user has selected two museum events to see.

Finally, pressing the Sports at Suncorp button on April 17th, 2019 caused our GUI to display the following list of upcoming rugby and soccer games, of which our user has selected three of interest.

Exporting the entertainment schedule

At this point the user can press the button in the main GUI window to export the selected events as a detailed entertainment plan.



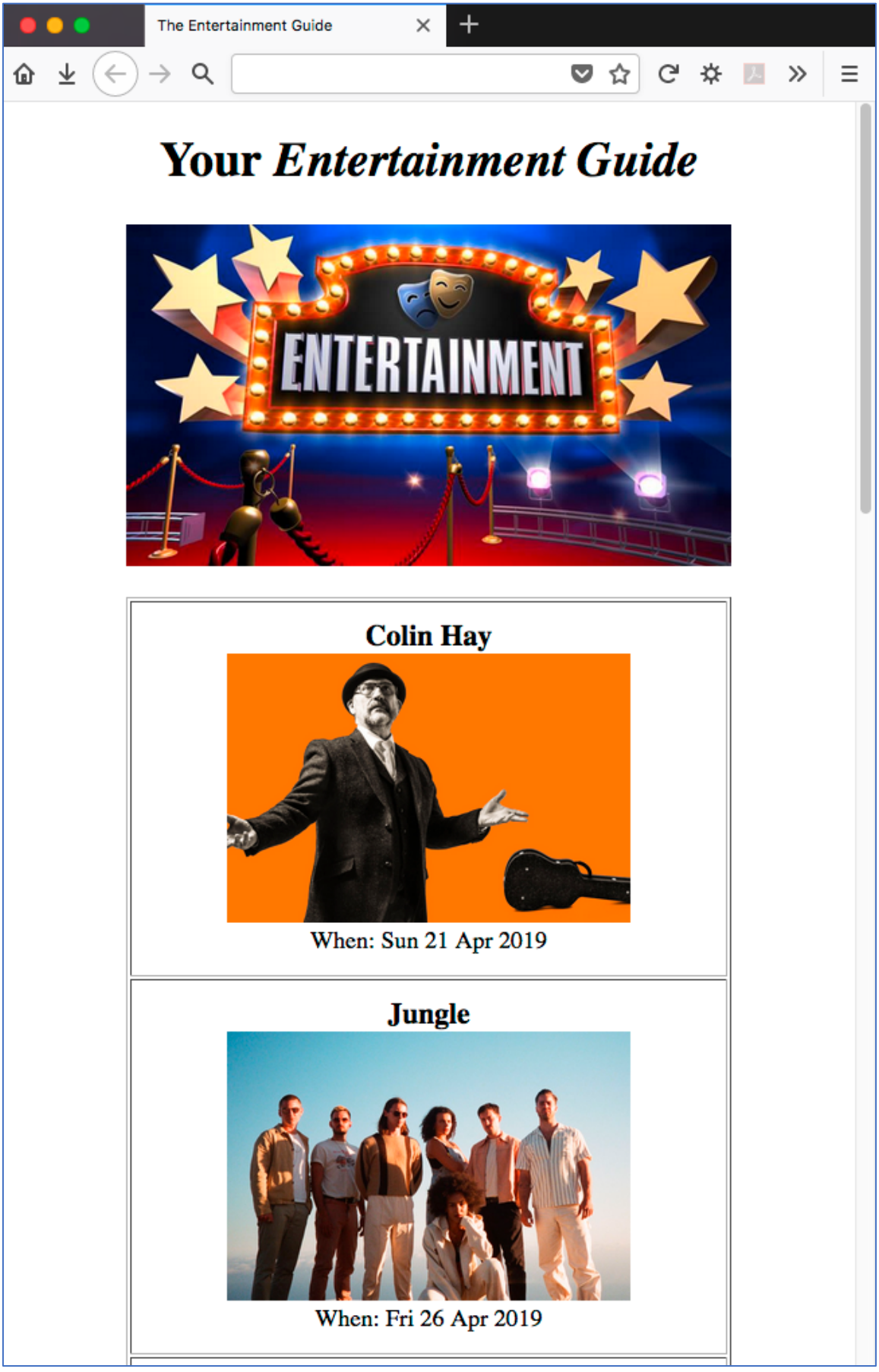
Our program then generates the entertainment schedule as an HTML document describing the events selected by the user. Importantly, this document contains more information about each event than is shown in the GUI. While the GUI shows the event name and date/time, the exported HTML document also includes some additional piece of information about the event. In our demonstration solution this is an image illustrating the event, but some other feature of the event, such as a textual description, could be used instead.

Given the eight selections made by the user above, our app creates an HTML file which looks like the example shown on the following three pages when opened in a web browser.

The generated document contains a title and heading identifying our application and a large image at the top which serves as a logo (different from the one used in the GUI). For each of the events selected by the user it shows the event's name, date/time and an image. At the end it provides links to the original web pages from which the event information was extracted.

Importantly, all the images in the document, including the logo up the top, are online images, not ones stored on our local computer. To ensure that the exported web document is portable and can be viewed on any computer, the images are all links to online images using appropriate HTML "`img`" tags.

You are *not* required to follow the details of our demonstration GUI or exported HTML document. You are strongly encouraged to use your own skills and initiative to design your own solution, provided it has all the functionality and features described herein.


The Entertainment Guide
Your Entertainment Guide
ENTERTAINMENT
Colin Hay
When: Sun 21 Apr 2019
Jungle
When: Fri 26 Apr 2019

When: Fri 26 Apr 2019

Ruel

When: Sun 5 May 2019

The Designers' Guide: Easton Pearson Archive

When: 23 Nov 2018 - 22 Apr 2019

New Woman

When: 13 Sep 2019 - 15 Mar 2020

Brisbane Broncos v Cronulla Sharks

Brisbane Broncos v Cronulla Sharks

When: Sat 27 Apr

NRL Magic Round - Day 1

When: Thu 09 May - Day 1

Queensland Reds v NSW Waratahs

When: Sat 18 May

Events sourced from:

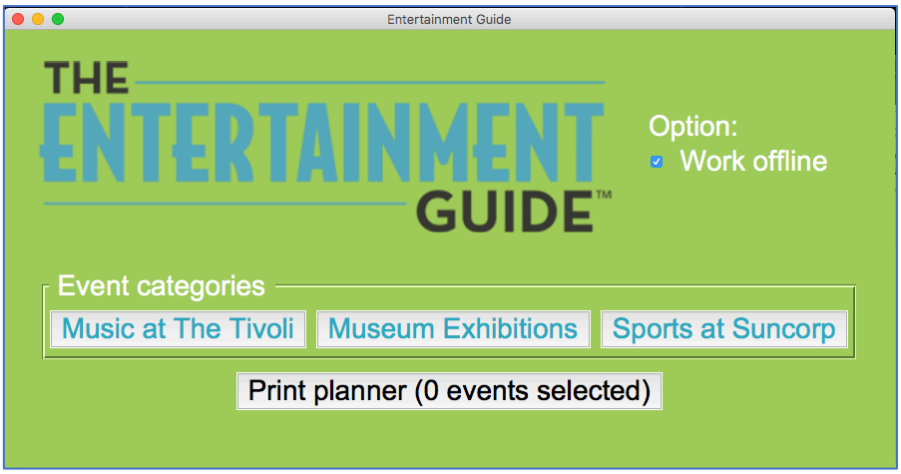
- https://thetivoli.com.au/events
- https://www.museumofbrisbane.com.au/whats-on/
- https://suncorpstadium.com.au/what-s-on.aspx

Working offline

One of the risks of working with online data is that there is always the possibility that the web pages you rely on are inaccessible when someone runs your code. For this reason you are strongly encouraged to use three distinct web sites as the sources of your entertainment events, not just three lists of events from the same site.

Furthermore, since the source web pages change periodically, developing the part of your application that extracts data from the HTML source code can be challenging. Therefore, to address both of these problems, you are also required to provide an "offline mode" for your application. In this mode the event categories are sourced from local files on the computer rather than the Internet.

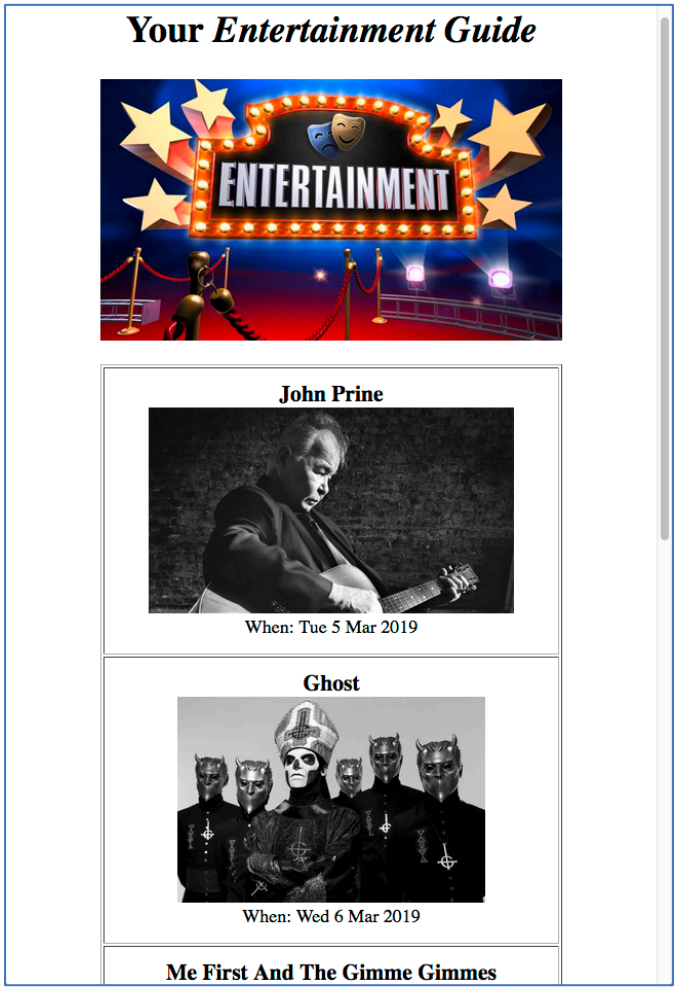
For each of our three entertainment categories we downloaded copies of the corresponding web pages (one on April 10th and the other two on March 5th) and stored them in a folder along with our Python program to serve as an "archive" of old event information. These files are used as the information source when the user chooses to work offline in our application.

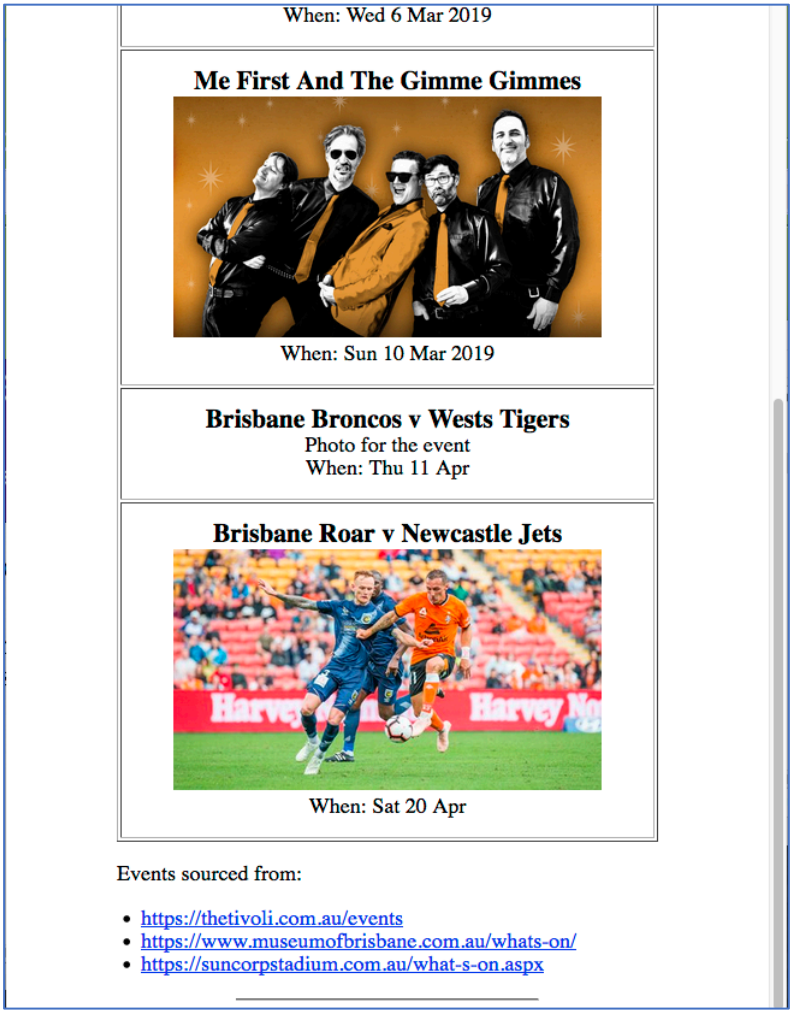


Now when the user chooses an entertainment category this previously downloaded information is displayed, instead of using the current web site. For instance, below are the events shown when the Tivoli and Suncorp categories are selected in offline mode.

Notice that the events are significantly different from those displayed using the current web sites' contents and many of them had already occurred at the time of writing (April 17th).

In this case the "offline" user has selected five events of interest and when the "Print" button is pressed an entertainment schedule containing these events, some of which have already occurred, is produced as follows.





As usual, all the images in the document are links to online files. However, notice that when we view the resulting HTML document in our web browser, the image for one of the sporting events is missing. Since this event has already occurred, the relevant file has already been deleted from the server! In this case we provided some "alternative text" to mark the place where the image should be. (This situation is very rare. You can usually expect web site images to remain available on the server long after they are needed, so most of the time your "archived" web information will display completely even when the event is over. For instance, the three music events at the top all occurred over a month before we ran our program, but the images were still accessible online.)

Extracting the HTML elements

To produce the categories of events for displaying in the GUI and exporting as part of our HTML document, our application used regular expressions to extract elements from the relevant source web documents, whether they were stored in the static archive or downloaded from the Internet when the program runs.

To achieve this, we first downloaded copies of the web pages and then studied them to identify text patterns that would allow us to find the specific parts we wanted. For instance, in the Suncorp web site we found that the sporting event's names always appeared in a heading of the following form.

`<h6 class="event-title">` *event name* `</h6>`

This knowledge was enough to allow us to create a regular expression which returned all the text patterns of this form, and thereby extract the event names using Python's `findall` function. The dates and the URLs of the images were extracted similarly.

Sometimes it's easier to use other Python features as well as, or instead of, regular expressions to help extract the data. For instance, we found that our regular expression for extracting the images from the Suncorp web site also matched various logos at the top of the web page as well as the images of the events we wanted. The first four image URLs matched were always part of the web page's header so, rather than complicating our regular expression, we simply deleted the first four items returned by `findall` each time we extracted data from the page.

In all cases, care was also taken to ensure that no HTML/XML tags or other HTML entities appeared in the extracted text when displayed in either the GUI or the exported HTML documents. In some cases it was necessary to delete or replace such mark-ups in the text after it was extracted from the original web document. The information seen by the user must not contain any extraneous tags or unusual characters that would interfere with the appearance of the events either in the GUI or the exported document.

Exporting the HTML document

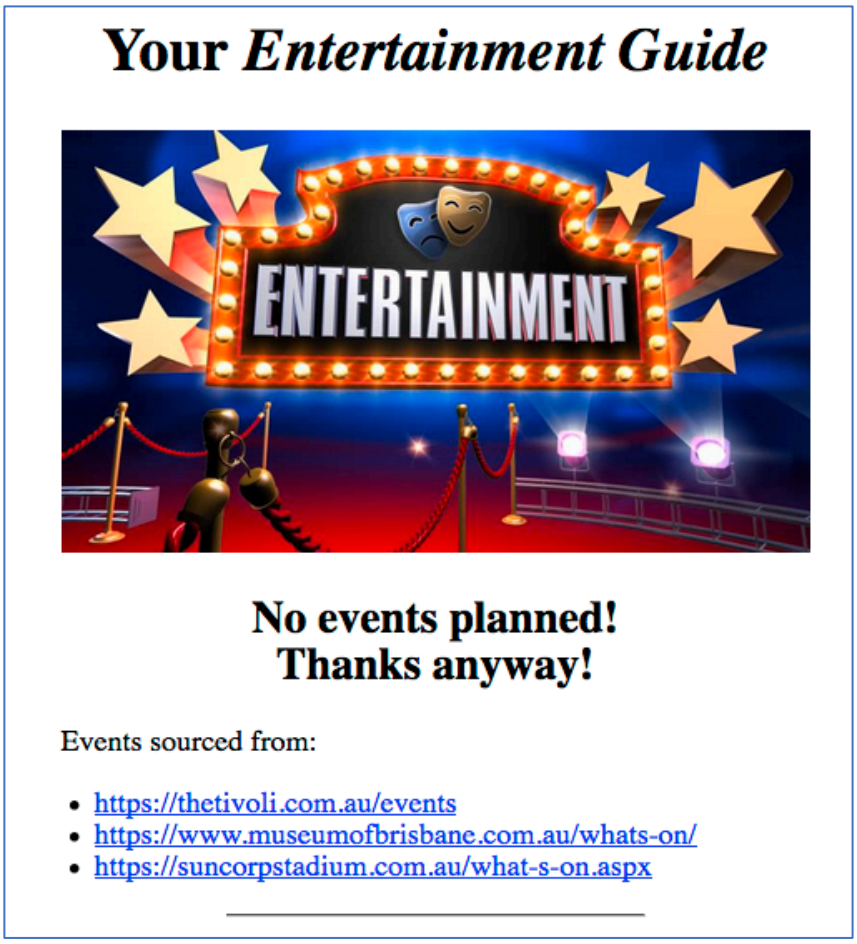
A small part of the HTML code generated by our Python program is shown below. Although not intended for human consumption, the generated HTML code is nonetheless laid out neatly, and with comments indicating the purpose of each part.

```
<!DOCTYPE html>
<html>

  <head>
    <!-- An automatically generated HTML docume
    <meta charset="UTF-8">
    <title>The Entertainment Guide</title>
    <!-- Fix the width of all elements and cent
    <style>
        p {width: 450px; margin-left: auto; mar
        h1 {width: 500px; margin-left: auto; ma
        h2 {width: 450px; margin-left: auto; ma
        table {width: 450px; margin-left: auto;
        ul {width: 450px; margin-left: auto; ma
        td {padding: 15px}
    </style>
  </head>

  <body>

    <!-- Events selected summary -->
    <h1 align="center">Your <em>Entertainment G
    <!-- Entertainment guide logo - online imag
    <p><img src="https://i2.wp.com/www.columbus
    <!-- Alternative image: "https://entertainm

    <!-- Events planned (name, description or i
    <table border=1>

      <!-- Row with event photo -->
```

Robustness

Another important aspect of the system is that it must be resilient to user error. This depends, of course, on your choice of GUI widgets and how they interact. Whatever the design of your GUI, you must ensure that the user cannot cause it to "crash".

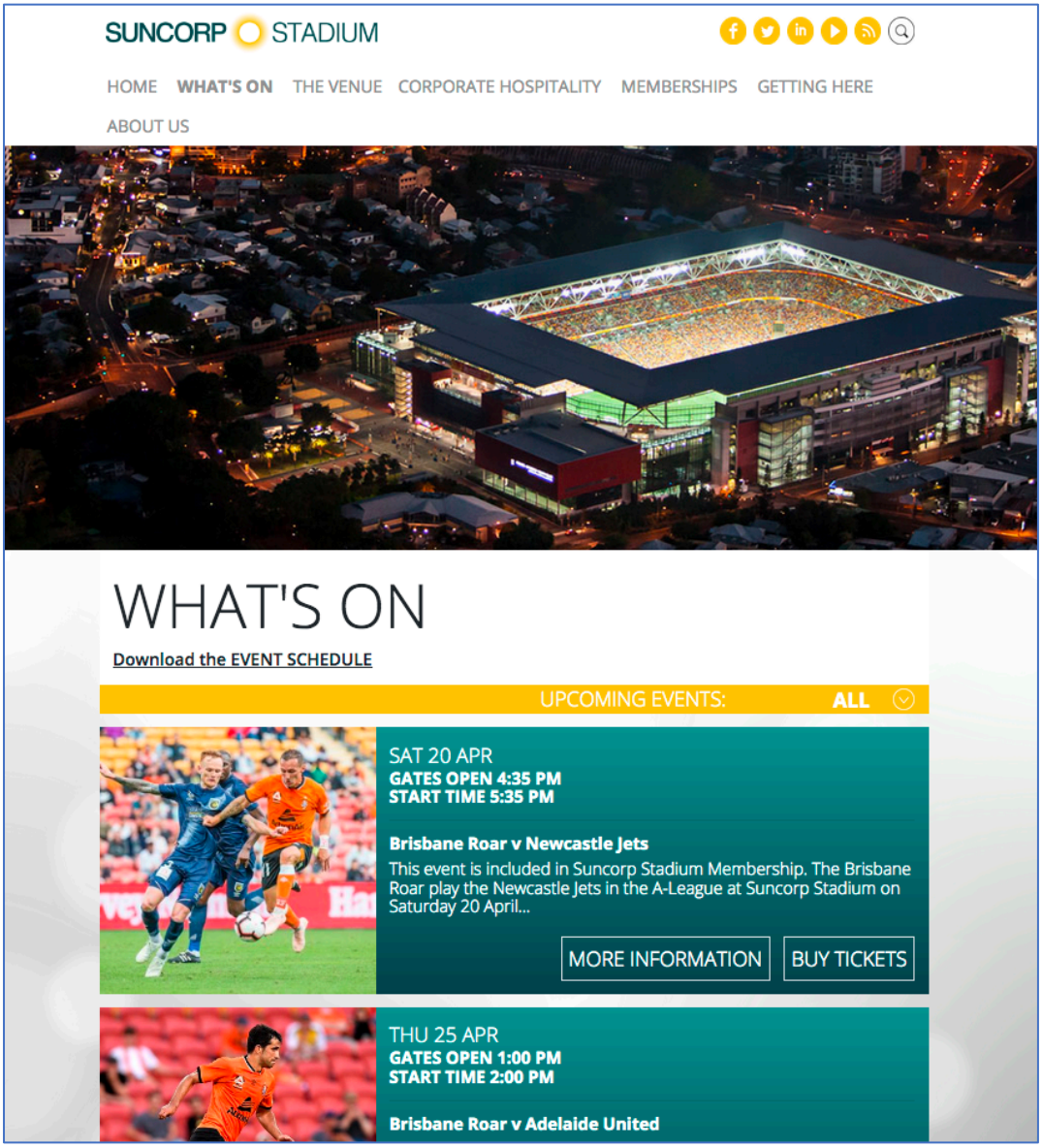
For instance, in our demonstration solution the user can press the buttons in any order they like. If they press the "Print" button before selecting any events the program still works correctly and exports a simple HTML document as follows.



Where the data comes from

A significant challenge for this assignment is that web servers deliver different HTML/XML documents to different web browsers, news readers and other software clients. This means the web document you see in a browser may be different from the web page downloaded by your Python application. For this reason, to create your "archived" files you should download the web documents using our `web_doc_downloader` program, or a similar application. This will ensure that the "online" and "offline" documents have the same format, thus making your pattern matching task easier.

For instance, the Suncorp event site we used appears as follows when we view it in a web browser.



Notice that all the elements we needed for our application appear in the document, including the events' names, dates and images. The challenge is extracting just these elements from the HTML source code.

Most importantly, however, this is not how your Python program "sees" this document. It will receive it as a single, very long character string. For instance, the source code for the above page is actually as shown below when downloaded by a Python script. It is this form of the data that your Python program must work with. From it you will need to use some form of pattern matching to find the textual elements needed to construct the lists to display in your GUI and your exported document. Most importantly, you must do so in a general way that will still work when the contents of the source web page are updated.

Obviously working with such complex code is challenging. You should begin with your static, "archived" documents to get some practice at pattern matching before trying the dynamically changeable web documents downloaded from online.

```
<!DOCTYPE html>
<html  >
<head id="head"><title>
    Suncorp Stadium - What&#39;s On
</title><meta charset="UTF-8" /><script type="text/javascript">window.NREUM||(NREUM={});NREUM.info =
<meta name="viewport" content = "width = device-width, initial-scale = 1.0, minimum-scale = 1, maximu
<meta name="apple-mobile-web-app-title" content="SuncorpStadium" />
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent" />

<!-- Bootstrap CSS -->
<link rel="stylesheet" type="text/css" href="/CMSPages/GetResource.ashx?stylesheetfile=~/SuncorpStad:
<!-- Add the slick-theme.css if you want default styling -->
<link rel="stylesheet" type="text/css" href="/CMSPages/GetResource.ashx?stylesheetfile=~/SuncorpStad:
<!-- Add the slick-theme.css if you want default styling -->
<link rel="stylesheet" type="text/css" href="/CMSPages/GetResource.ashx?stylesheetfile=~/SuncorpStad:

<link href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i,80(
<link href="/SuncorpStadium/css/style.css" rel="stylesheet" type="text/css">

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="/CMSPages/GetResource.ashx?scriptfile=~/SuncorpStadium/js/jquery3.3.1.min.js"></script>
<script src="/CMSPages/GetResource.ashx?scriptfile=~/SuncorpStadium/js/popper.min.js"></script>
<script src="/CMSPages/GetResource.ashx?scriptfile=~/SuncorpStadium/js/bootstrap/bootstrap.min.js"></
<script defer src="https://use.fontawesome.com/releases/v5.0.6/js/all.js"></script>
<script src="/CMSPages/GetResource.ashx?scriptfile=~/SuncorpStadium/js/slick@1.8.1.min.js"></script>

<!--<script src="/CMSPages/GetResource.ashx?scriptfile=/Gabba/js/carousel.js"></script>-->
<script src="/CMSPages/GetResource.ashx?scriptfile=~/SuncorpStadium/js/modernizr.js"></script>
<script src="/SuncorpStadium/js/template.js"></script>
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async src="https://www.googletagmanager.com/gtag/js?id=UA-105618542-7"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());

  gtag('config', 'UA-105618542-7');
</script>
<!-- Facebook Pixel Code -->
<script>
  !function(f,b,e,v,n,t,s)
```

Specific requirements and marking guide

To complete this part of the assignment you are required to produce an application in Python 3 equivalent to that above, using the provided `whats_on.py` template file as your starting point. In addition you must provide a folder containing three previously-downloaded web documents to serve as your archive of old entertainment events and one or more image files needed to support your GUI. (However, all of the images in the exported HTML file must be online images and must *not* be included in your submission.)

Your complete solution must support *at least* the following features.

- **An intuitive Graphical User Interface (5%).** Your application must provide an attractive, easy-to-use GUI which has all the features needed for the user to see, select and export entertainment events. You have a free choice of which Tkinter widgets to use to do the job, as long as they are effective and clear for the user. This interface must have the following features:
    - An image which acts as a "logo" to identify your application. The image file should be included in the same folder as your Python application.
    - Your GUI must name your application in both the Tkinter window's title and as a large heading in the displayed interface. The name may appear as part of the logo (as it does in our demonstration solution), but if the logo does not contain any text you must separately add a textual label to the GUI to give your application a name.

  - A widget or widgets that allows the user to select whether to work online or offline. It must be possible to easily tell which mode is currently selected.
  - A widget or widgets that allows the user to select three different event categories, in either online and offline modes.
  - A widget or widgets that allows the user to select specific events of interest from any of the three categories, offering at least six different events per category.
  - A widget or widgets that allows the user to choose to export their chosen entertainment schedule as an HTML document.

- **Displaying entertainment events in "offline" mode in the GUI (3%).** Your GUI must be capable of displaying three distinct categories of "archived" entertainment events, with at least six events per category. For each category you must show
  - the event type (movies, live music, etc),
  - the name of each event, and
  - the date/time of each event.

  The events must be extracted from HTML/XML files previously downloaded and stored in an "archive" folder. The documents must be stored in exactly the form they were downloaded from the web server; they cannot be edited or modified in any way. Pattern matching must be used to extract the relevant elements from the documents so that the code would still work if the archived documents were replaced with others in the same format. To keep the size of the archive folder manageable only single HTML/XML source files can be stored. No image files may be stored in the archive.

- **Displaying "live" entertainment events in the GUI (4%).** Your GUI must be capable of displaying three distinct "live" entertainment event categories, as currently available online at the time the program is run, with at least six events per category. For each category you must show
  - the event type (movies, live music, etc),
  - the name of each event, and
  - the date/time of each event.

  The events must be extracted from HTML/XML files downloaded from the web when the program is run. Pattern matching must be used to extract the relevant elements from the documents so that the code still works even after the online documents are updated. The chosen source web sites must be ones that are updated on a regular basis, at least weekly and preferably daily.

- **Exporting HTML documents containing events selected by the user (5%).** Your program must be able to generate an "entertainment planner" HTML document containing descriptions of the specific events selected by the user, when working in both "offline" and "live" modes. The schedule exported must be written as an HTML document in the same local folder as your Python program and must be easy to identify through an appropriate choice of file name. Each generated file must contain HTML markups that make its contents easily readable in any standard web browser, and it must be self-contained (i.e., not dependent on any other local files). When viewed in

a browser, the generated document must be neat and well-presented and must contain at least the following features:

- A heading identifying your application.
- An image characterising your application, downloaded from online when the generated HTML document is viewed (i.e., not from a local file on the host computer).
- Details of each of the entertainment events selected by the user in the GUI. For each event at least the following information must be displayed:
  - The event's name.
  - The date/time of the event. (There is no need to standardise the format of the date/time. It can appear in exactly the same format as the source web document. There is also no need to sort the events into chronological order, although this would obviously be a helpful feature in a real application.)
  - Some other distinguishing attribute of the item. The attribute may be a textual description of the event or may be an image.

  All of this information must be extracted via pattern matching from HTML documents downloaded from the web. Most importantly, each of these sets of items *must all belong together*, e.g., you can't have the name of one event paired with an image of another. Each of the elements must be extracted from the original document separately and used to construct your own HTML document.
- Hyperlinks to the three web sites from which the information was extracted. (This will help the markers compare the original web pages with your extracted information).

When viewed in a browser the exported document must be neatly laid out and appear well-presented regardless of the browser window's dimensions. The textual parts extracted from the original documents must not contain any visible HTML tags or entities or any other spurious characters. The images must all be links to images found online, not in local files, should be of a size compatible with the rest of the document, and their aspect ratio should be preserved (i.e., they should not be stretched in one direction).

- **Good Python and HTML code quality and presentation (4%).** Both your Python program code and the generated HTML code must be *presented in a professional manner*. See the coding guidelines in the *IFB104 Code Presentation Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this for Python. In particular, each significant Python or HTML code segment must be *clearly commented* to say what it does, e.g., "Extract the link to the photo", "Show the event's date", etc.
- ***Extra feature (4%).*** *Part B of this assignment will require you to make a 'last-minute extension' to your solution. The instructions for Part B will not be released until just before the final deadline for Assignment 2.*

You can add other features if you wish, as long as you meet these basic requirements. You must complete the task using only basic Python 3 features and the modules already imported into the provided template. **You may not use any Python modules that need to be downloaded and installed separately, such as "Beautiful Soup" or "Pillow". Only modules that are part of a standard Python 3 installation may be used.**

However, your solution is *not* required to follow precisely our example shown above. Instead you are strongly encouraged to *be creative* in your choices of web sites, the design of your Graphical User Interface, and the design of your generated HTML document.

Support tools

To get started on this task you need to download various web pages of your choice and work out how to extract the necessary elements for displaying data in the GUI and generating the HTML output file. You also need to allow for the fact that the contents of the web documents from which you get your data will change regularly, so you cannot hardwire the locations of the elements into your program. Instead you must use Python's string `find` method and/or regular expression `findall` function to extract the necessary elements, no matter where they appear in the HTML/XML source code.

To help you develop your solution, we have included two small Python programs with these instructions.

1. `web_doc_downloader` is a Python program containing a function called `download` that downloads and saves the source code of a web document as a Unicode file, as well as returning the document's contents to the caller as a character string. A copy of this function also appears in the provided program template. You can use it both to save copies of your chosen web documents for storage in your archive, as well as to download "live" web documents in your Python application at run time. Although recommended, you are not required to use this function in your solution, if you prefer to write your own "downloading" code to do the job.
2. `regex_tester` is an interactive program introduced in the lectures and workshops which makes it easy to experiment with different regular expressions on small text segments. You can use this together with downloaded text from the web to help perfect your regular expressions. (There are also many online tools that do the same job you could use instead.)

Portability

An important aspect of software development is to ensure that your solution will work correctly on all computing platforms (or at least as many as possible). For this reason you must complete the assignment using standard Python 3 functions and modules only. You may not import any additional modules or files into your program other than those already imported by the given template file. In particular, **you may not use any Python modules that need to be downloaded and installed separately, such as "Beautiful Soup" or "Pillow". Only modules that are part of a standard Python 3 installation may be used.**

Security warning and plagiarism notice

This is an individual assessment item. All files submitted will be subjected to software plagiarism analysis using the MoSS system (http://theory.stanford.edu/~aiken/moss/). Serious violations of the university's policies regarding plagiarism will be forwarded to the Science and Engineering Faculty's Academic Misconduct Committee for formal prosecution.

As per QUT rules, you are not permitted to copy or share solutions to individual assessment items. In serious plagiarism cases SEF's Academic Misconduct Committee prosecutes both the copier and the original author equally. It is your responsibility to keep your solution secure. In particular, **you must not make your solution visible online via cloud-based code development platforms such as GitHub.** Note that free accounts for such platforms are usually public. If you wish to use such a resource, do so only if you are certain you have a private repository that cannot be seen by anyone else. For instance, university students can apply for a free private repository in GitHub to keep their assignments secure (https://education.github.com/pack). However, we recommend that the best way to avoid being prosecuted for plagiarism is to keep your work well away from the Internet!

Internet ethics: Responsible scraping

The process of automatically extracting data from web documents is sometimes called "scraping". However, in order to protect their intellectual property, and their computational resources, owners of some web sites may not want their data exploited in this way. They will therefore deny access to their web documents by anything other than recognised web browsers such as Firefox, Internet Explorer, etc. Typically in this situation the web server will return a short "access denied" document to your Python script instead of the expected web document (Appendix B).

In this situation it's possible to trick the web server into delivering you the desired document by having your Python script impersonate a standard web browser. To do this you need to change the "user agent" identity enclosed in the request sent to the web server. Instructions for doing so can be found online. We leave it to your own conscience whether or not you wish to do this, but note that this assignment can be completed successfully without resorting to such subterfuge.

Deliverables

You should develop your solution by completing and submitting the provided Python template file `whats_on.py`. Submit this in a "zip" archive containing all the files needed to support your application as follows:

1. Your `whats_on.py` solution. Make sure you have completed the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. **Submissions without a completed statement will be assumed not to be your own work.**
2. One or more *small* image files needed to support your GUI interface, but **no other image files**.
3. A folder containing the previously-downloaded web documents used for your static "archive" of old events. Again, this folder may contain HTML/XML source code

files only. **It must not contain any image files**. All images needed for your exported HTML document must be sourced from online when it is viewed in a web browser.

Once you have completed your solution and have zipped up these items submit them to Blackboard as a single file. **Submit your solution compressed as a "zip" archive. Do not use other compression formats such as "rar" or "7z".**

Apart from working correctly your Python and HTML code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant elements and *helpful* choices of variable and function names. **Professional presentation** of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive **partial marks for incomplete solutions**.

How to submit your solution

A link is available on Blackboard under Assessment for uploading your solution before the deadline (11:59pm Sunday, May 26th, end of Week 12). Note that you can submit as many drafts of your solution as you like. You are strongly encouraged to *submit draft solutions* before the deadline as insurance against computer and network failures. If you are unsure whether or not you have successfully uploaded your file, upload it again!

Students who encounter problems uploading their files to Blackboard should contact the IT Helpdesk (ithelpdesk@qut.edu.au; 3138 4000) for assistance and advice. Teaching staff will not answer email queries on the weekend the assignment is due, so ensure that you have successfully uploaded at least one solution by close-of-business on Friday, May 24th.

Appendix A: Some web sites that may prove helpful

For this assignment you need to find three regularly-updated web sites that list upcoming entertainment events. Each site must contain at least six items at all times, and for each event must include its name, time and some other attribute such as a textual description or an image. This appendix suggests some web sites which *may* be suitable for this assignment, but we don't guarantee that they all are. In particular, we have *not* checked them all to see if

- the lists change frequently enough, at least weekly or preferably daily, and
- they can be successfully downloaded by a Python program.

You should check both of these things before deciding to use any of the web sites below or any others you find yourself.

Radio programmes

- https://www.abc.net.au/radionational/guide/
- https://www.bbc.co.uk/sounds/schedules
- https://www.radiotimes.com/radio/radio-listings/

Video game releases

- https://www.metacritic.com/browse/games/release-date/coming-soon/all/date

Home video releases

- https://www.dvdsreleasedates.com/ Warning: This site may not have six items listed at all times.
- http://www.onvideo.org/weekly/tuesday.htm Note: This site uses the character encoding "windows-1252" instead of "utf-8".

Video streaming release dates

- https://movierewind.com/everything-netflix/ Note: This particular site is complex and looks difficult to use, but there are no doubt many others available in this category.

Broadcast television

- https://www.tvguide.com/tv-premiere-dates/
- https://www.ontvtonight.com/au/guide/listings/BrisbaneNight.html plus many others such as https://www.ontvtonight.com/guide/listings/NewYorkNight.html and https://www.ontvtonight.com/uk/guide/listings/LondonNight.html

Academia

- https://www.qut.edu.au/news/events Warning: This site may not have six items listed at all times.
- http://www.uq.edu.au/events/calendar_view.php?category_id=13 Note: This site uses the character encoding "windows-1252" instead of "utf-8".

Sporting events

- https://www.brc.com.au/calendar

- https://thegabba.com.au/what-s-on.aspx
- https://suncorpstadium.com.au/what-s-on.aspx
- https://qsac.com.au/what-s-on.aspx
- https://cbussuperstadium.com.au/what-s-on.aspx

Multi-purpose venues

- https://www.bcec.com.au/whats-on/
- https://www.brisent.com.au/Event-Calendar
- https://www.brisbaneshowgrounds.com.au/whats-on

Arts and culture

- https://www.museumofbrisbane.com.au/whats-on/
- https://www.queensland.com/en-au/events/arts-and-culture
- http://bneart.com/
- https://www.weekendnotes.com/brisbane/cultural-events/ Note: This site uses the character encoding "windows-1252" instead of "utf-8".
- https://apps.catholic.net.au/assets/xml/events/events.rss

Cinema

- https://www.blueroomcinebar.com/movies/now-showing/
- https://www.dendy.com.au/events/
- https://regaltwin.com.au/now-showing/movie/64/showtime/
- https://fivestarcinemas.com.au/drive-in

Live music

- https://thetivoli.com.au/events
- https://thetriffid.com.au/gigs/page/2/
- http://thezoo.com.au/
- https://www.oldmuseum.org/
- http://ricsbar.com.au/whos-playing/
- http://www.thebrightsidebrisbane.com.au/events-tickets/
- https://www.qso.com.au/events/all-events
- https://www.eatstreetmarkets.com.au/whats_happening

Public events

- https://www.ticketmaster.com.au/Riverstage-Brisbane-tickets-Brisbane/venue/155825 and https://www.ticketmaster.com.au/browse/classical-catid-203/arts-theatre-and-comedy-rid-10002 and https://www.ticketmaster.com.au/browse/comedy-catid-51/arts-theatre-and-comedy-rid-10002

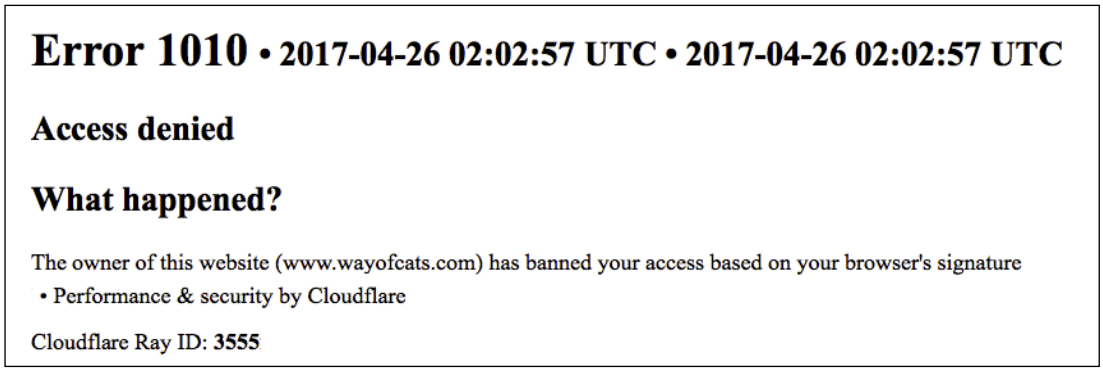
- https://www.eventbrite.com.au/d/australia--brisbane-city/events/ and https://www.eventbrite.com.au/d/australia--brisbane-city/music/ and https://www.eventbrite.com.au/d/australia--brisbane-city/film/ and https://www.eventbrite.com.au/d/australia--brisbane-city/christian/ and https://www.eventbrite.com.au/d/australia--brisbane-city/politics/
- https://brisbane.eventful.com/events/categories/family_fun_kids and https://brisbane.eventful.com/events/categories/comedy and http://brisbane.eventful.com/events/categories/politics_activism
- https://www.qpac.com.au/event/ and others such as https://www.qpac.com.au/search/?listingtype=event&sort=date&term=&daterange=&genre=Comedy
- https://concreteplayground.com/brisbane/events

Finally, while searching for "event" schedules we came across some sites that list upcoming releases of running shoes, such as https://sneakernews.com/release-dates/! We're not sure that this counts as an "entertainment" event!

Appendix B: Web sites that block access to Python scripts

As noted above, some web servers will block access to web documents by Python programs in the belief that they may be malware attempting a "denial of service" attack or in order to protect the web site owner's intellectual property. In this situation they usually return a short HTML document containing an "access denied" message instead of the document requested. This can be very confusing because you can usually view the document without any problems using a standard web browser even though your Python program is delivered something entirely different.

If you suspect that your Python program isn't being allowed to access your chosen web page, use the `web_doc_downloader` program to check whether or not Python programs are being sent an access denied message. When viewed in a web browser, such messages typically look something like the following example. In this case blog `www.wayofcats.com` has used anti-malware application *Cloudflare* to block access to the blog's contents by our Python program.



In this situation you are encouraged to choose another source of data. Although it's possible to trick some web sites into delivering blocked pages to a Python script by changing the "user agent" signature sent to the server in the request we *don't* recommend doing so, partly because this solution is not reliable and partly because it could be considered unethical to deliberately override the web site owner's wishes.