Assignment 2, Part A: Runners-Up

(20%, due 11:59pm Sunday, October 18th, 2020, end of Week 12)

Overview

This is the first part of a two-part assignment. This part is worth 20% of your final grade for IFB104. Part B will be worth a further 5%. 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.

Motivation

In most competitive environments people like to see scorecards which tell them who's currently winning the race. Here you will develop a Python program that provides such lists but instead of focussing on the number one competitor it will emphasise *number two*, the runner-up!

While people normally strive to be number one, there are significant advantages to being in the runner-up's position.

- In 1962 *Avis Rent-a-Car* launched one of the most famous and successful advertising campaigns ever when it announced, "We're #2. We Try Harder." By openly admitting to not being #1, and pointing out that its rival *Hertz* was complacent due to being in the dominant position, Avis dramatically increased its share of the marketplace overnight. Avis' new concern became the very real danger that it would soon overtake Hertz and wouldn't be able to use the slogan any more!
- Arguably, *Microsoft*'s entire global empire was built on coming second! Founder Bill Gates didn't write the MS-DOS operating system, he bought it from the original developers and sold it on to IBM. Microsoft didn't invent Powerpoint, they bought out the small company that created it. The Windows operating system has now duplicated all of the major user interface innovations originally introduced by Apple's MacOS. Letting someone else go first, do all the work, and take all the risks, can be a winning strategy!

The runner-up's position is thus a good one. A runner-up is poised to take the lead but is protected from the risks associated with being out front.

Your Python program will thus emphasise the runner-up's strategic position in a set of competitive lists. It will have a Graphical User Interface that allows its user to see the current runner-up in several lists, as well as the other competitors, and it will extract its data from online sources so that it is always up to date.

This "capstone" assignment is designed to incorporate all of the concepts taught in IFB104. To complete it you will need to use Tkinter to create an application with an interactive Graphical User Interface and use pattern matching to extract specific elements from HTML documents downloaded from the Internet.

Goal

In this assignment you will develop an interactive program which lets its users select from several ranked lists of competitors to see who is the current runner-up in each list, as well as seeing the other competitors in each list. The lists are extracted from regularly-updated online web documents, so the user can be confident that the data is current. If desired, the user must also be able to inspect the original web documents from which the data was extracted.

There must be at least four distinct lists, three of which are derived from online sources which are updated frequently. The fourth list can be extracted from a previously-downloaded web document which never changes. (This final list will help you develop your solution by using an unchanging web document with known contents.)

For the purposes of this assignment you have a free choice of which lists your application will access, provided they always contain at least ten competitors, are updated frequently, and include at least one distinctive property of each competitor apart from its identity and position. Your application must offer access to at least three entirely different online lists, e.g.:

- music charts,
- movie or television ratings,
- stock market listings,
- online gaming player rankings,
- book rankings,
- crowd-sourced popularity lists,
- customer ratings of products or services,
- web site statistics,
- etc.

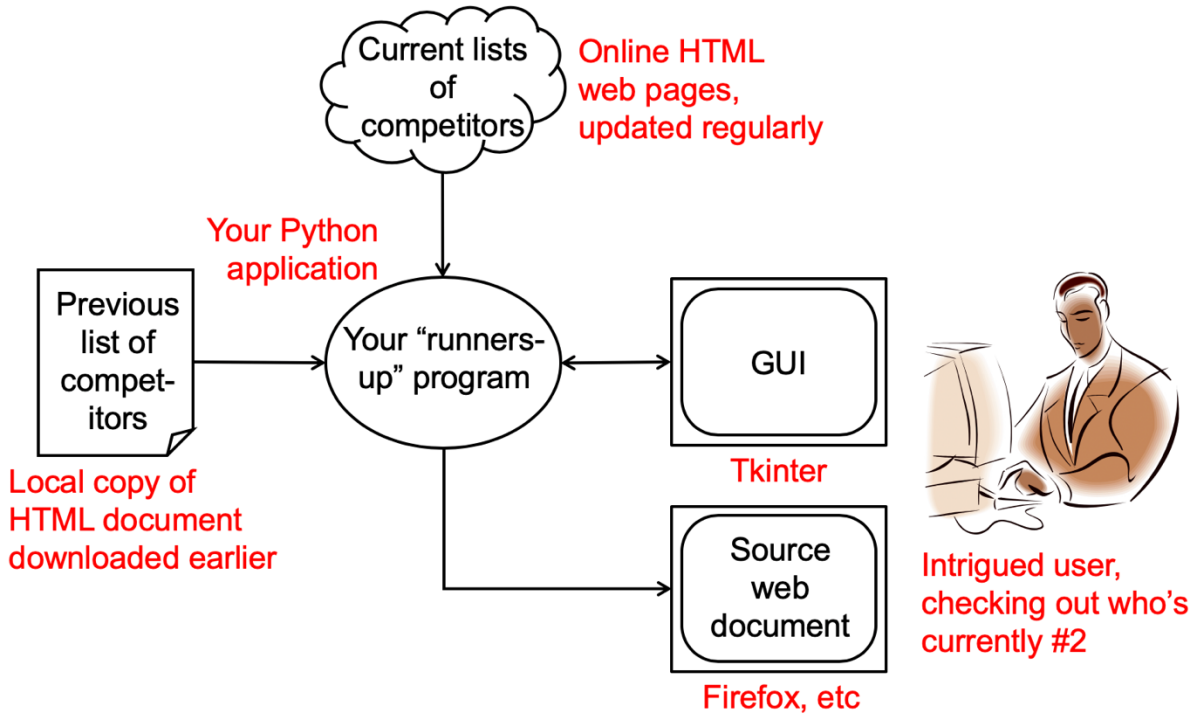
However, whatever lists you choose, you must confirm that the online web documents are updated regularly.

For each competitor in each list the source web site must contain the competitor's identity and some other distinguishing property (other than its position). Properties could be:

- a book's author or publisher,
- a movie's lead actor or director,
- a music album's artist or publisher,
- a textual description of the competitor,
- a numeric score, especially one which explains the competitor's position in the list,
- etc.

Appendix A below lists many web sites which *may* be suitable for this assignment, but you will need to confirm their suitability, and you are encouraged to find your own of personal interest instead of relying on our suggestions.

Using such web sites as your data source, you are required to build an IT system with the following general architecture.



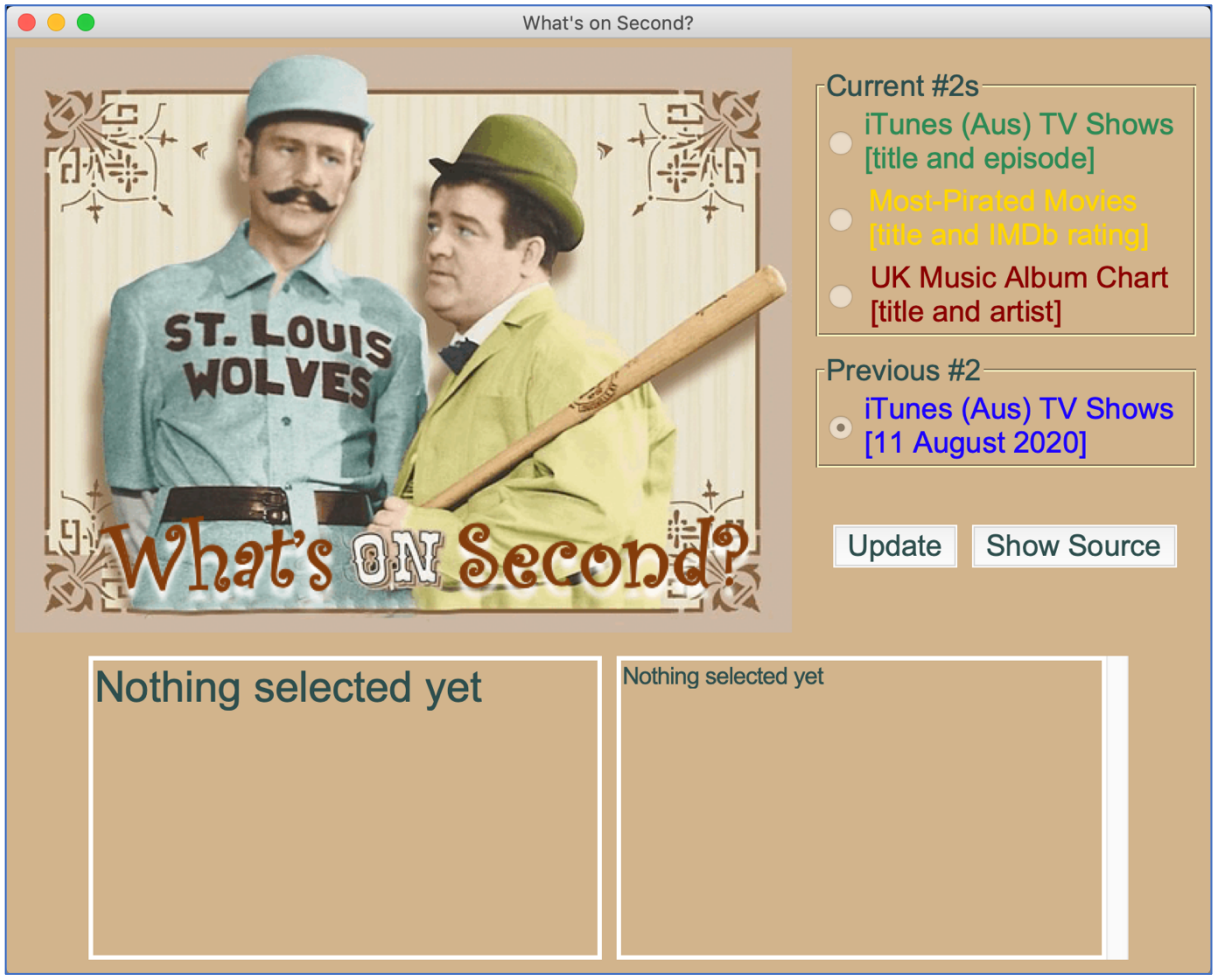
Your solution will be a Python program with a Graphical User Interface. Under the user's control, it extracts ranked lists of competitors from four distinct web documents. One of the web documents is stored locally in the same folder as the Python script and never changes. This previously-downloaded list can be used to help perfect your data extraction code on an unchanging source. The other web documents are all downloaded "live" from the Internet whenever you run your progran. In this case the web documents' contents will likely be different each time you run your code. This produces more interesting data for the user but is obviously more challenging for you to process.

Having extracted a list of competitors from one of these sources, your program must prominently display the competitor in the number two position as the current "runner-up". In addition, the other nine competitors at the top of the list must be displayed separately, with a clear indication of their position in the list. All ten competitors displayed must be accompanied by some additional characteristic property, also extracted from the source web document.

As well as viewing the extracted data in your program's GUI, the user must also be given the ability to view the original web documents from which the data was extracted. For this purpose your program must allow the user to open the source HTML document in their computing platform's default web browser. (As well as giving the user a chance to find out more about the competitors, this helps you confirm that you have extracted the data correctly.)

Illustrative example

To demonstrate the idea, below is our own "runners-up" application, which uses data extracted from four different web documents. The screenshot below shows our solution's GUI when it first starts. We've called our application *What's on Second?* in homage to the classic "Who's on First?" vaudeville sketch popularised by Abbott and Costello, and have included a relevant image to serve as the application's logo.



The GUI offers the user four distinct lists of competitors. One list, the most popular TV Shows viewed on iTunes Australia as of August 11th, is static and is never updated. The other three categories, the current most-popular iTunes Australia TV shows, the current movies most-pirated internationally on torrent sites, and the current UK music album chart, are all "live" and represent the contents of these lists right now. These three lists may contain different data each time we check them because all three web sites we use as data sources are updated on a regular basis.

The user can select a list by clicking on the corresponding radio button. For instance, the user may choose to look at the static list of TV shows first, by selecting it and pressing the "Update" button. This causes the names of the TV shows and the specific episodes to be shown in the GUI as follows, reflecting the state of this list on August 11th:

Given our interest in who's coming second, the runner-up is displayed prominently on its own at the bottom left. In this case the runner-up was an episode of the TV show *The Real Housewives of Beverly Hills* called *There's No Place Like Rome*.

The other nine competitors are listed separately. All of these competitors must be clearly numbered to show their position in the list, given that the second one does not appear in the list on the right. Thus, the first competitor was the opening episode of the new *Perry Mason* series, the third competitor was the pilot episode of *Euphoria*, and so on.

Also note that two episodes of *War of the Worlds* appear on the right. In this list the same competitor can appear more than once, which is why it's essential to include another property, in this case the episode name, to uniquely distinguish the entries.

Apart from this static list, the user can also see who is *currently* runner-up in the other three lists by selecting the corresponding "live" option, e.g., the current UK music album chart:

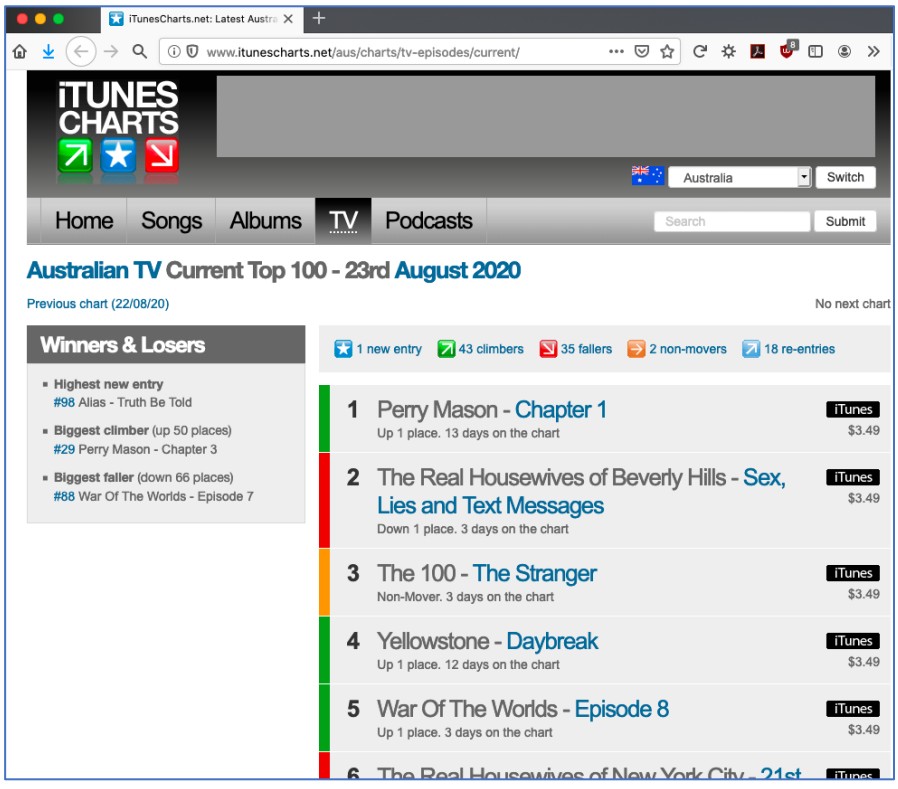
The screenshot above was taken when we ran our program on August 24th (the final three entries in the right-hand list could be seen by scrolling down). We thus discovered that Taylor Swift was the current runner-up on the official UK music albums chart at that time. For this category our program lists each album's title and artist. The extracted text is all in upper case because this is the way it appeared on the source web site. Our program simply copies the text verbatim from the original web document, although we are careful to remove any unusual Unicode characters or HTML entities that would not display properly in our Tkinter GUI.

Our application also gives the user the option of seeing the current list of iTunes TV shows, rather than the previously-saved one. In this case it produced the following results on August 24th:

We thus learnt that the first episode of the new *Perry Mason* series was still the most popular show, but that a different episode of *The Real Housewives of Beverly Hills* now holds the coveted runner-up position. Several other changes since we copied the web site on August 11th can be seen further down the list. (The tenth show could be seen by scrolling down the list on the right.)

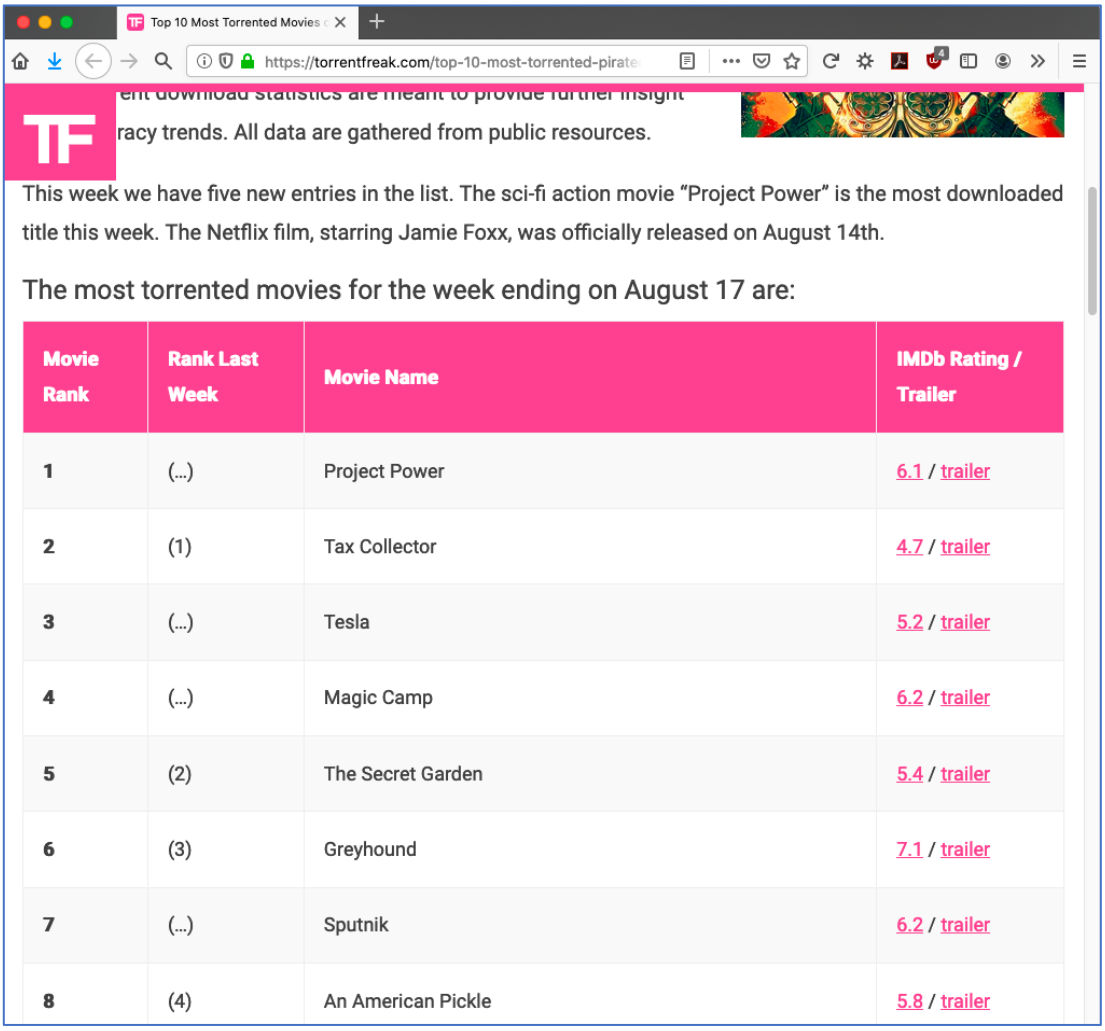
Finally, selecting the second radio button and pressing "Update" allows us to see which movies are currently being pirated most frequently on torrent sites. When we did this on August 24th the runner-up was *Tax Collector*. Each of the ten entries shows the movie's title and its user rating on the widely-used IMDb rankings.

As well as showing a curated version of the data, which emphasises the runner-up, your program must also make it easy for the user to inspect the original web documents, for all four data sources. In our sample solution this is done by selecting an option via the radio buttons and pressing "Show Source". Doing so for the three "live" sources causes the original web site to be opened in our computer's default web browser. For instance, selecting the current TV shows list and pressing "Show Source" causes the original web site from which we extracted our data to appear in our web browser:



This allows us to see more information than we showed in the GUI (and to check the accuracy of our program!).

Similarly, choosing the pirated movies option and pressing "Show Source" opens the original TorrentFreak site, confirming that Tax Collector was the runner-up when we ran our program, and that it had an IMDb rating of 4.7:



Movie Rank	Rank Last Week	Movie Name	IMDb Rating / Trailer
1	(...)	Project Power	6.1 / trailer
2	(1)	Tax Collector	4.7 / trailer
3	(...)	Tesla	5.2 / trailer
4	(...)	Magic Camp	6.2 / trailer
5	(2)	The Secret Garden	5.4 / trailer
6	(3)	Greyhound	7.1 / trailer
7	(...)	Sputnik	6.2 / trailer
8	(4)	An American Pickle	5.8 / trailer

The same thing can be done for the previously-downloaded web document, in this case an earlier copy of the iTunes TV shows list. Selecting the "Previous #2" option and pressing "Show Source" causes the copy of the web site we stored locally to be displayed in our web browser, as shown below. However, when we do so, the web document is not properly formatted, and looks quite different than the "live" site shown above:

Australian TV Current Top 100 - 11th August 2020

- Previous Australian TV Episode chart (10/08/20)
- No next chart

Winners & Losers

- **Highest new entry**

  #1 Perry Mason - Chapter 1

- **Biggest climber (up 55 places)**

  #39 Succession - Sh*t Show at the F**k Factory

- **Biggest faller (down 51 places)**

  #96 MotherFatherSon - Episode 5

- 6 new entries
- 27 climbers
- 41 fallers
- 4 non-movers
- 21 re-entries

- 1 Perry Mason - Chapter 1

  Brand new entry at #1!

  buy from iTunes $3.49

- 2 The Real Housewives of Beverly Hills - There's No Place like Rome

However, this is to be expected because we only downloaded the main HTML file and not any CSS style files needed to support it. Although you must provide the ability to see the previously-downloaded web document in a browser, there is no need to preserve its original formatting. The unformatted document above is still sufficient to confirm who was runner-up.

In summary, therefore, our demonstration program provides its users with a Graphical User Interface providing the following capabilities:

- The ability to clearly identify the application, including its name and logo.
- The ability to select from at least four distinct lists of competitors, one previously-downloaded and three "live" ones which are downloaded whenever the program runs.
- The ability to see the runner-up in any of the selected lists, including the competitor's identity and some additional property of that competitor.
- The ability to see the other top nine competitors in any of the selected lists, including each competitor's identity, its additional property and its position in the original list.
- The ability to view the source web documents for any of the four lists in the host operating system's web browser.

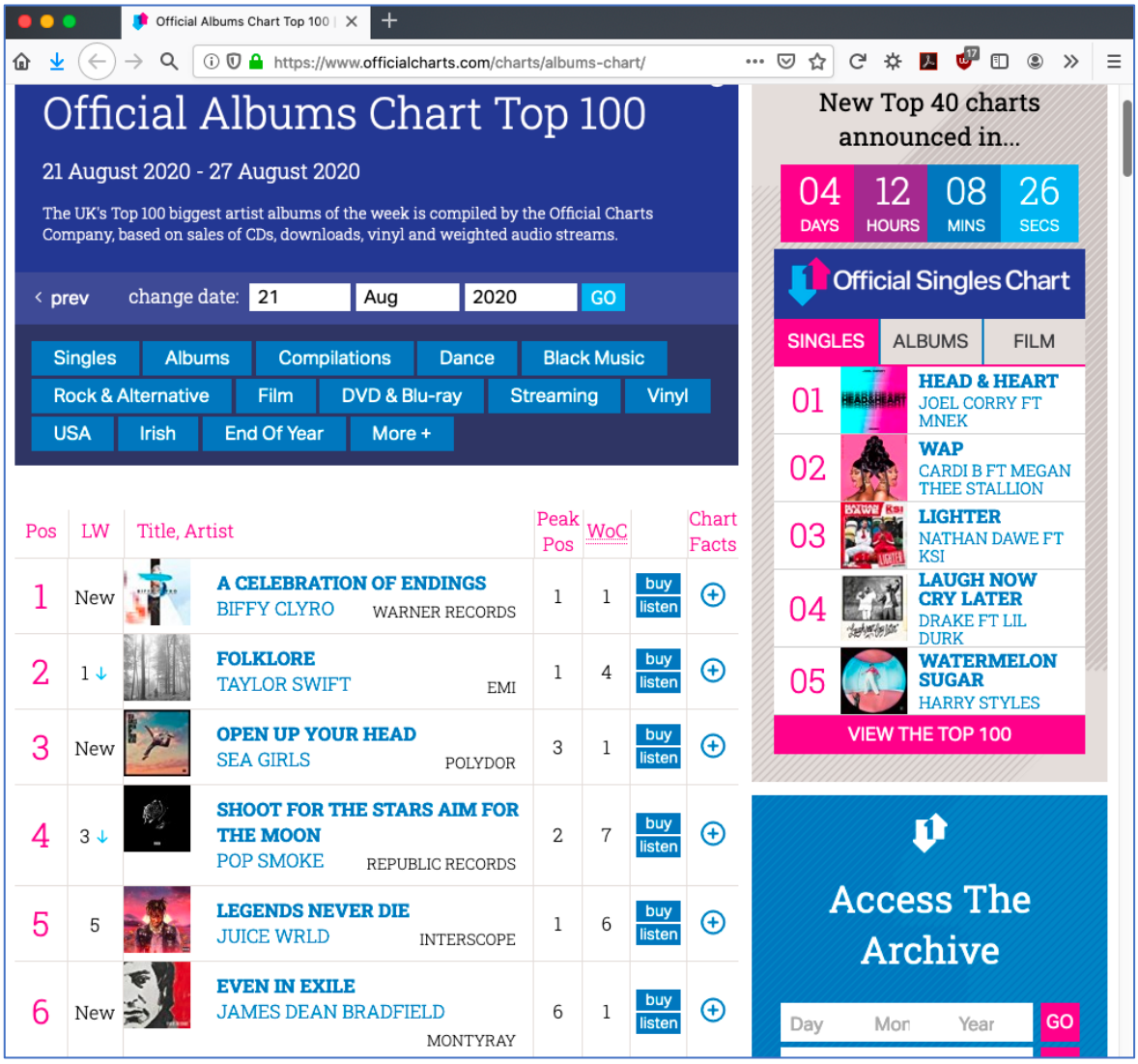
You are *not* required to follow the details of our demonstration program. You are strongly encouraged to use your own skills and initiative to devise your own solution, provided it has all of the capabilities listed above. For instance, different widgets could be used for the GUI,

such as press buttons or pull-down menus for selecting lists. Similarly, our example uses text boxes of fixed size to display the results, but this can be inconvenient because sometimes the user needs to scroll down to see all the text. A possible improvement would be to display the information in a widget which resizes itself so that all the text can be seen at once.

Where the data comes from

To produce the data for display our Python program uses regular expressions to extract elements from the relevant web documents, whether they are stored as a local file or downloaded when the program is run.

We began by using the `downloader.py` program (see below) to download copies of the web documents from the relevant sites and we then examined their HTML source code to determine how the elements we needed are marked up. For instance, the UK music albums chart appears as shown below when viewed in a web browser. It contains information about 100 albums and their artists, so our challenge was to extract this data for just the first ten of these, ignoring all the many other elements of the web page.

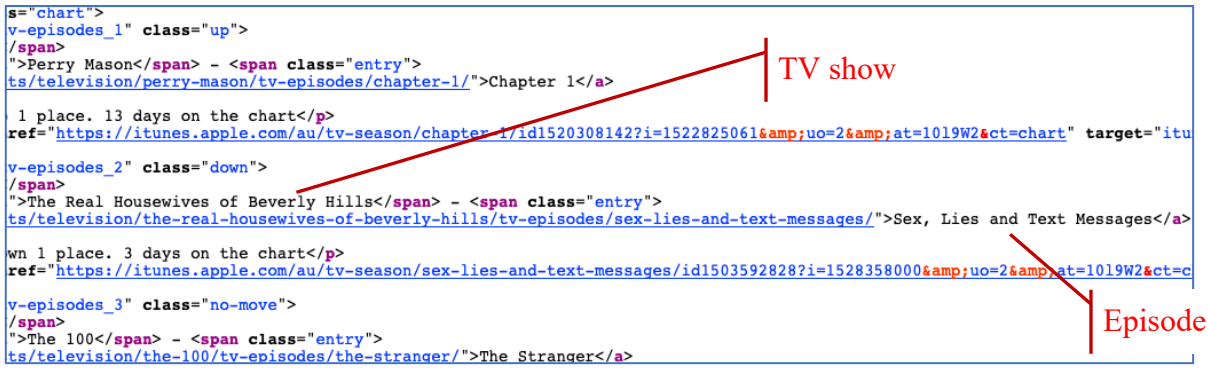


To do this we inspected the HTML source code to see how the elements of interest are marked up. For example, we found the name of the album and artist for the runner-up embedded in the source code as follows.


```
722
723                 <span class="last-week icon-down">
724                     1
725                 </span>
726             </td>
727
728             <td>
729                 <div class="track">
730                     <div class="cover" style="height:60px;width:60px">
731                         <img src="https://d35iaml2i6ojwd.cloudfront.net/img/small?url=/image
732                     </div>
733                     <div class="title-artist">
734                         <div class="title">
735                             <a href="/search/albums/folklore/">FOLKLORE</a>
736                         </div>
737                         <div class="artist">
738
739                             <a href="/artist/5387/taylor-swift/">TAYLOR SWIFT</a>
740                         </div>
741                         <div class="label-cat"><span class="label">EMI</span></div>
742                     </div>
743                 </div>
744             </td>
745
746                 <!-- Peak Position-->
747                 <td>1</td>
748                 <!-- Wks -->
749                 <td>4</td>
750                         <td>
751                     <div class="actions">
752                         <a href="" data-toggle="actions-view-buy-2">Buy</a>
753
```

We thus discovered that the album titles and artist's name all appear within `<a href="..."> ... </span>` HTML tags, which made them easy to find with appropriate regular expressions and Python's `findall` function.

Similarly, as shown below, in the iTunes TV site we found the names of the TV shows and the episodes appear within `<span ...>` and `<a ...>` tags, respectively, so they could also be found with corresponding regular expressions.




```
s="chart">
v-episodes_1" class="up">
/span>
">Perry Mason</span> - <span class="entry">
ts/television/perry-mason/tv-episodes/chapter-1/">Chapter 1</a>

 1 place. 13 days on the chart</p>
ref="https://itunes.apple.com/au/tv-season/chapter-1/id1520308142?i=1522825061&amp;uo=2&amp;at=1019W2&ct=chart" target="itu

v-episodes_2" class="down">
/span>
">The Real Housewives of Beverly Hills</span> - <span class="entry">
ts/television/the-real-housewives-of-beverly-hills/tv-episodes/sex-lies-and-text-messages/">Sex, Lies and Text Messages</a>

wn 1 place. 3 days on the chart</p>
ref="https://itunes.apple.com/au/tv-season/sex-lies-and-text-messages/id1503592828?i=1528358000&amp;uo=2&amp;at=1019W2&ct=c

v-episodes_3" class="no-move">
/span>
">The 100</span> - <span class="entry">
ts/television/the-100/tv-episodes/the-stranger/">The Stranger</a>
```

For the most-pirated movies site we discovered that the movies' names and their IMDb ratings appeared in a table with hyperlinks to the IMBd web site, so paying careful attention to the corresponding `<td>` and `<a ...>` tags was the key to devising regular expressions to extract them in this case. (We also discovered that this site is not very reliable, and often contains formatting changes and even obvious errors in its HTML source code, so we don't recommend using it.)

```
<td><strong>1</strong></td>
<td>(&#8230;)</td>
<td>Project Power</td>
<td><a href="https://www.imdb.com/title/tt7550000/">6.1</a> / <a href="https://www
</tr>
<tr>
<td><strong>2</strong></td>
<td>(1)</td>
<td>Tax Collector</td>
<td><a href="https://www.imdb.com/title/tt8461224/">4.7</a> / <a href="https://www
</tr>
<tr>
<td><strong>3</strong></td>
<td>(&#8230;)</td>
<td>Tesla</td>
<td><a href="https://www.imdb.com/title/tt5259822/">5.2</a> / <a href="https://www
</tr>
<tr>
<td><strong>4</strong></td>
<td>(&#8230;)</td>
<td>Magic Camp</td>
<td><a href="https://www.imdb.com/title/tt3979300/">6.2</a> / <a href="https://www
</tr>
<tr>
<td><strong>5</strong></td>
```

Movie

IMDb rating

Obviously working with complex HTML code like this is challenging. You should begin with your static, unchanging document to get some practice at pattern matching before trying dynamically changeable web documents.

Robustness

Another important aspect of your solution is that it must be resilient to error. The biggest risk with this kind of program is some problem accessing the source web sites. We have attempted to make our `download` function as robust as possible. In particular, if it detects an error while downloading a web document it returns the special value `None` instead of a character string, so your program must check for this. We don't claim that the `download` function is infallible, however, because the results it produces are dependent on the behaviour of your specific Internet connection. For instance, some systems will generate a default web document when an online site can't be reached, in which case the `download` function will be unaware that a failure has occurred and won't return the special value.

As an example of resilience, if our demonstration solution tries to access a "live" web site when there is no Internet connection, it simply displays error messages, as follows. Importantly, however, the program does *not* generate any red exception error messages in the shell window.

Furthermore, as insurance against the risk of a web site failing completely, your program's three "live" web sources must come from entirely different web servers. One way of achieving this is to ensure that the parts of the address at the beginning of each site's URL are all distinct. For example, our sample solution used three distinct sources for its runner-up data, *iTunes*, *TorrentFreak* and the *BBC*'s official music charts. These three sites have the following URLs and clearly come from different web servers as shown by the parts highlighted in red.

```
http://www.itunescharts.net/aus/charts/tv-episodes...
https://torrentfreak.com/top-10-most-torrented-...
http://www.officialcharts.com/charts/albums-chart/
```

Another source of potential problems in IT systems is user error. This depends on the design of your GUI. Your program should anticipate any sequence of user actions and respond appropriately. For instance, in our solution if the user presses the "Update" or "Show Sources" buttons before selecting a list using the radio buttons, a default document source is used, which in this case is the previously-dowloaded iTunes TV shows data. Your program should never produce unhandled exception errors in the Python interpreter's shell window.

Specific requirements and marking guide

To complete this task you are required to produce a Python program with functionality equivalent to that described above, using the provided `runners_up.py` template file as your starting point. In addition you must provide a previously-downloaded web document as your source of "previous runner-up" data and (at least) one image file to display as your app's logo in the GUI.

Your complete solution must support at least the following features. Marks available are as shown.

- **An intuitive Graphical User Interface (5%).** Your application must provide an attractive, easy-to-use GUI. You have a free choice of which Tkinter widgets 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 identifies the application. Note that many Tkinter implementations only support GIF images, so we recommend using that format. The image file should be included in the same folder as your Python application.
    - A heading which names your application. This can be part of the above image or a separate widget. Either way, you must also name your app in the Tkinter window's title.
    - A widget or widgets that allow the user to select from four runners-up lists, three "live" and one previously-downloaded. It must be obvious in the GUI which are the "live" lists and which is the unchanging one. It must also be obvious what properties are displayed about each of the competitors, e.g., their identity, author, artist, score, etc, in addition to their position in the list.
    - Widgets for displaying the runner-up and the other competitors from each list selected.

- A widget or widgets to allow the user to open any of the four source HTML document's in their computer's default web browser.

Note that this criterion concerns the front-end GUI elements only. The functionality of these elements is covered in the following criteria.

- **Displaying data from the previously-downloaded document (2%).** Whenever the user chooses to do so your program must display the runner-up from the unchanging list and the other nine competitors at the top of the list. The runner-up must be displayed separately from the others and must include its identity and whatever other "property" is associated with each competitor. The remaining nine competitors must also be displayed, including their identity, associated property and their position in the list. The data must be extracted using pattern matching techniques from the previously-downloaded HTML document so that if the source document was replaced with a different one from the same web site your program would still work. The competitors' identities, their associated properties and their positions in the list must align correctly as per the original document. No unintended HTML mark-ups or odd characters should be displayed in the GUI.
- **Displaying data from online documents (6%).** Whenever the user chooses to do so your program must display the runner-up and the other nine competitors currently at the top of each of the three "live" lists. The data must be extracted from copies of the source web pages downloaded whenever your Python program runs, so that it is always up to date. The runner-up must be displayed separately from the others and must include its identity and whatever other "property" is associated with each competitor. The remaining nine competitors must also be displayed, including their identity, associated property and their position in the list. The competitors' identities, their associated properties and their positions in the list must align correctly as per the data currently in the source web site. No unintended HTML mark-ups or odd characters may be displayed in the GUI. If there is some problem accessing the web sites your program must display an appropriate error message; no unhandled exceptions may appear in the interpreter's shell window.
- **Opening the source web documents in a web browser (3%).** Your program must give the user the ability to select and view the source web documents for any of the four runners-up lists in their computer's default web browser.
- **Code quality and presentation (4%)**. Your program code, for both Parts A and B of the assignment, 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. Each significant code block must be clearly commented to describe its *purpose* (but not *how* it works, which can be seen from the code itself). Similarly, the names of your functions, parameters and variables should be indicative of their role, not just "`i`", "`j`", etc. Also, you should use function definitions and loops to avoid unnecessary duplication of similar or identical code segments. All instructions in the provided Python template must be followed. Finally, to get full marks for this criterion you must provide a significant amount of code to assess; a few lines of well-presented code will not be considered sufficient.

- ***Extra feature (5%).*** *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 functions 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 precisely duplicate our example shown above. Instead you are strongly encouraged to *be creative* in your choices of web sites to access, and the design and operation of your Graphical User Interface.

Support tools

To get started on this task you need to download various web documents of your choice and work out how to extract the necessary elements from them. 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 these 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 source code.

To help you do this, we have included two small Python programs with these instructions, as well as some helpful features in the Python template.

1. `downloader.py` 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 a copy of your "static" web document, as well as to download the "live" web documents in your Python program. 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.py` 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 sites to help perfect your regular expressions. (There are also many online tools that do the same job.) When copying regular expressions from such a "tester" into your Python program keep in mind that certain characters are meaningful in both regular expressions and Python strings, especially backslashes "\". If this causes problems the easiest solution is to precede the character string with "r", e.g., using expression "`r'[a-z]\n'`" to match a lower case letter at the end of a line. The "r" tells the Python interpreter to treat the following character string as uninterpreted "raw" text and to ignore the backslash.
3. Python's standard `webbrowser` module supports opening of online documents in the host operating system's default web browser, given their URL. Our Python template file imports the necessary function and names it "`urldisplay`" (to clearly distinguish it from the built-in "`open`" function for local files).

4. However, Python's `webbrowser` module does not reliably support opening of local files in a web browser. The Python documentation says that this capability is available only in certain Python implementations. The assignment's template file therefore includes a function of our own, called `open_html_file`, which allows HTML files to be opened in the default web browser, provided they are stored in the same folder as the Python program. The function has been tested on selected Apple, Microsoft and Linux platforms, although it comes "as is", with no guarantees!

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 browser software such as Firefox, Chrome, etc. Typically in this situation the web server will return a short "access denied" document to your Python program instead of the expected web document (see Appendix B).

In this situation it's sometimes possible to trick the web server into delivering 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. Our `download` function has an option for doing this, but we make no guarantees that it will always work. We leave it to your own conscience whether or not you wish to use such an approach, but note that the assignment can be completed successfully without resorting to such subterfuge.

Development hints

This "capstone" assignment is a substantial task, so you should not attempt to do it all at once. In particular, you should approach it methodically and attempt one part at a time.

- Developing a way of extracting relevant elements from web documents is a challenge in its own right. Having selected the web sites of interest you should download copies of the relevant web documents so that you can study their structure. You should download the documents using the supplied `download` function, rather than saving them from a web browser, to ensure that they have the same structure that will be seen by your Python program. Examine the HTML source code of the documents to determine how the elements you want to extract are marked up. This can be done using a plain text editor or a web browser's source code inspection feature. Typically you will want to identify the markup tags that uniquely indicate the beginning and end of the text you want to extract. Using the provided `regex_tester.py` application, you can then devise regular expressions which extract just the necessary elements from the web documents' source code. Having perfected the regular expressions you can then develop a simple prototype of your "back end" Python function(s) that extracts and returns the required elements from a web document.
- The assignment has been designed so that you have to work with a previously-downloaded, unchanging web document, and "live" web documents downloaded at run time. Obviously it is easiest to work with the static document, so you should develop your code for that part of the assignment first, before tackling "live" web pages.

- To complete your program you need to develop both a GUI front end for user interaction and back-end functions for extracting data from web documents. These tasks can be completed in parallel, so if you get stuck on one turn your attention to the other. Developing an attractive GUI can be the fiddliest step, so don't worry about making it look nice until you have the core functionality of your program working.

If you are unable to complete the whole task, just submit those stages you can get working. You will receive **partial marks** for incomplete solutions. However, if your solution is only partially working it will help the marker if you explain the limitations of your submission, either by adding comments when you upload it to Blackboard or with comments in the submission itself. Ensure that your program runs as submitted, even if its functionality is incomplete.

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 modules and functions 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 and are thus unsafe. 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 both the Internet *and your fellow students*!

Deliverables

You must develop your solution by completing and submitting the provided Python template file `runners_up.py`. **Your solution may not rely on any Python modules that need to be downloaded and installed separately, such as "Beautiful Soup". Programs that do not run in a standard Python 3 environment will receive 0%.** Submit your Python code in a "zip" archive containing all the files needed to support it as follows:

1. Your `runners_up.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. Copies of the previously-downloaded web document used for your static "previous runner-up" list. Include only a single HTML document. **Do not include any image or style files associated with the web document**. Only the HTML source code is needed by your Python program.

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 parts 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, October 18th, 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 assignment files to Blackboard should contact *HiQ's Technology Services* (http://qut.to/ithelp; askqut@qut.edu.au; 3138 2000) for assistance and advice. **Do not email assignments to the teaching staff.** They cannot be accepted for marking because we cannot upload assignments to Blackboard on your behalf. Also, 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, October 16th.

Appendix A: Some web sites that may prove helpful

For this assignment you need to find three or more regularly-updated online lists, containing at least ten items at all times. 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 at least weekly, 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.

The following sites *appear* to be updated frequently, so *may* be suitable for this assignment. Note that it's not enough to take the publisher's word. Just because a site says "updated daily" doesn't mean it will *change* daily!

- The most-played tracks on Soundcloud: https://soundcloud.com/charts/top
- Spotify Charts, Australia: https://spotifycharts.com/regional/au/daily/latest
- Box Office Mojo's Daily Box Office Receipts: https://www.boxofficemojo.com/daily/chart/
- Players' Rankings for Steam: http://steamcharts.com/top
- US TV Guide's Trending TV shows: https://www.tvguide.com/trending-tonight/
- Top 100 Stocks to Buy: https://www.barchart.com/stocks/top-100-stocks
  and Stock Price Surprises: https://www.barchart.com/stocks/price-surprises
  plus several other promising looking lists at the same site
- Publisher's Weekly Top 10 Books: http://www.publishersweekly.com/pw/nielsen/top100.html
- Rotten Tomatoes Top Box Office: https://www.rottentomatoes.com/browse/in-theaters/
  and Top DVD and Streaming: https://www.rottentomatoes.com/browse/top-dvd-streaming
  plus several other promising looking lists at the same site
- Apple iTunes Charts: http://www.itunescharts.net
  including several lists such as TV Charts, Australia:
  http://www.itunescharts.net/aus/charts/tv-episodes/current/ (as used in our sample solution)
- Official UK Music Charts: http://www.officialcharts.com/charts
  including the Albums Chart: http://www.officialcharts.com/charts/albums-chart/ (as used in our sample solution)
- TorrentFreak's Most-Pirated Movies list: https://torrentfreak.com/top-10-most-torrented-pirated-movies/ (as used in our sample solution, but with some difficulty due to the site's inconsistency)
- Google Real-Time Search Trends, Australia: https://trends.google.com/trends/trendingsearches/realtime?geo=AU

- Ultimate Guitar's Top Songs: https://www.ultimate-guitar.com/top/tabs?type=all&order=hitsdaily_desc
- YouTube Trending Videos: https://www.youtube.com/feed/trending
- American Top 40 Songs: https://www.at40.com/charts/top-40-238/latest/
- Billboard Music Charts: https://www.billboard.com/charts
  e.g., Pop Songs: https://www.billboard.com/charts/pop-songs
- The Movie Times Top 10 Netflix Movies: https://www.the-movie-times.net/?p=410
- New Zealand Music Charts, Singles: https://www.nztop40.co.nz/chart/singles
  and Albums: https://www.nztop40.co.nz/chart/albums
- MYX Music Chart Daily Top Ten: https://myx.abs-cbn.com/charts/myx-daily-top-10-pinoy/
- Australian Music Charts, Singles: https://australian-charts.com/weekchart.asp?cat=s
  and Albums: https://australian-charts.com/weekchart.asp?cat=a
- IMDb US Movie Box Office Receipts: https://www.imdb.com/chart/boxoffice
- Big Top 40 Music Chart: https://www.bigtop40.com/
- Aria Music Charts: http://www.ariacharts.com.au/charts
  with several to choose from
- Event Hubs Video Game Tiers: https://www.eventhubs.com/tiers/ (but it's not clear how often these update)
- MyAnimeList's Anime Listings: https://myanimelist.net/
  with many to choose from, e.g., Top Airing: https://myanimelist.net/topanime.php?type=airing

In addition there are many sites that provide very large numbers of lists, but their update frequency is unclear, so they are probably *not* suitable for the assignment:

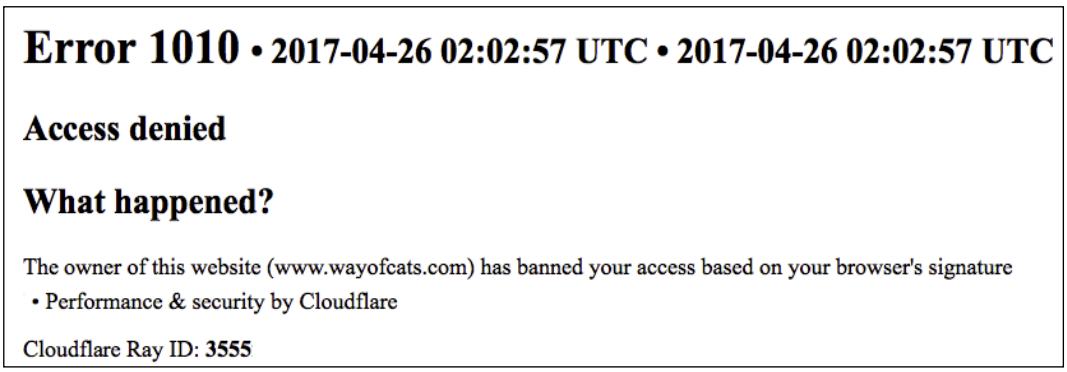
- Official World Golf Ranking: http://www.owgr.com/ranking (we generally suggest avoiding sports rankings, because they may not update at all out of season)
- The Ranker site has hundreds of crowd-sourced popularity lists, but it's not clear how often each is updated: https://www.ranker.com/
  If you're thinking of using one of these lists, first confirm that it changes frequently.
- GoodReads Trending Books: https://www.goodreads.com/shelf/show/trending
  but no indication of update frequency
- List Challenges: http://www.listchallenges.com
  has a huge number of crowdsourced lists but with an unclear update frequency
- Wonderlist: https://www.wonderslist.com
  has a huge number of lists but they don't look like they are updated regularly

Before committing time to working on a web page, make sure that the list it contains is updated frequently.

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 "bots" or malware, or in order to protect the owner's computing resources and data assets from abuse. In this situation they usually return a short HTML document containing an "access denied" message instead of the desired document. 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 by the server.

If you suspect that your Python program isn't being allowed to access your chosen web page, use our `downloader.py` program to check whether or not your Python program is 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 its 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 website owner's wishes.