

Take-Home Task 2: The Best, Then and Now

(10%, deadline 11:59pm Sunday, January 26, end of Week 12)

Overview

This is the first part of a two-part assignment. This part is worth 10% of your final grade for ITD104. . The In-Class Test 2 (ICT2) is the second part, and will be worth a further 15%. The ICT will test you for the same skills required to complete this THT.

Motivation

People are always interested in lists of “the best” things in a wide range of categories. One of the features that makes such lists interesting is the way the entries change over time. Here you will develop a software application that allows its users to browse a number of online “top ten” lists, including the ability to compare old and new versions. Your application will have a Graphical User Interface that allows its user to preview the lists they are interested in. They will then be able to export a more detailed version of any chosen list, which can be examined in a standard web browser.

This “capstone” assignment is designed to incorporate most of the concepts taught in ITD104. To complete it you will need to create a Python program to: (a) create an interactive Graphical User Interface using Tkinter; (b) open local documents or download web documents, 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 and export top-ten lists downloaded from the web. There must be at least two distinct lists available, and both old and current versions of the lists must be made available. Most importantly, the online web documents from which you collect your lists must be ones that are updated on a regular basis, either daily or weekly, so that the old and new lists are different.

For the purposes of this assignment you have a free choice of which lists your application will display, provided they always contain at least ten items, are updated frequently, and include the name of each item listed and at least one distinctive “attribute” for each item. Your application must offer access to (at least) two entirely different lists. The lists could be:

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

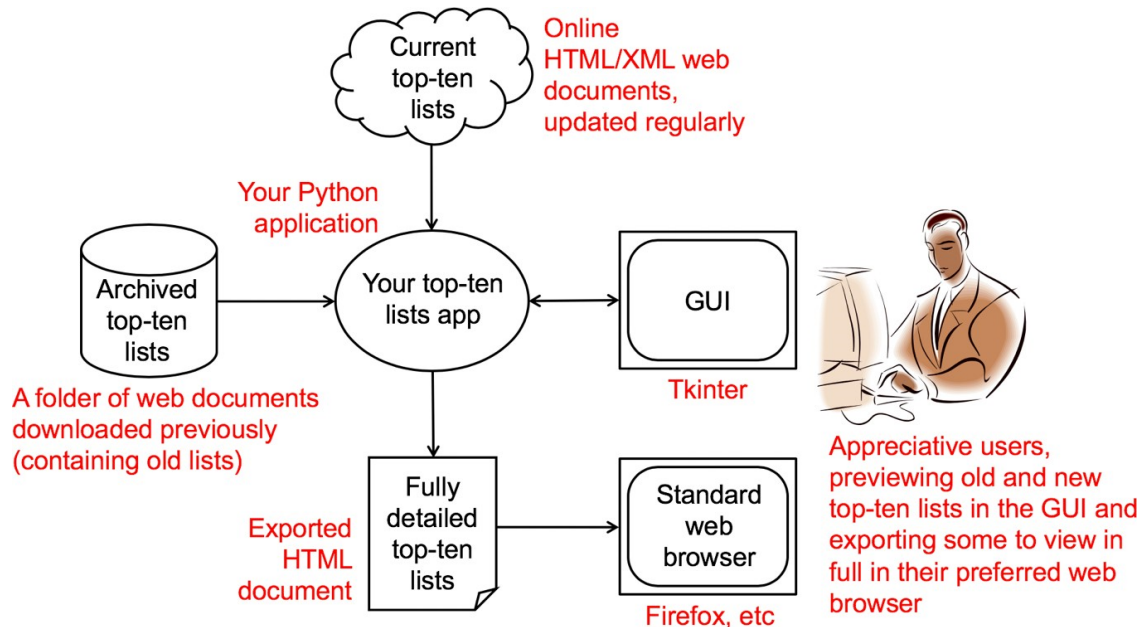
However, whatever lists you choose, you must confirm that the online web documents are updated frequently. For each item in each list the source web site must contain the item's name and some other distinguishing "attribute" of the item listed. Attributes could be:

- an image or photo,
- some additional property of the item other than its name, such as the author of a book or the lead actor in a movie,
- a detailed textual description of the item,
- some kind of numeric score, ideally one which justifies or explains the item's appearance in the list, such as a number of user votes or downloads,
- etc.

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 such lists is sport. However, you cannot always rely on sporting lists being updated "out of season". Therefore, if you choose to use a sports-based list, you must confirm that the sport is being played during the period in which this assignment will be developed and assessed, i.e., December 2019 and January 2020.

Using the data in the online top-ten lists 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 allows the contents of several top-ten lists to be previewed in the GUI. One collection of lists is static and is stored in an "archive", i.e., a folder of previously-downloaded HTML/XML documents. The other source of lists is the "live" Internet. Your application must offer both a previously-downloaded and a "current" list of two different kinds. Having previewed the lists in the GUI, the user can then choose to export them to an HTML file. This file will contain a more detailed version of the list than the one previewed in the GUI and can be studied by the user in any standard web browser at their leisure.

This is a large 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:

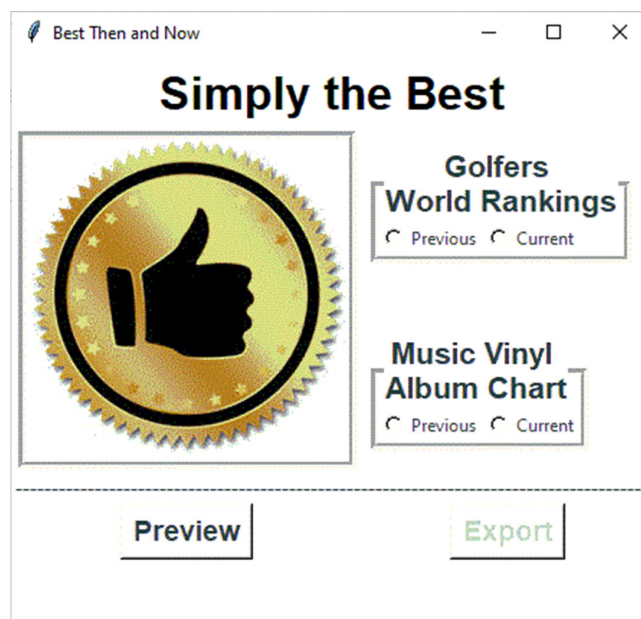
1. Develop code that allows the static, archived top-ten lists to be previewed in the GUI.
2. Extend your solution so that it allows the corresponding “live”, online top-ten lists to be previewed.
3. Extend your solution further so that any of the lists previewed, archived or live, can be exported as HTML documents.

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 a sample solution, which uses data extracted from two different web sites, one which lists the top world ranked golfers, and one which lists the UK's most popular music albums. These lists are updated online at least weekly, so the program is designed to continue working even after the lists have changed.

The screenshot below shows the example solution's GUI when it first starts. (Choose your own name and GUI design.)

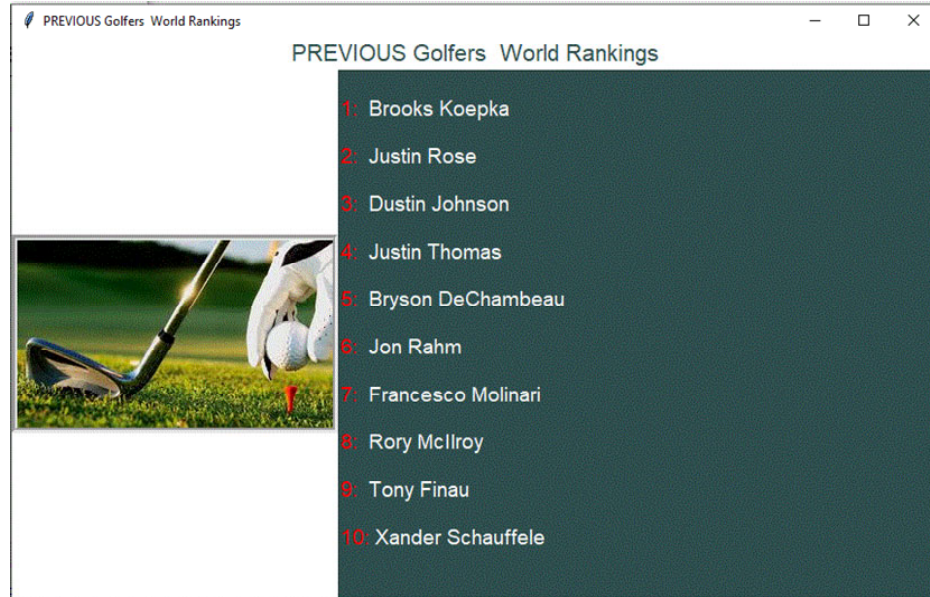


The GUI has a title, an image, four radio buttons allowing the user to select lists of interest, and two push buttons allowing the user to preview or export the list selected. You do not need to copy this example and are encouraged to design your own GUI with equivalent functionality. For instance, menus could be used to allow the selection rather than radio buttons.

Previewing old lists

For each of the two types of list, the sample app allows the user to preview an old, previously- downloaded copy of the list or the current one online. Working with unchanging web documents is obviously easiest, so I recommend you start your solution with this requirement.

For each of the two lists in the example app, web pages were downloaded (one on **December 8th** and the other on **November 30th**) and stored in a folder to serve as the “archive” of old lists. The user can preview any of these lists by selecting the corresponding radio button and pressing the “Preview” button. For instance, pressing the button while the “Previous Golfers World Rankings” list is selected causes the following window to be displayed, showing the world ranking of golfers that was downloaded last year:

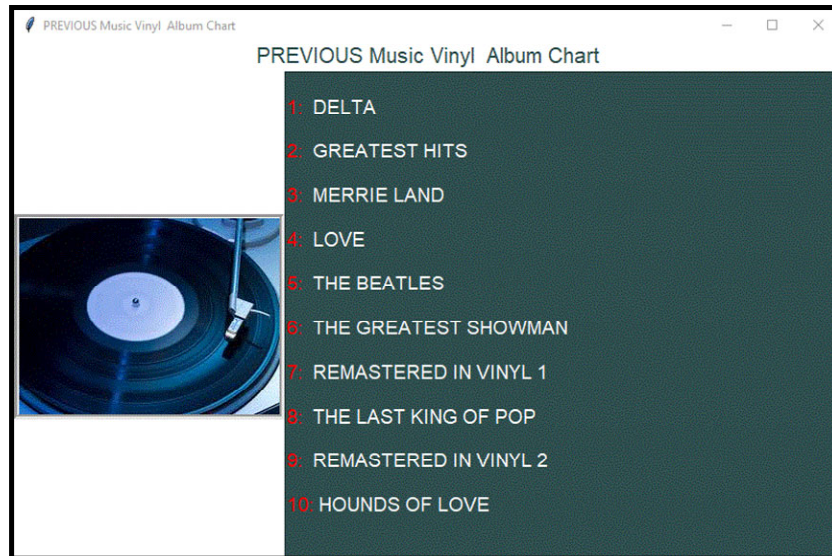


The sample app pops up a new window to display this data, but again you are free to design your own GUI. A single window could be used for all functions instead.

Of course, we can select a different "previous" list for previewing via the GUI, as follows.



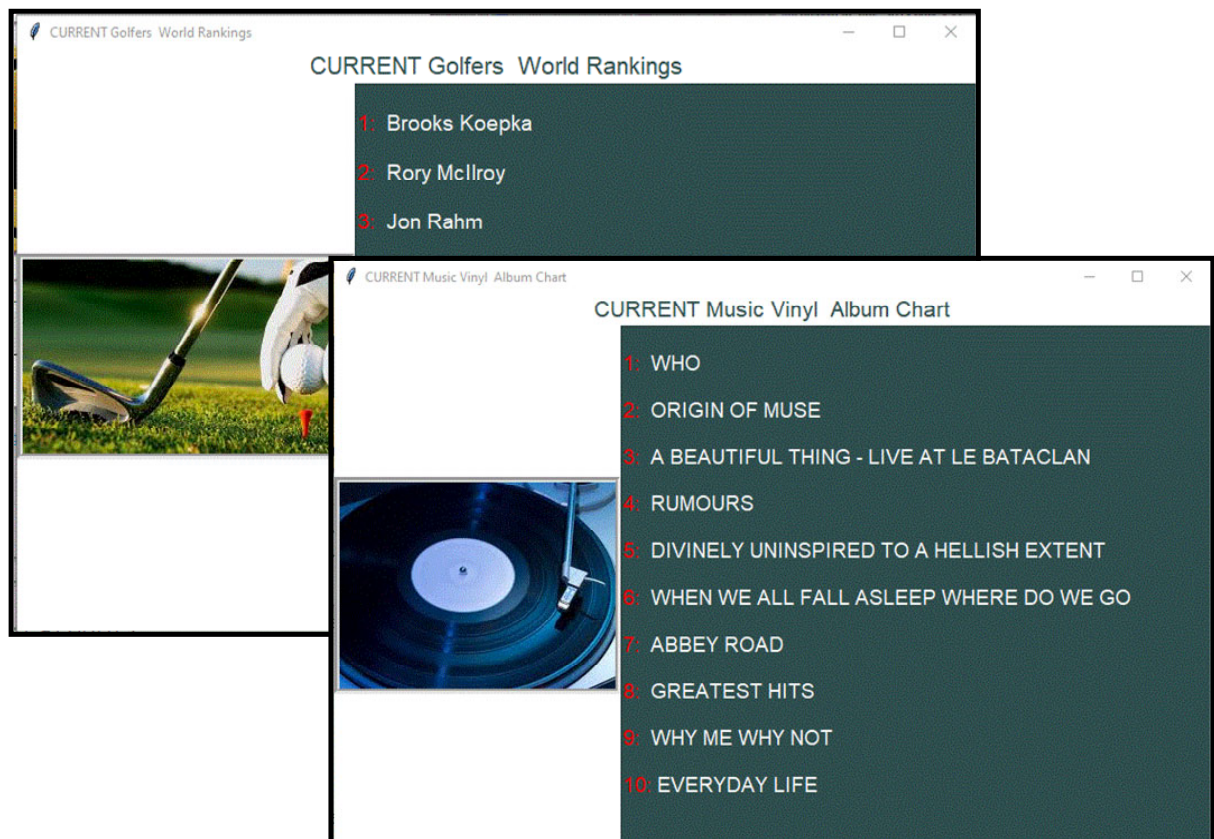
Pressing the “Preview” button in this case causes the following list to be displayed, based on a web document previously downloaded from the “OfficialCharts” web site for the UK’s music album chart.



The main GUI window and each of the two “preview” windows contains an evocative image. These GIF images are all stored in local files, in the same folder as the Python application.

Previewing current lists

As well as previewing old lists, the user can also preview the latest (current) online data. For instance, when the current versions of the two lists are viewed we see:




Exporting old and current lists

The list previews provided in the GUI are very simple, consisting of just an image, the list's name, and an ordered list of the items' names. To give the user further information about lists of interest, the app also allows lists to be exported as HTML documents that can be viewed in a web browser. The exported lists contain additional information, including the list's publication date, the file or web site from which the data was obtained, and some additional distinguishing attribute for each item in the list.

For instance, if the user exports the archived list of most-discussed TV shows, the app creates an HTML file which looks like the following when opened in a web browser.

PREVIOUS...
Golfers World Rankings
8th December 2018



#	Golf	Country
1	Brooks Koepka	USA
2	Justin Rose	ENG
3	Dustin Johnson	USA
4	Justin Thomas	USA
5	Bryson DeChambeau	USA
6	Jon Rahm	ESP
7	Francesco Molinari	ITA
8	Rory McIlroy	NIR
9	Tony Finau	USA
10	Xander Schauffele	USA

archives/download_golf_2018.html

When viewed in this form the name of (and a link to) the previously-downloaded HTML file is displayed (at the bottom of the page) from which the Python program extracts the information. The publication date for the list is also shown at the top of the page. Note that this date is extracted from the source web document; it is *not* the date the list was downloaded.

Most importantly, the exported list contains an extra attribute for each list item. In this case the attribute chosen is the Country of the golfer's origin. Also notice that a different image is used to identify the list. So that the generated HTML file can be viewed on any computer, the image(s) contained in the exported document must all be links to online images, not local files.

Of course, current lists can also be exported. When the current music chart list was exported (on December 16th, 2019) the program created the following web document:

<i>CURRENT...</i> <i>Music Vinyl Album Chart</i>		
13 December 2019 - 19 December 2019		
		
#	Albums	Artist
1	WHO	WHO
2	ORIGIN OF MUSE	MUSE
3	A BEAUTIFUL THING - LIVE AT LE BATACLAN	IDLES
4	RUMOURS	FLEETWOOD MAC
5	DIVINELY UNINSPIRED TO A HELLISH EXTENT	LEWIS CAPALDI
6	WHEN WE ALL FALL ASLEEP WHERE DO WE GO	BILLIE EILISH
7	ABBAY ROAD	BEATLES
8	GREATEST HITS	QUEEN
9	WHY ME WHY NOT	LIAM GALLAGHER
10	EVERYDAY LIFE	COLDPLAY

<https://www.officialcharts.com/charts/vinyl-albums-chart/>

All the details, including the publication date up the top, were extracted from the online web document, which is identified down the bottom. In this case the extra “attribute” is the album’s artist. The URL and link at the bottom of this page takes the user to the current online source of the listing.

You are *not* required to follow the details of the demonstration GUI or exported documents. You are strongly encouraged to use your own skills and initiative to design your own solution, provided it has all the functionality shown above.

Extracting the HTML elements

To produce the lists for previewing and exporting, the application used regular expressions to extract elements from the relevant web documents, whether they were stored in the static archive or downloaded when the program is run.

To do so, we first need to download copies of the web pages and then study them to identify text patterns that would allow us to find the specific parts we wanted. For instance, by inspecting the HTML code from the UK music album charts it was found that the album’s names always appeared after the following tag:

```
<div class="title">...
```

This knowledge was enough to allow us to create a regular expression to extract the album names using Python’s `findall` function. The names of the artists were extracted similarly.

Sometimes it’s easier to use other Python features as well as, or instead of, regular expressions. For instance, when I looked at the US Nielsen Survey web page I found it hard to use for this assignment because the web site contains multiple lists embedded in

the same HTML document. In this case it may have been necessary to use Python's `string` `find` method to discover the starting point of the "social media" list I wanted, and then use string indexing to extract just the part of the HTML code containing the list needed. Cutting the document down in this way made it easier to use `findall` to extract the relevant elements.

Sometimes it's also necessary to download more than one web page to get all the data necessary. Some sites are actually RSS Feeds which have links to multiple copies of lists. To display the "current" list it may be necessary to first extract the URL of the latest such list and use that information to separately download the web page containing the list of interest.

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 lists seen by the user must not contain any extraneous tags or unusual characters that would interfere with the list's appearance.

Exporting the HTML document

A small part of the HTML code generated by the Python program is shown below, in this case for the current Music Vinyl Album Chart. Although not intended for human consumption, the generated HTML code is nonetheless laid out neatly, and with comments indicating the purpose of each part.

```
1
2 <!DOCTYPE html>
3 <html>
4
5 <head>
6 <!-- An automatically generated HTML document -->
7 <title>Simply The Best ... Previous and Current</title>
8 <!-- Fix the width of all elements and centre them -->
9 <style>
10 p {width: 500px; margin-left: auto; margin-right: auto}
11 h1 {width: 500px; margin-left: auto; margin-right: auto}
12 h2 {width: 400px; margin-left: auto; margin-right: auto}
13 table {width: 600px; margin-left: auto; margin-right: auto}
14 </style>
15 </head>
16
17 <body>
18
19 <!-- Titles and cost summary -->
20 <h1 align="center"><em>CURRENT... <br>Music Vinyl
21 Album Chart</em></h1>
22 <h3 align="center">30 November 2018 - 06 December 2018</h3>
23
24 <p align="center"><img src = "https://encrypted-tbn0.gstatic.com/s
25
26
27 <!-- Items description and additional info) -->
28 <table border=1>
```

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 the demonstration solution the user can only produce the exported HTML page **after** the listing has been previewed. This is achieved by having the “Export” button disabled until the list is previewed. Alternatively a clear error message could be displayed if the user pushes the buttons in the wrong order.

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 “old” lists you should download the web documents using the `web_doc_downloader` program, or a similar application. This will ensure that the “old” and “current” pages have the same format, thus making your pattern matching task easier.

For instance, the music chart site I used appears as follows when I view it in a web browser.

Pos	LW	Title, Artist	Peak Pos	WoC
1	2 ↑	GREATEST HITS QUEEN UMC	1	86
2	5 ↑	THE BEATLES BEATLES EMI	2	3
3	6 ↑	THE GREATEST SHOWMAN MOTION PICTURE CAST RECORDING ATLANTIC	1	35
4	11 ↑	RUMOURS FLEETWOOD MAC WEA	2	174
5	19 ↑	STAYING AT TAMARA'S GEORGE EZRA COLUMBIA	1	31
6	15 ↑	THE DARK SIDE OF THE MOON PINK PANTHER COLUMBIA	1	143

Notice that all the elements needed for the application appear in the document, including the publication date, the names of the albums and artists. 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 overleaf 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 kind 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 each source web page is

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.

```

1  <!doctype html>
2  <!--[if lt IE 7]><html class="no-js ie6 oldie" lang="en"><![endif]-->
3  <!--[if IE 7]><html class="no-js ie7 oldie" lang="en"><![endif]-->
4  <!--[if IE 8]><html class="no-js ie8 oldie" lang="en"><![endif]-->
5  <!--[if gt IE 8]><!-->
6  <!--[if gt IE 8]><!-->
7  <html class="no-js" lang="en">
8  <!--<![endif]-->
9
10 <head>
11
12
13 <meta charset="utf-8" />
14 <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
15
16 <title>Official Vinyl Albums Chart Top 40 | Official Charts Company</title>
17 <meta name="description" content="Compiled by the Official Charts Company, th
18 <meta name="keywords" content="Top 40, UK Top 40, Charts, Top 40 UK, UK Chart
19 <meta name="viewport" content="width=device-width,initial-scale=1" />
20
21
22 <meta property="og:image" content="https://www.officialcharts.com"/>
23
24 <link rel="shortcut icon" href="/favicon.ico"/>
25
26 <link href="//fonts.googleapis.com/css?family=Roboto+Slab:400,300,700" re
27 <link rel="stylesheet" href="/Css/style.css" />
28
29
30 <!-- Ad integration -->
31
32
33 <script async="async" src="https://www.googletagmanager.com/tag/js/gtm.js"><
34

```

Specific requirements and marking guide

To complete this task you are required to produce an application in Python 3 similar to that above, using the provided `the_best.py` template file as your starting point. In addition you must provide a folder containing (at least) two previously-downloaded web documents to serve as your archive of old lists and all the GIF images needed to support your GUI. (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 (2%).** 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 acts as a “logo” to identify your application. Note that, in general, Python Tkinter implementations only support GIF images. 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 the 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 allow the user to select two different top-ten lists, in both old and current forms. It must be possible to easily distinguish the archived “old” lists from the current “live” ones.
- A widget or widgets that allows the user to choose to preview or export any of the four (or more) lists on offer. Note that this capability could be combined with the one above, depending on which kind of GUI widgets you choose to use.
- If your solution is developed on a Mac, it must also be useable on a Windows machine. Mac users will need to ensure in particular that every widget on the GUI is visible and usable while running on Windows.
- **Previewing old “archived” top-ten lists in the GUI (2%).** Your GUI must be capable of displaying (at least) two distinct archived top-ten lists. These lists must be sourced from completely different websites (not just different pages from the same website). For each list you must show:
 - a heading in the GUI (not just the window title) identifying the list,
 - an image/logo that characterises the list, and
 - a numbered list with the top-ten items extracted from the archived web document.

The list items must be extracted from HTML/XML files previously downloaded from online and stored in your “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.

- **Previewing “live” online top-ten lists in the GUI (2%).** Your GUI must be capable of displaying (at least) two distinct “live” top-ten lists, as currently available online at the time the program is run. These lists must be sourced from completely different websites (not just different pages from the same website). For each list you must show
 - a heading in the GUI (not just the window title) identifying the list,
 - an image/logo that characterises the list, and
 - a numbered list with the top-ten items extracted from the online web document.

The list data must be extracted from HTML/XML files downloaded from online 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**, i.e. at least weekly.

- **Exporting HTML documents containing (old or current) top-ten lists (2%).** Your program must be able to generate an HTML document containing any top-ten list selected by the user, from both the “archived” lists and the online ones. The data 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 reliant 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:

- An appropriate title in the browser tab.
- A heading identifying the list.
- If an archive list, the name of the local HTML file from which the data was extracted; If a current list, a link to the URL address of the online web page from which the top-ten data was extracted. This must be sufficient to allow the original source documents to be found easily (so that the marker can compare the original web pages with your displayed lists).
- The publication date for the list, extracted from the source document (not just the date when the file was downloaded because they may not be the same).
- An image characterising the list, downloaded from online when the generated HTML document is viewed (i.e., not from a local file on the host computer).
- A numbered list of the top-ten items. For each item (at least) the following data must be displayed:
 - The item's position in the top ten.
 - The item's name.
 - Some other distinguishing attribute of the item. The attribute may be textual or may be an image.

All of this data must be extracted via pattern matching from web documents downloaded from online. Most importantly, each of these sets of items *must all belong together*, e.g., you can't have the name of one item paired with an attribute of another. Each of the elements must be extracted from the original document separately.

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/XML 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 (2%).** Both your Python program code and the generated HTML code must be *presented in a professional manner*. See the coding guidelines in the *ITD104 Code Presentation Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this for Python. 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.

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 other Python modules without prior approval from your 'client' (i.e., the person who is going to mark your assignment!).** Failure to adhere to this requirement will result in a deduction of two percent (2%).

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

Your solution is *not* required to follow precisely the example shown above. Instead you are strongly encouraged to *be creative* in your choices of web sites to access, 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, I 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 text file, as well as returning the document's contents to the caller as a character string. You can use this program to save copies of your chosen web documents for storage in your archive, as well use similar code 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.)

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. I 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.

Security warning and plagiarism notice

This is an individual assessment item. Be warned that marks will only be awarded for YOUR OWN WORK, so if you copy from someone else's solution (in part or in full), or get someone to write your solution for you (in part or in full) – expect to fail.

Note that it is considered a breach of academic integrity if you 'self plagiarise', i.e., reuse work you have previously submitted for assessment in this or any other units. If you are repeating this unit, you must therefore use completely different websites for this assignment.

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 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 the 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 have a *private* repository that cannot be seen by anyone else. For instance, students can apply for a *free* private repository in GitHub to keep their work secure (<https://education.github.com/pack>). However, I recommend that the best way to avoid being prosecuted for plagiarism is to keep your work well away from the Internet and your fellow students!

Deliverables

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

- 1 Your `the_best.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 and will automatically receive a mark of 0.**
- 2 Several *small* GIF 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 lists. 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”.**

How to submit your solution

A link is available on Blackboard under Assessment for uploading your solution before the deadline. 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. I 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 the Friday before the deadline.

Appendix A: Some web sites that may prove helpful

For this assignment you need to find two (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 there is no guarantee that they all are. In particular, I 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.

- Top 100 stocks, <https://www.barchart.com/stocks/top-100-stocks>, plus several other promising pages at the same site.
- Surprisingly bullish stocks (ranked by standard deviations), <https://www.barchart.com/stocks/price-surprises>.
- Rotten Tomatoes has lots of lists that look useful, such as top box office, <https://www.rottentomatoes.com/browse/in-theaters>, most popular TV, <https://www.rottentomatoes.com/browse/tv-list-2> and top DVD and streaming, <https://www.rottentomatoes.com/browse/top-dvd-streaming>.
- Another top ten box office movies list, <https://www.fandango.com/rss/top10boxoffice.rss>.
- The most-played tracks on Soundcloud, <https://soundcloud.com/charts/top>, which can also be listed by genres.
- Spotify charts Australia, <https://spotifycharts.com/regional/au/daily/latest>, plus other countries and dates.
- Various Apple RSS feeds, including songs, albums, movies, tv, and apps can be found at <https://www.apple.com/rss/>. For instance, iTunes store top songs, <http://ax.itunes.apple.com/WebObjects/MZStoreServices.woa/ws/RSS/topsongs/xml>, iTunes top albums <http://ax.itunes.apple.com/WebObjects/MZStoreServices.woa/ws/RSS/topalbums/xml>, etc.
- Google trends has many sites such as Google Australia daily search trends, <https://trends.google.com/trends/trendingsearches/daily?geo=AU>, plus the same data as an RSS feed, <https://trends.google.com/trends/trendingsearches/daily/rss?geo=AU>, and Google real-time search trends, <https://trends.google.com/trends/trendingsearches/realtime?geo=AU&category=all>.
- Google Play lists such as top selling apps, https://play.google.com/store/apps/collection/topselling_paid
- Another list of daily box office receipts, <https://www.boxofficemojo.com/daily/chart/>.
- Lists of the week's top-ten pirated movies, <https://torrentfreak.com/category/dvdrip/feed/>
- Players' rankings for Steam, <http://steamcharts.com/top>, seemingly updated continuously?

- Ultimate Guitar's top songs ranked by clicks in the last 24 hours, <https://www.ultimate-guitar.com/top/tabs>, which presumably updates frequently, but this has not been confirmed.
- US Nielsen rankings for television, music, video games, social interactions, <https://www.nielsen.com/us/en/top10s.html>. (Update rates vary and all lists are embedded in one web document.)
- New Zealand music charts, singles, <https://www.nztop40.co.nz/chart/singles>, and albums, <https://www.nztop40.co.nz/chart/albums>, updated weekly.
- Several UK music charts, *as used in the demonstration solution*, <http://www.officialcharts.com/charts/>, updated weekly, e.g., singles, <http://www.officialcharts.com/charts/singles-chart-update/>, vinyl albums, <http://www.officialcharts.com/charts/vinyl-albums-chart/>, etc.
- Australian TV ratings, <https://decidertv.com/ratings/>, apparently updated daily.
- US TV Guide's trending TV shows, <https://www.tvguide.com/trending-tonight/>.
- American Top 40 music list, <https://www.at40.com/charts/top-40-238/latest/>, updated weekly.

Here are some other sites that may be suitable, but it's not always clear how often they are updated, so you need to check this. It's not enough to take the publisher's word. Just because a site says "updated daily" doesn't mean it will *change* daily!

- Top ten baby names, <https://www.babynamenameguide.com/toptenDaily.asp>, updated daily but how often does it change?
- Rotten Tomatoes top 100 movies, including "today's" top rated movies, <http://www.rottentomatoes.com/top/bestofrt/>.
- Video game sales, with lots of lists but probably not updated often, http://vgsales.wikia.com/wiki/Video_Game_Sales_Wiki, plus many others, e.g., http://vgsales.wikia.com/wiki/Best_selling_game_franchises.
- Australian music charts, <https://australian-charts.com/>.
- Chess rankings, <https://2700chess.com/>, but may not be updated often.
- GuildWars statistics, <https://leaderboards.guildwars2.com/en/na/achievements>, apparently updated daily?
- BBC Top 40 singles chart, <http://www.bbc.co.uk/radio1/chart/singles>, and album chart, <http://www.bbc.co.uk/radio1/chart/albums>.
- Vodafone Big Top 40 music chart, <https://www.bigtop40.com/>.
- What seems to be a list of trending YouTube videos, <https://www.youtube.com/feed/trending>.
- Official World Golf Ranking, <http://www.owgr.com/ranking> *as used in the demonstration solution*, (although we generally suggest avoiding sports rankings, because they may be out of season, this one seems to change a lot).
- YouTube channels, most subscribed, <https://www.statsheep.com/p/Top-Subscribers>, and most watched, <https://www.statsheep.com/p/Top-Video-Views> (plus several other categories, but it's not clear how often they're updated).
- Character rankings for Street Fighter 5, <https://www.eventhubs.com/tiers/sf5/>,

and Super Smash Bros 4, <https://www.eventhubs.com/tiers/ssb4/>, and similarly for lots and lots of other games.

- Publisher's Weekly Top 10 books, <http://www.publishersweekly.com/pw/nielsen/top100.html>, Top 10 science-fiction books, <https://www.publishersweekly.com/pw/nielsen/xscifi.html>, plus dozens of other categories.
- Most popular Anime series, <http://myanimelist.net/topanime.php?type=bypopularity> (claims to update twice daily), plus top Anime TV series, <https://myanimelist.net/topanime.php?type=tv> and top currently airing Anime, <https://myanimelist.net/topanime.php?type=airing>, and several other such lists.
- Aria albums chart, <http://www.ariacharts.com.au/charts/albums-chart>, singles chart, <https://www.ariacharts.com.au/charts/singles-chart>, music DVDs, <https://www.ariacharts.com.au/charts/music-dvds-chart>, plus many more music categories.
- IMDb movie box office receipts, <https://www.imdb.com/chart/boxoffice>, plus simpler mobile version, <https://m.imdb.com/chart/boxoffice/>.
- The Ranker site has hundreds of crowd-sourced popularity lists, but it's not clear how often each is updated, e.g., fantasy books, <https://www.ranker.com/list/best-fantasy-book-series/ranker-books>, best animated TV series, <https://www.ranker.com/crowdranked-list/the-greatest-animated-series-ever-made>, plus thousands of lists on virtually any topic!
- Billboard Top 100 lists, <https://www.billboard.com/charts>, probably updated weekly, e.g., the Hot 100, <https://www.billboard.com/charts/hot-100>.
- Ranker's RSS feed of its most popular lists, <https://www.ranker.com/feed/mostpopularlists.rss>.
- Card deck ratings for an online game, <http://hearthpwn.com/decks?sort=rating&sort=rating>, updated regularly, but very complicated page structure.
- World's worst spammers, <https://www.spamhaus.org/statistics/spammers/>, updated daily, but does it change often?
- List of trending books, <https://www.goodreads.com/shelf/show/trending>, but no indication of update frequency.
- Games rankings for Dota 2, <https://www.gosugamers.net/dota2/rankings>, Counter-Strike, <http://www.gosugamers.net/counterstrike/rankings>, etc.
- Various music charts, <https://www.beatport.com/genre/trance/7>, but site has a very complex structure.
- IMDb top-rated movies, <https://www.imdb.com/chart/top>, plus many other lists, e.g., most popular comedy titles, <https://www.imdb.com/search/title?genres=comedy>, top-rated TV shows, <https://www.imdb.com/chart/toptv>, most popular movies, <http://www.imdb.com/chart/moviemeter>, most popular celebrities, <https://m.imdb.com/chart/starmeter>, but may not change often?

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 or in order to protect the owner's computing resources and data assets. 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 my Python program.

Please enable cookies.

Error 1010 • 2017-04-26 02:02:57 UTC • 2017-04-26 02:02:57 UTC

Access denied

What happened?

The owner of this website (www.wayofcats.com) has banned your access based on your browser's signature

- Performance & security by Cloudflare

Cloudflare Ray ID: 3555

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.