Assignment 2, Part A: Internet News Archive

(20%, due 5:00pm Wednesday, October 25th)

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 13. Whether or not you complete Part B you will submit only one solution, and receive only one mark, for the whole 25% assignment.

Motivation

The Internet gives us access to a vast amount of data in digital form, but one of its disadvantages is that this data is ephemeral. Web sites appear, update their contents, and disappear at bewildering speed. While we can still read books printed hundreds of years ago, or even a papyrus written thousands of years in the past, we often fail to find a web site or online article from just a few years ago!

One prominent attempt to preserve digital data is the *Internet Archive* (http://www.archive.org/index.php), a vast repository of web pages downloaded and stored over a long period of time. Via the Internet Archive's *Wayback Machine* interface you can retrieve archived copies of web pages from specific dates in the past. In this assignment you will build your own Internet Archive-like IT system using all the technologies you have learned in this teaching unit. Your system will give its user the ability to view news or current affairs articles that have been stored in a permanent archive, as well as the current news. It will have a graphical interface that allows the user to select dates from the archive, after which the relevant news articles can be extracted and viewed in a standard web browser. It will also allow the user to access the latest data from an online "feed".

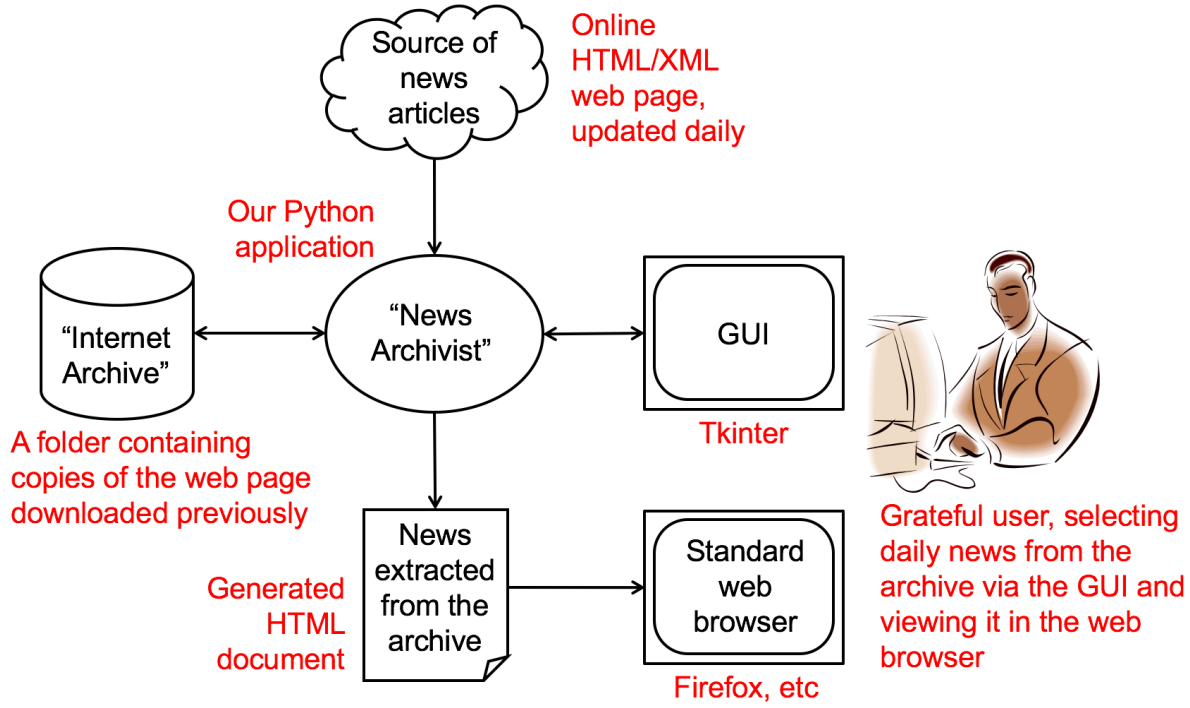
Goal

For the purposes of this assignment you need to choose a regularly-updated source of online news or current affairs. You have an entirely free choice of the news category, which could be:

- world news,
- politics,
- fashion and lifestyle,
- arts and entertainment,
- sports,
- business and finance,
- science and technology,
- etc.

Whatever category you choose, you must be able to find an online web site which contains at least ten current stories or articles on the topic at all times. The web site must be updated regularly, typically at least once a day. Each of its articles must contain a heading, a short synopsis, a (link to a) photograph, a link to a full description of the story, and a publication date. A good source for such data is Rich Site Summary (RSS) web-feed documents. Appendix A below lists many such sites, but you are encouraged to find your own of personal interest.

Using this data source you are required to build an IT system with the following general architecture.



Your "News Archivist" application will be a Python program with a Graphical User Interface. Under the user's control, it extracts individual elements from web documents stored in an "archive" (a folder of previously downloaded HTML/XML documents) and uses these elements to construct an entirely new HTML document which summarises a particular days' top stories in a form which can be viewed in a standard web browser. It also allows the user to download the current version of the source web page and store it in the archive as the "latest" news.

Illustrative example

To demonstrate the idea, we have created an archive from the Australian Broadcasting Corporation's Queensland news service. We began by populating the archive by downloading a copy of the relevant news feed site each day over a period of one week. Importantly, as explained below, we downloaded the web document using a small Python script, rather than copying it from our web browser, to ensure that the files in the archive are in the same format that our Python application receives them from the Internet.

We then developed our "News Archivist" application with a "back end" that uses the files in the archive to generate all-new HTML documents summarising a particular days' news and a GUI "front end" that lets the user control the process.

The screenshot below shows our example solution's GUI when it first starts.

The GUI has the following features: a visual image to clearly describe its purpose; a list that allows the user to select from seven different files in our archive, plus the "latest" news; a button to extract the selected days' news; a button for displaying the news in the host operating system's default web browser; a button for downloading the "latest" news from the Internet and storing it in the archive; and a "progress" or "status" area up the top to instruct the user on what to do and to display messages. Your solution does *not* need to copy this design, as long as it provides equivalent functionality. For instance, radio buttons or a pull-down menu may be a better way of allowing the user to make their selection.

Having selected a day the user can then choose to extract that days' news from the archive. In the following case the user has selected the oldest day in the archive and has pressed the "extract news" button.

At this point the application uses the corresponding copy of the web page in the archive to generate an entirely new HTML document containing a summary of that day's news. This document is written into the same folder as the Python application. Since it is an ordinary HTML file the user can open it manually using their preferred web browser. In this instance,

however, our GUI also provides a "display news" button which automatically opens the document in the host operating system's default browser. The first part of the displayed news as it appears in the web browser in this case is as shown below.

The generated web document has a heading which identifies its contents, including the date the news was archived, an image illustrating the source or category of news (the ABC logo in this case), a link to the original "archived" web page, and the name of the application's developer (we've used cartoon character Grimly Feendish, but obviously this should be your name).

Importantly, the image file containing the ABC logo seen above does *not* reside on the local computer. It is actually a link to an image file online. In order to keep the size of your system manageable, your News Archivist application and its archive of files may not contain any image files except for GIF images used in the Tkinter GUI. All images in the extracted web documents must be links to online files.

Scrolling further down the generated web document then reveals the top ten news stories from this particular date, part of which is shown in the two screenshots overleaf. As can be seen, each of the articles displayed has a story number, a heading, a short synopsis, a photograph, a link to the full description of the story, and a publication date. Once again, the photo in each case is produced by linking to an online source, not a local file.

ABC

News source: http://www.abc.net.au/news/feed/50990/rss.xml
Archivist: Grimly Feendish

---

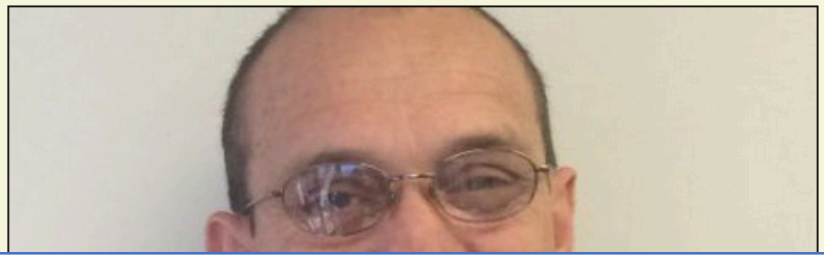
1. Tiny house stolen from Canberra 'spotted in rural Queensland less than a day later'

A tiny house prototype stolen from a Canberra business appears to have made its way to a Queensland town 1,250 kilometres away in a matter of hours, according to the home's owner.

Full story: http://www.abc.net.au/news/2017-09-13/tiny-house-stolen-from-canberra-appears-in-rural-queensland/8939072
Dateline: Wed, 13 Sep 2017 09:20:14 +1000

---

2. Senior Ipswich City Council officer charged by corruption watchdog



4. Kandanga Farm Store focuses on growing food without chemicals

A Queensland couple helps the economic revival of the Mary Valley with a chemical-free produce store.

Full story: http://www.abc.net.au/news/rural/2017-09-13/kandanga-farm-store-goes-chemical-free-mary-valley/8873262
Dateline: Wed, 13 Sep 2017 06:32:39 +1000

5. Outback kids escape the drought for ocean visit

Outback kids from drought-stricken towns travel 1,200 kilometres to swim in the ocean, marking for many their first time in the surf.

Full story: http://www.abc.net.au/news/2017-09-13/surf-swim-first-outback-kids-school-trip-rainbow-beach-qld/8903600
Dateline: Wed, 13 Sep 2017 06:09:35 +1000

Apart from generating documents from "archived" web pages, the other major feature of the application is the ability to add the "current" or "latest" news to the archive. In our demonstration solution this is done by pressing the "archive latest" button. This causes the application to download a current copy of the file from the original web site and add it to the local archive folder.

The freshly downloaded file can then by viewed by selecting "latest" in the menu and "extracting" and "displaying" the selection as usual. In this case we did so on Tuesday, September 19th, 2017 around lunchtime. The current news downloaded to the archive, extracted to produce a new web document containing the top ten stories, and ultimately displayed in a browser, was as shown in the following series of screenshots at this time.

ABC News Archive (Queensland)

Latest Update

News source: http://www.abc.net.au/news/feed/50990/rss.xml
Archivist: Grimly Feendish

1. Anti-Adani protesters arrested near Abbot Point

Police arrest 10 protestors they say were blockading a road to the Abbot Point Coal terminal near Bowen.

Full story: http://www.abc.net.au/news/2017-09-19/anti-adani-protestors-arrested-bowen/8959752
Dateline: Tue, 19 Sep 2017 12:07:47 +1000

Police arrest 10 protestors they say were blockading a road to the Abbot Point Coal terminal near Bowen.

Full story: http://www.abc.net.au/news/2017-09-19/anti-adani-protestors-arrested-bowen/8959752
Dateline: Tue, 19 Sep 2017 12:07:47 +1000

---

2. ABC crew denied access to Gold Coast council media conference

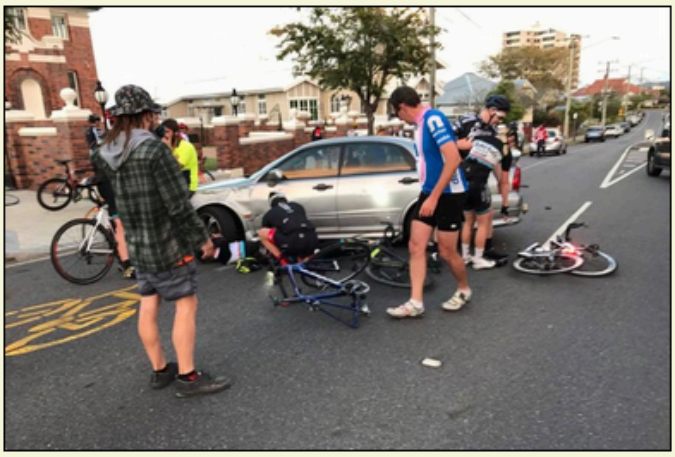
ABC journalists and a cameraman have been denied access to a press conference held by Gold Coast Mayor Tom Tate, a day after Four Corners reported on the extent of political donations to the city's councillors.

Full story: http://www.abc.net.au/news/2017-09-19/abc-crew-denied-access-to-gold-coast-council-media-conference/8959930
Dateline: Tue, 19 Sep 2017 12:03:05 +1000

---

3. Gold Coast Mayor bars ABC from media conference

8. What happens when cyclist and car collide?



In the hourly contest between cyclists and cars on our roads it's not just civility that can go out the window, it's the knowledge that when metal meets flesh and bone there can be only one outcome.

Full story: http://www.abc.net.au/news/2017-09-19/west-end-cycling-accident-photos-calvin-treacy/8868554
Dateline: Tue, 19 Sep 2017 05:31:26 +1000

9. Gold Coast Council concerns prompt push for ban on developer donations

Senior Labor Government figures in Queensland are pushing for a state-wide ban on developer donations over concerns council decisions are being improperly influenced.

Full story: http://www.abc.net.au/news/2017-09-19/labor-figures-push-for-queensland-ban-on-developer-donations/8957218
Dateline: Tue, 19 Sep 2017 05:15:26 +1000

To produce this news summary, our "News Archivist" application used regular expressions to extract elements from the archived web document (see below). For instance, we found that the

publication date for each news story appeared between `<pubdate>` … `</pubdate>` XML tags, which made them easy to extract. However, extracting the synopses for each article proved more awkward. Each story summary appeared between `<description>` … `</description>` tags in the original web page, but there were some other XML markups between these tags which we had to remove before we could include the text in our new HTML document. Your generated documents must not contain any extraneous tags or unusual characters that would interfere with the document's appearance when viewed.

The first part of the HTML code generated by our Python program is shown below.

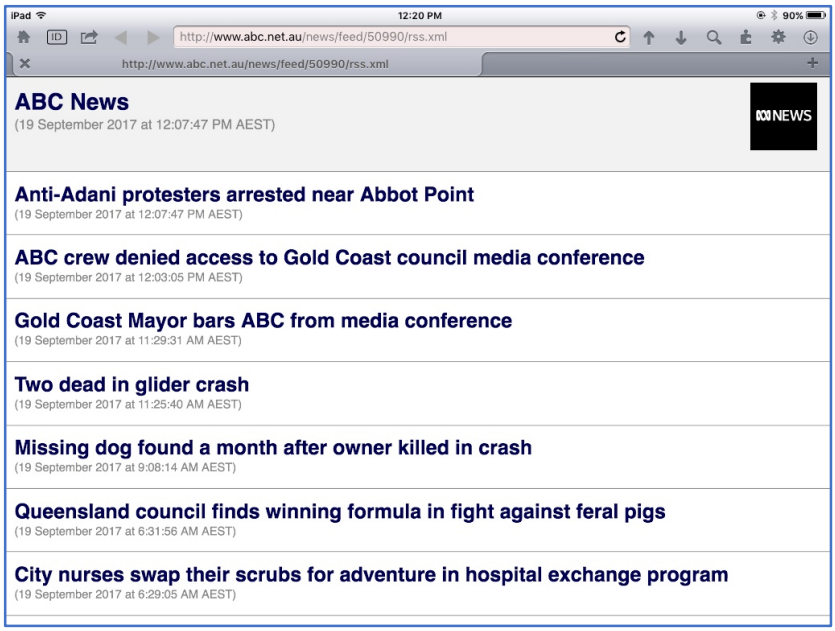
```
1
2  <!DOCTYPE html>
3  <html>
4
5      <head>
6
7          <!-- Title for browser window/tab -->
8          <title>ABC News Archive (Queensland)</title>
9
10         <!-- Overall document style -->
11         <style>
12         body {background-color: beige}
13         p    {width: 80%; margin-left: auto; margin-r
14         h1   {width: 80%; margin-left: auto; margin-r
15         h2   {width: 80%; margin-left: auto; margin-r
16         h3   {width: 80%; margin-left: auto; margin-r
17         hr   {border-style: solid; margin-top: 1em; m
18         </style>
19
20     </head>
21
22     <body>
23
24         <!-- Masthead -->
25         <h1>ABC News Archive (Queensland)</h1>
26         <h2>Latest Update</h2>
27         <!-- ABC Logo from http://www.abc.net.au/news
28         <p style = "text-align:center"><img src = "ht
29         <p><strong>News source:</strong> <a href = "h
30            <strong>Archivist:</strong> Grimly Feendis
31
32         <hr width = "80%" size = 5px>
33
34
35         <!-- A news article -->
36
37         <!-- Headline -->
```

Although not intended for human consumption, the generated HTML code is nonetheless laid out neatly, and with comments indicating the purpose of each part.

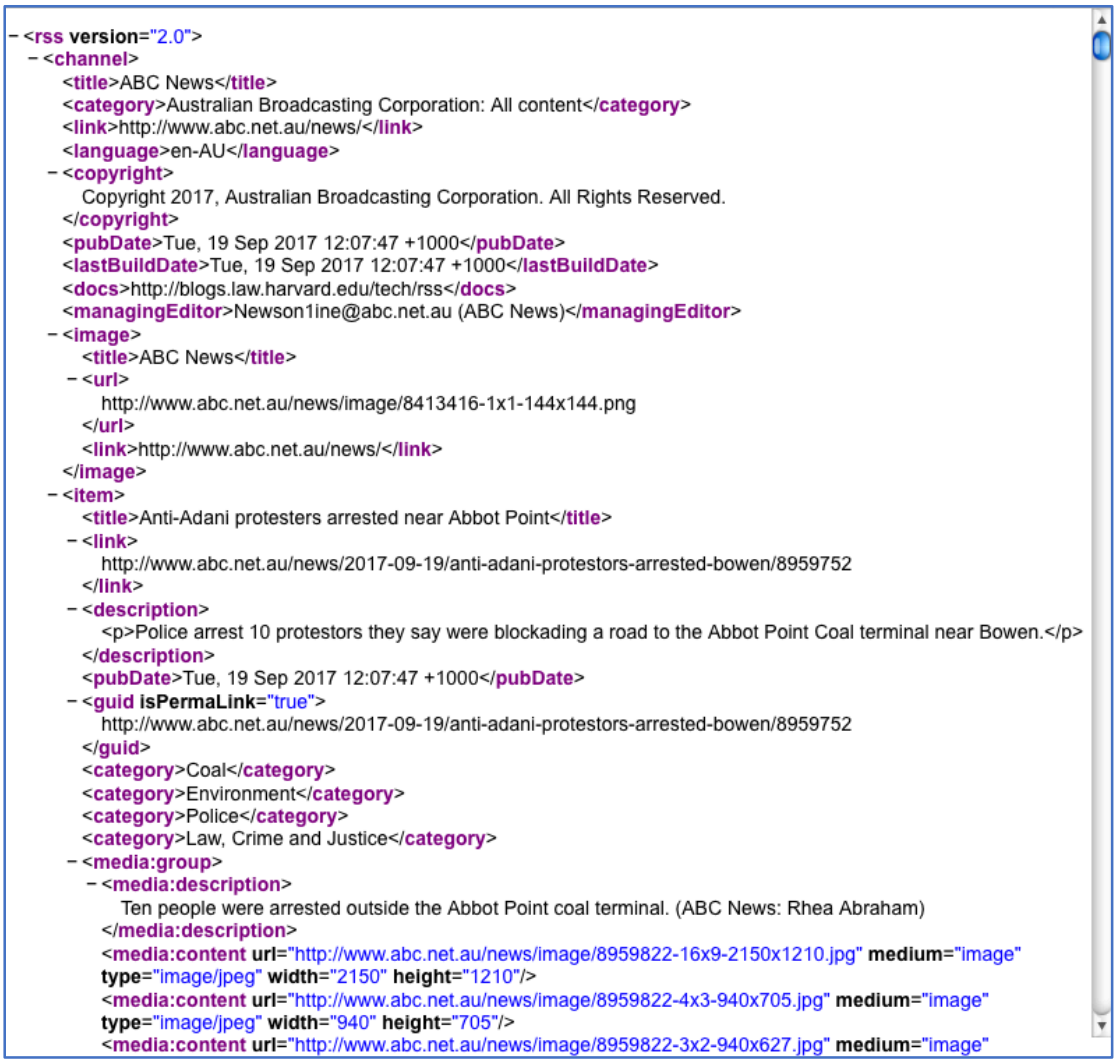
What you see is not necessarily what you get!

A significant challenge for this assignment is that web servers deliver different documents to different web browsers, news readers and other clients. This means the web page you see in a browser may be very different from the web page downloaded by your Python application. Consider again the example above of the "latest news" we downloaded on Tuesday, September 19th, 2017. When viewed in Firefox on a laptop computer the page appeared in the following human-friendly format.

Notice that this page has all the elements we need for this assignment. However, when the same page was downloaded at the same time on a tablet computer it appeared as shown below.



This time the links to the photos and the story synopses are not included at all. Similarly, when we downloaded the same page at the same time with a Python script we were delivered a "raw" XML file which appeared as follows in a browser.

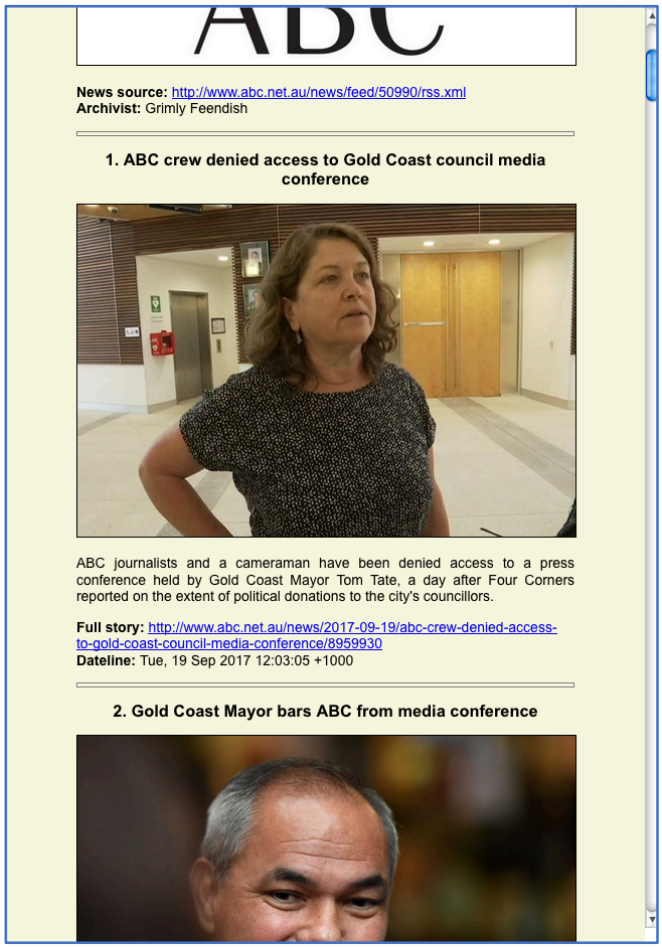


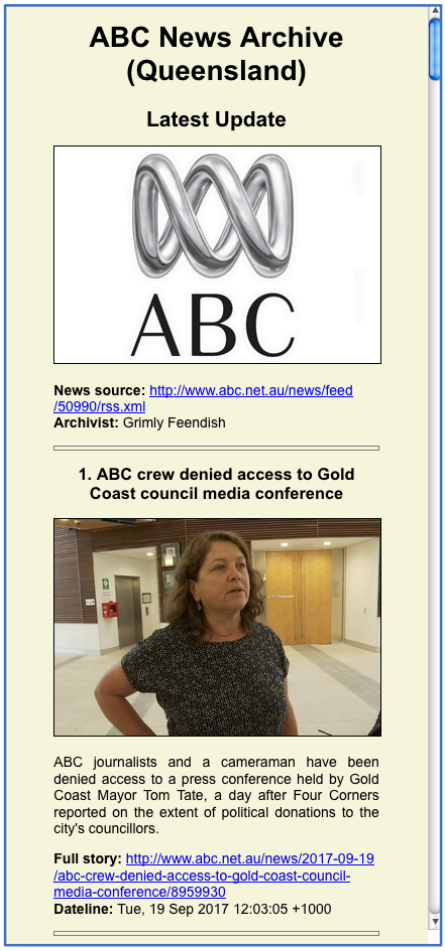
Apart from the lack of formatting information for the browser there are other differences from the previous two versions, such as the inclusion of picture credits for the photos and the way the times are all expressed relative to GMT. This third version was the one stored in our archive and processed by our Python application.

Although confusing, this is actually an *advantage* for our purposes because the XML document delivered to our Python script has a much simpler format than the one delivered to the web browser, and is consequently much easier to extract elements from. However, for this reason it's very important that you populate your archive of web page copies with downloads produced by a Python script, such as the provided `downloader.py` program, rather than copying them out of a web browser.

Web document properties

Naturally an important requirement from the user's perspective is that the generated news document is attractive and easy to read. In particular, the document's layout should adjust elegantly to changes in the size of the web browser's window. For instance, the two screenshots below show how our generated web page adjusts automatically to a change in the browser's width. (These screenshots were taken of the "latest" news about ten minutes before the example above. Notice that the news was updated in between!)





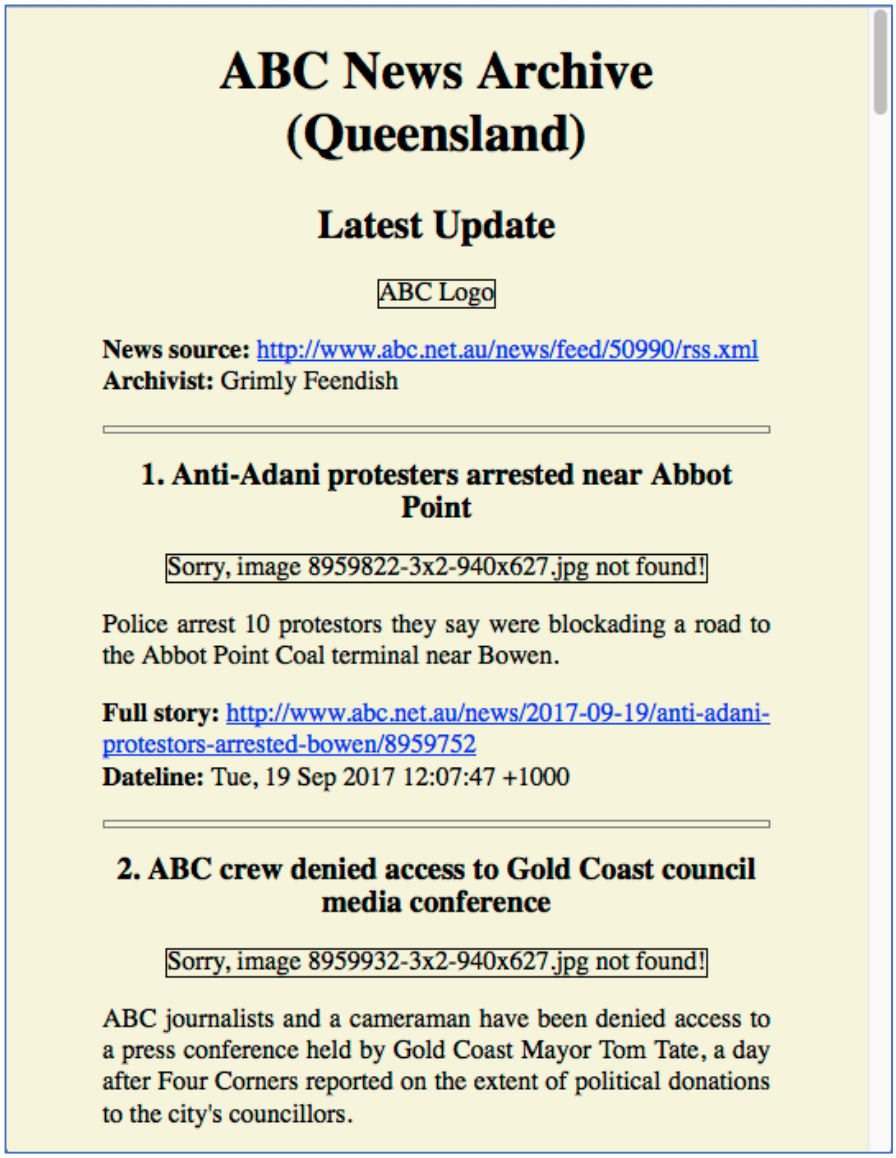
Robustness

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 is required to extract news from the archive after selecting a date. Pressing the "extract news" button without first selecting a date produces the following instructive message in the GUI.

Similarly, the user must download the "latest" news before attempting to extract it from the archive. Selecting the "latest news" from the menu and pressing the "extract news" button without ever having downloaded fresh news from the Internet produces the following error message.

Similarly, the generated HTML documents must be resilient to errors. A particular problem for this system, which afflicts the "real" Internet Archive, is that links to other web documents and images may be broken because the files referenced no longer exist. In this case your generated web document must still display a "best effort" result. For instance, attempting to display the latest news with all links broken (by disconnecting the Internet connection) produced the following result when our generated page was viewed. Notice that alternative text has been provided to describe the missing elements. A similar outcome would be produced when displaying very old news articles for which the images are no longer available online.



ABC News Archive (Queensland)

Latest Update

ABC Logo

News source: http://www.abc.net.au/news/feed/50990/rss.xml
Archivist: Grimly Feendish

---

1. Anti-Adani protesters arrested near Abbot Point

Sorry, image 8959822-3x2-940x627.jpg not found!

Police arrest 10 protestors they say were blockading a road to the Abbot Point Coal terminal near Bowen.

Full story: http://www.abc.net.au/news/2017-09-19/anti-adani-protestors-arrested-bowen/8959752
Dateline: Tue, 19 Sep 2017 12:07:47 +1000

---

2. ABC crew denied access to Gold Coast council media conference

Sorry, image 8959932-3x2-940x627.jpg not found!

ABC journalists and a cameraman have been denied access to a press conference held by Gold Coast Mayor Tom Tate, a day after Four Corners reported on the extent of political donations to the city's councillors.

Specific requirements and marking guide

To complete this task you are required to produce an application in Python similar to that above, using the provided `news_archivist.py` template file as your starting point, accompanied by an "archive" (folder) of **at least seven substantially different web documents** previously downloaded from your chosen news source. Your solution must support *at least* the following features.

- **Generating fixed document elements (2%).** Your program must be able to generate an HTML file which begins with certain fixed elements. These include:
  - a heading which identifies the document's contents,
  - the date the displayed news was archived,
  - an image indicative of the type or category of news,
  - a link to the original web site from which your archive was created, and
  - the name of the archivist (i.e., your name).

The image must be sourced from online (you cannot include image files in your solution apart from GIFs for the GUI). Since it will never change, the URL for this image can be "hardwired" in your Python code. The HTML source code generated by your Python program must be laid out neatly and the displayed document must adjust elegantly to the size of the web browser's window.

- **Generating ten news stories from archived web pages (8%).** Your Python program must be capable of generating an **entirely-new** HTML document from **any web page stored in the archive**, including the "latest" one, and **any new copy of the original web page downloaded from the same source**.

  Your archive must contain **at least seven substantially different web documents** previously downloaded from your chosen news source when you submit your solution.

  Your code **cannot be hardwired** to the specific web documents stored in the archive, and the stored web documents must be in **exactly the form they were retrieved from the web site** (otherwise your program won't work for "fresh" downloads of the latest news).

  The HTML document generated from *any* archived web page must contain the fixed elements described above, followed by at least the **first ten news stories** in the archived page. When viewed in a web browser, each story must contain at least the following elements:

  - the story number,
  - a link to the full version of the story,
  - the story's title (headline),
  - an image illustrating the story,
  - a short summary of the story, and
  - the date/time the story appeared online.

  The last four of these items must all be extracted from an archived document and **must all belong together** (i.e., you can't have an image from one story and the headline from another). Each of the elements must be extracted from the original document separately. It is *not* acceptable to simply copy large chunks of the original document's source code. The HTML source code generated by your Python program must be laid out neatly.

  The precise visual layout, colour and style of the generated elements is up to you, but the displayed new articles must be easy to read and the displayed document must adjust elegantly to the size of the web browser's window. No HTML markup tags or other odd characters should appear in any of the text displayed to the user.

  Data on the web changes frequently, so your solution **must continue to work even after the source web page has been updated**. For this reason it is unacceptable to "hardwire" your solution to the particular contents of web documents in your archive. You must allow for new copies of the source web page being added to the archive. Therefore you must use text searching functions and/or regular expressions to actively find the relevant elements in the archived pages, because you can't anticipate their specific contents.

- **Downloading and storing the latest news in the archive (1%).** Your program must be capable of downloading the current news from your chosen source web site and storing it as an HTML/XML file in the local `InternetArchive` folder in the same format as the documents already in the archive. To keep the size of the archive manageable only a single HTML/XML file source file can be stored. **No image files may be stored in the archive.**
- **Opening extracted news in the default web browser (1%).** Your program must be capable of opening the HTML file it generates in the host system's default web browser. It must work regardless of which operating system your program runs on. This can be done using the `getcwd`, `normpath` and `webopen` functions as explained below and in the provided `news_archivist.py` template file.
- **Providing an intuitive Graphical User Interface (4%).** Your application must provide an easy-to-use GUI. This interface must have the following features:
    - An image to identify the category or type of news stored in the archive. Tkinter only supports GIF images. The image file should be included in the same folder as your Python application.
    - Widgets to allow the user to control the archive effectively, i.e., to select and extract news, download and store the latest news, and display extracted news in the host system's default web browser. This can be achieved by a combination of menus, radio buttons, push buttons, etc.
    - The user interface must be robust to user error. There should be no way in which the user can cause it to "crash" (raise exceptions).
- **Python and HTML code quality and presentation (4%).** 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 code segment must be *clearly commented* to say what it does, e.g., "Extract the link to the photo", "Show the publication date", etc.
- ***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. For instance, an obvious improvement to this application would be to automatically extend the menu of archived documents shown in the GUI each time the "latest" news is downloaded.

You must complete the task using only basic Python features and the modules already imported into the provided template. However, your solution is *not* required to follow precisely our example shown above. Instead you are strongly encouraged to *be creative* in the your choices of web sites to access, the design of your Graphical User Interface, and the design of your generated news 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 five things for the first ten stories:

- The story's title.
- A link to an image associated with the story.
- A short summary of the story.
- The date/time the story appeared online.
- A link to the full version of the story online.

You also need to allow for the fact that the contents of the web pages from which you get your data will change regularly, so you cannot hardwire the locations of the document elements in your program. The solution to this problem is to use Python's string `find` method and/or regular expression function `findall` 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. `downloader.py` is a small Python script that downloads and saves the source code of a web document as a Unicode file. Use it to save the copies of your chosen web document for storage in your archive in *exactly* the form they are seen by Python programs. You may need to modify this "helper" program to suit your individual requirements. (Note that you should not submit this program with your final solution.)
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 the downloaded text from the web to help perfect your regular expressions. (There are also many online tools that do the same job.)

Internet ethics: Responsible scraping

The process of automatically extracting data from web documents is sometimes called "scraping". The RSS feeds we recommend using for this assignment are specifically intended to be easily "scrapable". However, in order to protect their intellectual property, and their computational resources, owners of some other 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, 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.

In this situation it's 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. 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.

Development hints

This is a substantial task, so you should not attempt to do it all at once. In particular, you should work on the "back end" parts first, designing your HTML document and working out how to extract the necessary elements for it from the archived web pages, before attempting

the Graphical User Interface "front end". 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.

It is suggested that you use the following development process:

1. Find a suitable source of online news articles. Keep in mind that the web site you choose must be updated daily, must have at least ten articles online at any time, and each article must have a heading, a short synopsis, a (link to a) photograph, a link to a full description of the story, and a publication date. Some starting points for finding such sites can be found in Appendix A below.
2. Create your "Internet Archive" by downloading seven copies of the web site, one per day. You should do so using the provided `downloader.py` program or a similar Python application so that the documents in your archive have the same structure that your application will see when downloading the "latest" news. (Otherwise you will have to write separate pattern matching code for extracting items from the archive and from the latest download!)
3. Study the HTML/XML source code of the archived documents to determine how the elements you want to extract are marked up. Typically you will want to identify the markup tags, and perhaps other unchanging parts of the document, that uniquely identify the beginning and end of the text and image addresses you want to extract.
4. Using the provided `regex_tester.py` application, devise regular expressions which extract just the necessary elements from the relevant parts of the archived web documents.
5. You can now develop a simple prototype of your "back end" function(s) that just extracts and saves the required elements from an archived web document.
6. Design the HTML source code for your "extracted" news stories, with appropriate placeholders for the downloaded web elements you will insert. Keep the document simple and its source code neat.
7. Develop the necessary Python code to extract the HTML elements from an archived document and create the HTML file. This completes the major "back end" part of your solution.
8. Develop a function to download the "latest" news and save it in the archive. (The provided `downloader.py` application is very helpful at this point!)
9. Develop a function to open a given HTML document in the host computer's default web browser. Importantly, this needs to be done in a way that will work on any computing platform. The provided template file contains hints for how to do this, using Python functions for finding the "path" to the folder in which the Python script resides (`getcwd`), for "normalising" full file names to the local computing environment (`normpath`), and for opening a file in the host system's default web browser (called `webopen` in our template).
10. Add the Graphical User Interface "front end" to your program. Decide whether you want to use push buttons, radio buttons, menus, lists or some other mechanism for choosing, extracting and displaying archived news. Developing the GUI is the "messiest" step, and is best left to the end.

Deliverables

You should develop your solution by completing and submitting the provided Python template file `news_archivist.py` together with a folder containing an "archive" of at least seven previously-downloaded web documents. These documents must be stored in a folder called `InternetArchive`. Your submission can also include one or more *small* GIF files needed to support your GUI interface, but **no other image files**. In particular, your "archive" folder may contain HTML/XML source code documents only, **no image files**.

1. Complete 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 not be marked.**
2. Complete your solution by populating your "archive" with downloaded web documents and developing your Python code. You must complete your solution using **only the modules imported by the provided template**. You do not need to use or import any other modules or files to complete this assignment.
3. Zip up your (a) Python program, (b) GIF files needed to support its GUI, and (c) the folder of archived web documents. **Upload the zip file** to Blackboard as a **single submission**.

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 (5:00pm on Wednesday, October 25th). Note that you will be able to submit as many drafts of your solution as you like. You are strongly encouraged to **submit draft solutions** before the deadline.

Appendix A: Some RSS feeds that may prove helpful

For this assignment you need to find a source of regularly-updated news or current affairs stories, containing a headline, a synopsis, a publication date, and links to a photo and a fully detailed story. You can choose any web site that has these features, but to simplify your task you should seek one that has a simple source-code format (when downloaded by a Python script). This appendix suggests some such sites, but you are strongly encouraged to find your own of personal interest.

The following links point to *Rich Site Summary*, a.k.a. *Really Simple Syndication*, web feed documents. RSS documents are written in XML and are used for publishing information that is updated frequently in a format that can be displayed by RSS reader software. Such documents have a simple standardised format, so we can rely on them always formatting their contents in the same way, making it relatively easy to extract specific elements from the document's source code via pattern matching.

Another important advantage of RSS feeds for our purposes is that such documents are specifically intended to serve as sources of online information for RSS readers and other such software, so they are unlikely to block Python scripts from accessing their contents (see Appendix B).

However, a disadvantage of using RSS feeds is that they can be hard to find! Often you can discover them only by looking for the RSS symbol at the bottom of web pages. Because RSS feeds are not intended for human consumption, they don't usually feature prominently in the results of web searches using standard search engines such as Google, DuckDuckGo, Bing, etc. You will need to do some exploration online to find a siuitable news feed for your solution.

Note that you are *not* limited to using RSS sites for this assignment, but you may find other, more complex, web documents harder to work with. Most importantly, you are *not* required to use *any* of the sources below for this task. You are strongly encouraged to find online documents of your own, that contain material of personal interest.

For our example solution above we used the ABC New's RSS feeds, but there are many other sites that may be suitable for this assignment. Some examples of RSS sites (or pages that point to such sites) containing news stories and embedded photo links include the following. We have *not* confirmed that these are all well-suited to the assignment. You will need to work that out for yourself.

- ABC News (as used in our sample solution): http://www.abc.net.au/news/feeds/rss/
- Wall Street Journal: http://www.wsj.com/public/page/rss_news_and_feeds.html
- BBC News: http://news.bbc.co.uk/2/hi/help/rss/default.stm
- The Daily Mail: http://www.dailymail.co.uk/home/article-2684527/RSS-Feeds.html
- The Orlando Sentinel: http://www.orlandosentinel.com/news/breaking-news/rss2.0.xml
- The Malaysian Star: http://www.thestar.com.my/rss/
- Sky News: http://news.sky.com/info/rss
- UPI news: http://rss.upi.com/news/news.rss

- The Seattle Times: http://www.seattletimes.com/rss-feeds/ (very large number of feeds)
- CNET News: https://www.cnet.com/rss/
- Sporting News: http://www.sportingnews.com/rss (e.g., http://www.sporting-news.com/rss/nascar)
- Apple Mac News and Rumours: http://feeds.macrumors.com/MacRumors-All (but note that the document format appears complex)
- Apple: https://www.apple.com/rss/ (mixed bag of feeds, some with photo links but no stories and vice versa; may not be suitable for the assignment)
- Times of India: http://timesofindia.indiatimes.com/rss.cms (another mixed bag; may not be suitable for the assignment)
- USA Today: http://rssfeeds.usatoday.com/usatoday-NewsTopStories
- TV Guide Breaking News: http://rss.tvguide.com/breakingnews (document format is complex)
- NASA: http://www.nasa.gov/content/nasa-rss-feeds (lots of good feeds but not all are updated daily)
- New York Times: http://www.nytimes.com/services/xml/rss/index.html (some feeds are updated irregularly and some don't have photo links for every story)
- 7 News Denver: http://www.thedenverchannel.com/rss/
- Yahoo7 Entertainment News: http://d.yimg.com/am/entertainment.xml
- The Guardian: https://www.theguardian.com/help/feeds (page format appears complex; note the instructions for creating a feed out of any section of the newspaper by adding "/rss" to the URL)
- US ABC News: http://feeds.abcnews.com/abcnews/topstories (lots of photos, but appears to have too little text associated with some stories)
- CNN: http://edition.cnn.com/services/rss/ (again, there may be too little text associated with the stories)
- New Zealand Herald: http://www.nzherald.co.nz/technology/news/article.cfm?c_id=5&objectid=11879231
- Wired Magazine: https://www.wired.com/feed/
- US Department of Defense: https://www.defense.gov/News/RSS/
- The Economist: http://www.economist.com/rss (complex document structure)
- NBC News: http://feeds.nbcnews.com/feeds/topstories
- Google news feeds: https://support.google.com/news/answer/59255?hl=en, e.g., https://www.usnews.com/info/features/rss-feeds (complex site; may not be suitable for this assignment)

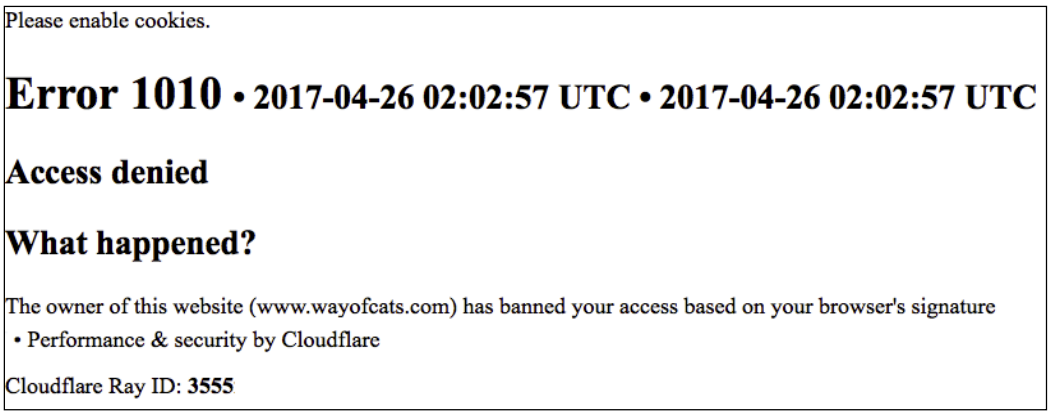
Finally, given that RSS feeds can be difficult to find using standard search engines, there are also sites specifically designed to help you locate feeds. Some examples are as follows. However, we have *not* tried these extensively ourselves, so cannot vouch for their quality.

- SpecificFeeds: https://www.specificfeeds.com/
- A searchable RSS directory: http://www.feedage.com/
- A list of several news sites: http://www.uen.org/feeds/lists.shtml
- Another long list of feeds: http://listofrssfeeds.com/

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. 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 different by the server.

If you suspect that your Python program isn't being allowed to access your chosen web page, use the small `downloader.py` application to check whether or not your Python program is being sent an access denied message. When viewed in a web browser, such messages 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.


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 web site owner's wishes.