Take-Home Task 2: DIY Periodical (15%, due October 14th)

Overview

Hardcopy periodicals such as newspapers, magazines, newsletters, etc. are all in decline as people increasingly turn to online media. Nonetheless, there is still a need for people to access regularly-updated information in an easy-to-read format. Here you will create an interactive user interface that presents topical information in a simple way, as a "do-it-yourself" periodical. Most importantly, your application will get its data from online "feeds" that are updated on a regular basis.

To do this you will need to develop (a) a "back end" function which fetches a Web document and finds relevant text and images in it, and (b) a "front end" Graphical User Interface that makes it easy for the reader to view such data.

To complete this task you will need to use Python, the Tkinter module, string functions and regular expressions and the Python Imaging Library. We have not used the last of these in IFB104 before, so you will need to install the necessary module and study some of its documentation yourself.

Example: A daily newspaper

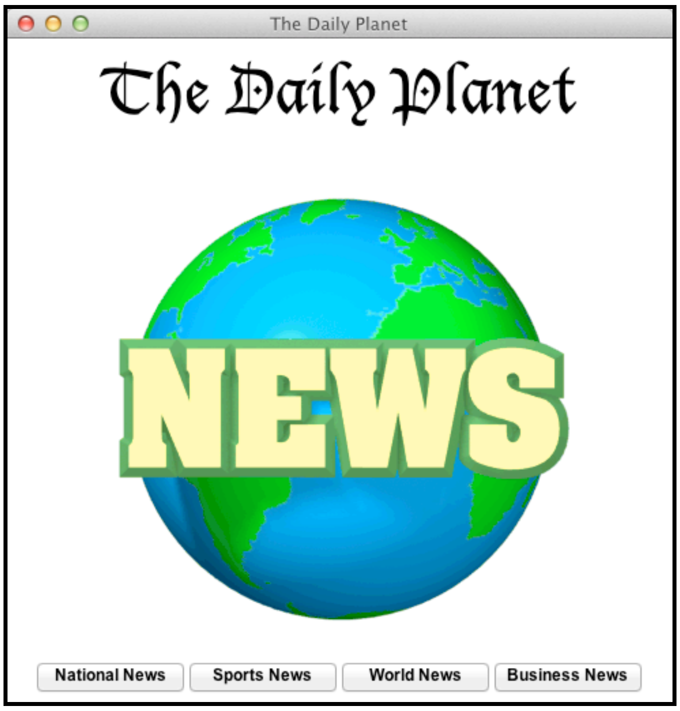
For the purposes of this task you have a totally free choice of what kind of periodical to produce. It could be a newspaper, a current affairs magazine, a hobbyist newsletter, etc. The important thing is that each of its "articles" must all be part of some consistent overall theme. Possible themes that you may consider could include:

- News
- Entertainment
- Politics
- Science and Technology
- Sports
- Games
- Fashion
- Lifestyle
- Et cetera

Whatever theme you choose, it must be a topic which is updated on a regular basis and for which information can be found readily on the Web. Your task is to extract data from online sources and display it in a simple format.

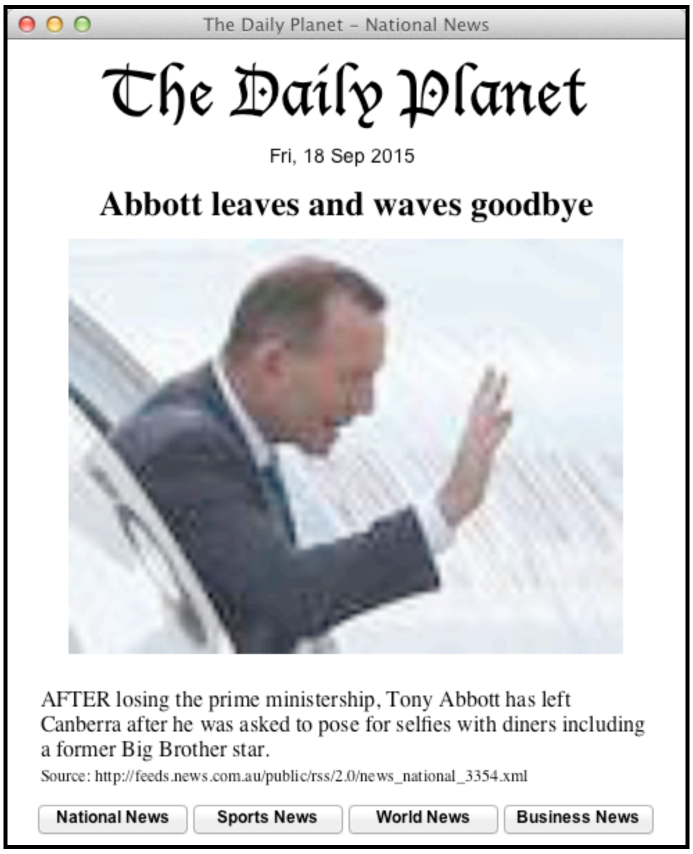
To demonstrate the idea, we have created an interactive newspaper called *The Daily Planet*. This periodical extracts data from News Limited's `news.com.au` Web site and presents it in a simplified format, showing the current lead item in four different categories, National News, Sports, World News and Business.

The screenshot below shows our application when it first starts. This is a "splash" page, waiting for the user to select a category.



The masthead is fixed text, and there are four interactive buttons at the bottom. In the centre is an image extracted from a Web page. (NB: Your solution must rely entirely on online sources for *all* images and data.)

When the first of the buttons is pushed, a headline, photo and associated story for the current top National News item is displayed, as shown below.



The Daily Planet – National News

The Daily Planet

Fri, 18 Sep 2015

Abbott leaves and waves goodbye

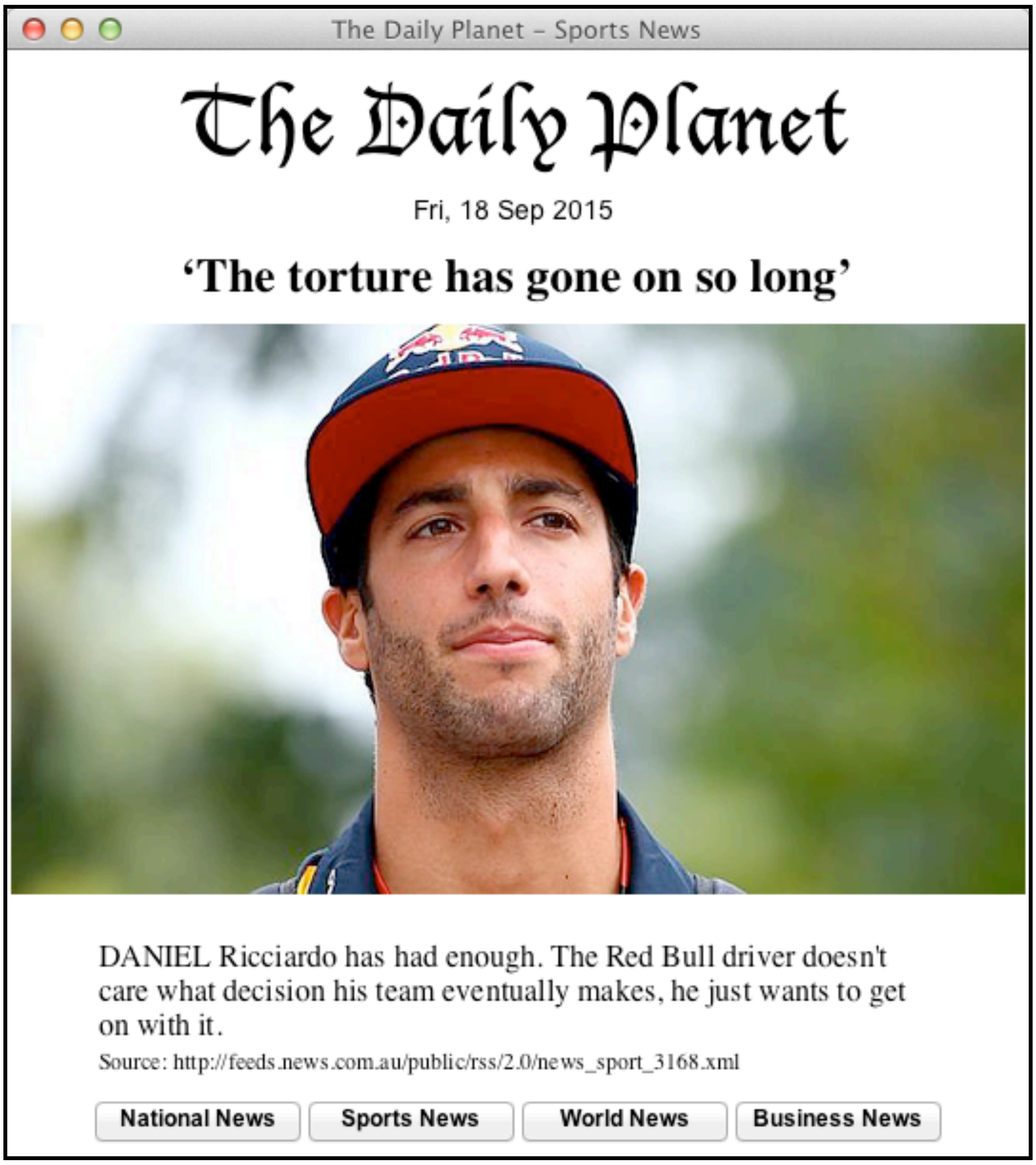
AFTER losing the prime ministership, Tony Abbott has left Canberra after he was asked to pose for selfies with diners including a former Big Brother star.

Source: http://feeds.news.com.au/public/rss/2.0/news_national_3354.xml

National News | Sports News | World News | Business News

Notice that the "newspaper" page displayed above contains the current date, a headline, photo, short story and the URL of the original Web document from which all this information was all obtained.

Similarly, pushing the Sports button on this occasion produced the following output.



Fri, 18 Sep 2015

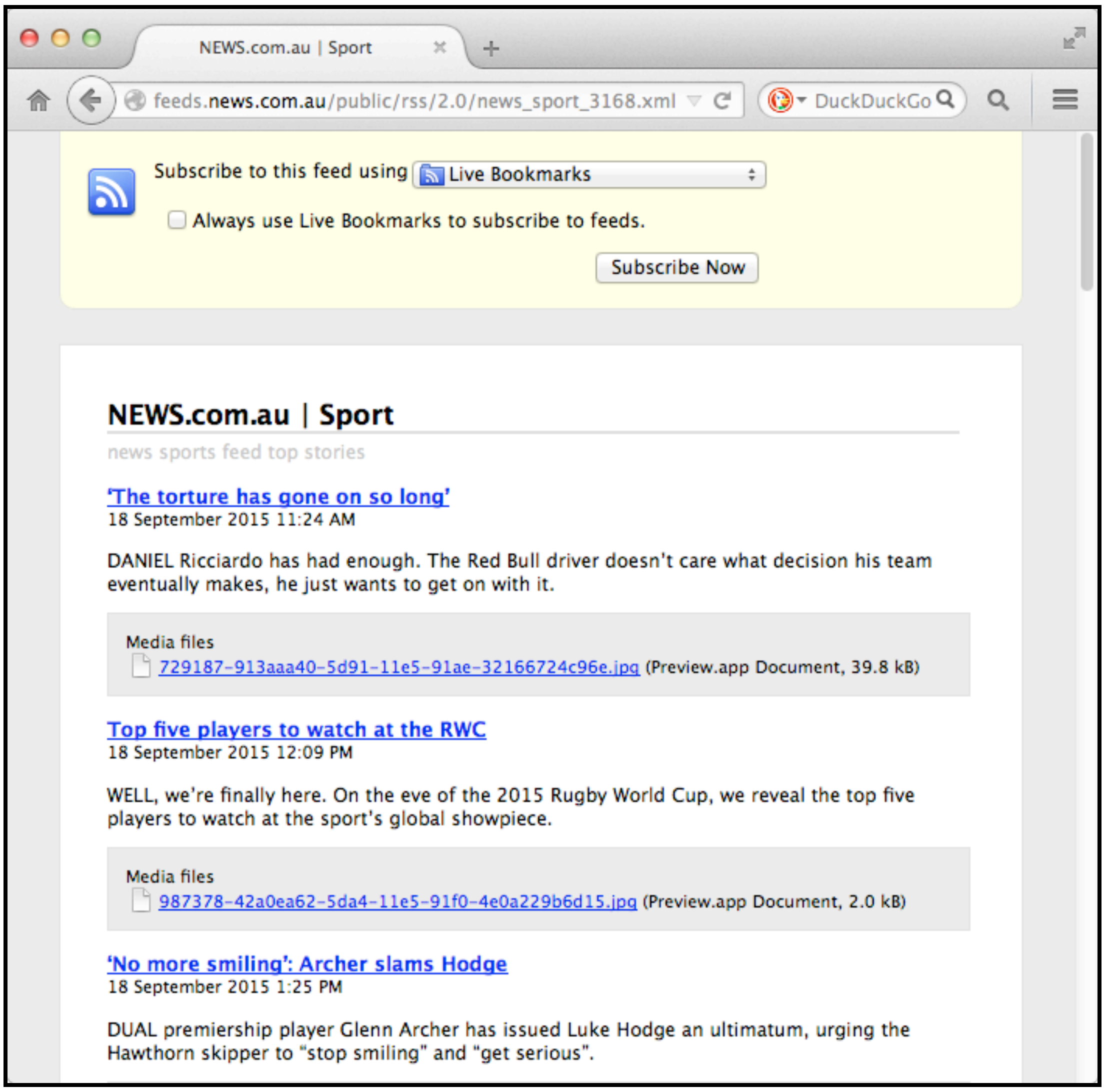
'The torture has gone on so long'

DANIEL Ricciardo has had enough. The Red Bull driver doesn't care what decision his team eventually makes, he just wants to get on with it.

Source: http://feeds.news.com.au/public/rss/2.0/news_sport_3168.xml

Here the headline, photo, story and URL have all been changed to our "sports page". In this second story the photograph is of a much higher resolution than in the first, but both were resized to the same height by our application for consistency.

As shown by the URL above, the actual source of this story is an XML document from the `feeds.news.com.au` Web site. When examined in a Web broswer, the original document from which the story was extracted is as shown below. You can see that the story we have displayed was the top one at the time.


NEWS.com.au | Sport
feeds.news.com.au/public/rss/2.0/news_sport_3168.xml
DuckDuckGo
Subscribe to this feed using
Live Bookmarks
Always use Live Bookmarks to subscribe to feeds.
Subscribe Now

NEWS.com.au | Sport

news sports feed top stories

'The torture has gone on so long'
18 September 2015 11:24 AM

DANIEL Ricciardo has had enough. The Red Bull driver doesn't care what decision his team eventually makes, he just wants to get on with it.

Media files
729187-913aaa40-5d91-11e5-91ae-32166724c96e.jpg (Preview.app Document, 39.8 kB)

Top five players to watch at the RWC
18 September 2015 12:09 PM

WELL, we're finally here. On the eve of the 2015 Rugby World Cup, we reveal the top five players to watch at the sport's global showpiece.

Media files
987378-42a0ea62-5da4-11e5-91f0-4e0a229b6d15.jpg (Preview.app Document, 2.0 kB)

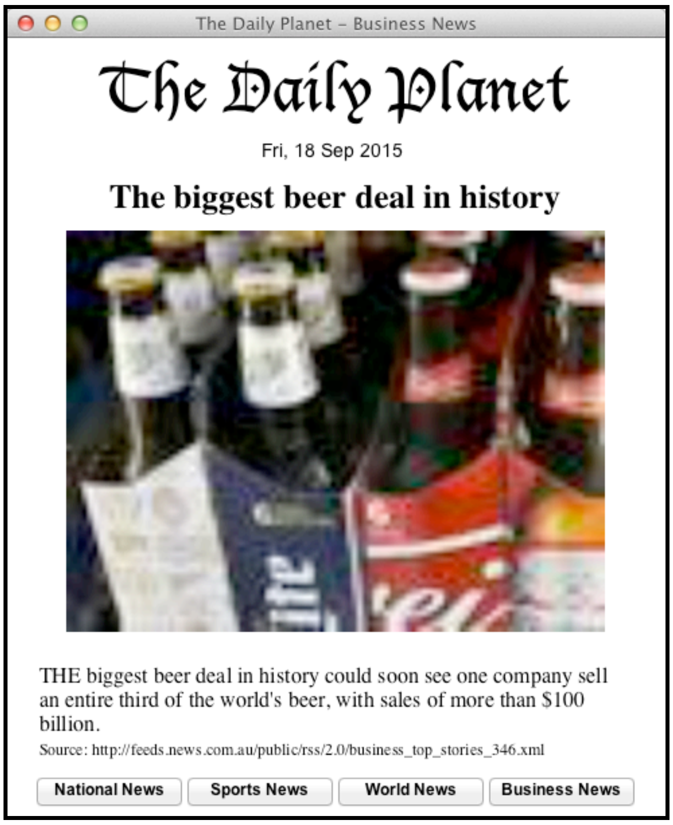
'No more smiling': Archer slams Hodge
18 September 2015 1:25 PM

DUAL premiership player Glenn Archer has issued Luke Hodge an ultimatum, urging the Hawthorn skipper to "stop smiling" and "get serious".

This is an example of a Rich Site Summary (RSS), Web-feed document. Such documents have a simple format which makes it easy for applications such as ours to extract data. Above we can see the list of sports stories available when our program was run, with the topmost one being the one we used to create our newspaper's Sports page. The headline and story text were extracted directly from the document's body and the photograph came from the address of the "media file" linked to the RSS page above. (The RSS document doesn't display the image, but it contains the URL at which the image can be found.)

To extract the necessary data we downloaded the RSS document as a character string and used a combination of Python string functions and regular expressions to isolate the parts of the XML source we needed to construct our own page. For instance, by examining the XML source of the document above we discovered that each news item appears between `<item>` … `</item>` tags, and that the short summaries of each story appear between `<description>` … `</description>` tags, so we used this knowledge to help extract the necessary text and image address.

Our solution does this each time one of the buttons is pressed. For instance, pressing the Business news button then generated the following story.



The Daily Planet

Fri, 18 Sep 2015

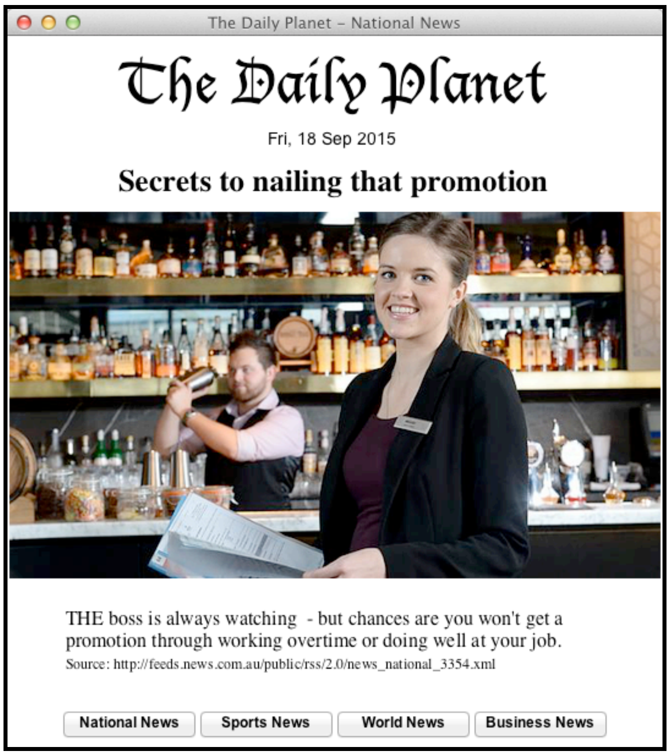
The biggest beer deal in history

THE biggest beer deal in history could soon see one company sell an entire third of the world's beer, with sales of more than $100 billion.

Source: http://feeds.news.com.au/public/rss/2.0/business_top_stories_346.xml

As with our first news story above, the photo attached to this story was a small "thumbnail" image, so we have enlarged it, hence the blurry appearance.

Most importantly, however, your application must react to changes in the original source document. Pressing the National News button again, after the RSS Feed had been updated, produced the following top story in our case, this time with a high-resolution image.



The Daily Planet

Fri, 18 Sep 2015

Secrets to nailing that promotion

THE boss is always watching - but chances are you won't get a promotion through working overtime or doing well at your job.

Source: http://feeds.news.com.au/public/rss/2.0/news_national_3354.xml

In this case the text attached to this particular story contained some HTML markups for special characters, such as `&#8212;` for a dash and `&#8217;` for a single quote. We therefore replaced these with the corresponding characters before displaying the text in our GUI.

Specific requirements

To complete this task you are required to develop an application in Python similar to that above, using the provided `diy_periodical.py` template file as your starting point. Your solution must have *at least* the following features.

- It must provide a Graphical User Interface which allows the reader to select from **at least four distinct pages** in your periodical, each containing the latest story in a particular category. All of the pages should belong to some consistent "theme" which should be indicated by your GUI's title.
- Each "page" of the GUI must have **at least six elements:**
  - a **masthead** telling the reader what periodical they are reading (and preferably giving some hint about its contents);
  - a **headline** telling the reader what the currently displayed story is about;
  - a **short summary** of the story or article;
  - an **image** related to the story;
  - a Web **address** (URL) telling the reader where the original data for the story cam from; and
  - one (or more) **interactive elements** which allow the reader to select different "pages" of the periodical.

  The headline, story, image and URL must all belong together (e.g., you can't have an image from one story and the headline from another). The precise layout, colour and style of these elements is up to you. Any mechanism can be provided for selecting pages as long as it is intuitive and easy to use. In the example above we have used push buttons to select different news categories, but you can use **any kind of selection mechanism** you like, such as radio buttons, lists, menus, "spinboxes", etc.
- Your GUI must present the reader with **up-to-date text and images** from a particular Web document or documents. The stories must come from Web documents which are **updated regularly**, preferably at least once a day. It is *not* acceptable to base your solution on static Web documents that do not change their content for long periods of time. At the very least the source documents must be freshly downloaded from the Web whenever your application is started. Even better would be to download them each time the reader selects a new page.
- The overall **size, proportions and layout of your GUI must not change** significantly as new pages are displayed. It's very disconcerting for readers if the position of widgets changes as they use an application. To achieve this **all the images must be presented at about the same size**, so if your source Web document provides images of varying sizes you will need to resize or crop them to fit your GUI. In our example above we resized the images to all have the same height. Note that if you resize images they must **retain their original proportions**. In other words, the images must not be distorted either horizontally or vertically.

- Data on the Web changes frequently, so your solution **must continue to work even after the Web documents you use have been updated**. For this reason it is unacceptable to "hardwire" your solution to the particular text and images appearing on the Web on a particular day. Instead you will need to use text searching functions and regular expressions to actively **find the text and images** in the document, regardless of any updates that may have occurred since you wrote your program.
- The user interface must be **well-presented and easy to use**. It would be unacceptable, for instance, to display HTML markup tags in the stories, or to expect the user to type in a URL to change the page displayed.
- As usual your program **code must be presented in a professional manner**. See the coding guidelines in the *IFB104 Code Marking Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this.

You can add other features if you wish, as long as you meet these basic requirements. For instance, in our example above we opened the application with a "splash" image which was extracted from a different Web page than the one we used for the stories. We also displayed the current date (extracted from the Web documents rather than generated by the Python program).

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

Support tools

To get started on this task you need to download your Web document, or documents, of choice and work out how to extract three things:

- The title of the current top "story".
- The text of the story itself.
- The address of a photograph associated with the story.

You also need to allow for the fact that the contents of the page will be updated occasionally, so you cannot hardwire the locations of these document elements in your solution. The answer to this problem, of course, is to use Python string functions such as `find` and the regular expression function `findall` to extract the necessary elements, no matter where they appear in the text.

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

1. `downloader.py` is a small script that downloads and displays the contents of a Web page. Use it to see a copy of your chosen Web page in exactly the form that it will be received by your Python program. (This is helpful because the version of a Web document opened by a Python program may not be the same as one opened by a particular Web browser! This can occur because some Web servers produce different HTML/XML code for different clients.)
2. `regex_tester.py` is an interactive program introduced in the Week 9 lecture 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 you perfect your regular expressions. (The advantage of using this tool,

rather than some of the online tools available, is that it is guaranteed to follow Python's regular expression syntax because it is written in Python itself.)

Image formats and the Python Imaging Library

Image files come in a wide variety of formats, including GIF, JPG, PNG, BMP, and so on. Python's Tkinter module can display 'photo images' but these must be in GIF format encoded as base-64 character strings. For instance, Python code that will download a GIF image from a particular URL of the form '`http://....gif`' in a format ready for displaying in Tkinter is as follows. Here we are using the `PhotoImage` constructor from Python's `Tkinter` module. Greek letters have been used for the Python variables below; obviously you should replace these with *meaningful* variable names. Let α be a string containing the URL.

```
# Read the image as a raw byte stream
β = urlopen(α).read()

# Encode the byte stream as a base-64 character string
γ = β.encode('base64', 'strict')

# Create the Tkinter version of the image
δ = PhotoImage(data = γ)
```

This code accesses the Web page at the given URL, reads its entire contents as a byte stream, encodes the stream as a 64-bit character string, and then converts this to Tkinter's 'photo image' format. The resulting variable δ can then be used as the `image` attribute in a Tkinter `Label` widget or any other such widget that can display images.

You will find, however, that most images on the web are in JPEG, PNG or some other format and therefore cannot be used in Tkinter without conversion. If you are unable to find a suitable page containing GIF images only you will need to use an additional module such as the *Python Imaging Library* to convert the image from its original format to GIF.

The Python Imaging Library is a well-established (stable since 2009) toolkit for manipulating image data in Python programs. It contains a large number of functions for converting images, resizing them, rotating them, changing colours, etc.

- To use it, Microsoft Windows users should download and install PIL version 1.1.7 for Python 2.7. Download the library from http://www.pythonware.com/products/pil/ and select the installation for Python 2.7.
- Alternatively, Mac OS X users should install disk image PIL-1.1.7-py2.7-python.org-macosx10.6.dmg. You can find a copy at http://www.astro.washington.edu/users/rowen/python/.

We have not used the PIL library previously in this unit, so if you decide you need to use it, part of this task is to learn its basic features for yourself. PIL documentation can be found at http://effbot.org/imagingbook/ or see the copy of the PIL manual enclosed with these instructions.

To convert a non-GIF image into Tkinter's format using PIL, the process is similar to the one above, except that we use functions and constructors from the `PIL` module's `Image` and `ImageTk` classes. For instance, if we have the URL of a photograph in JPEG format, '`http://....jpg`', then we can convert it to a format ready for use in a Tkinter widget as follows. Greek letters have been used for the Python variables below; obviously you should replace these with *meaningful* variable names. Let α be a string containing the URL.

```
# Read the image as a raw byte stream
β = urlopen(α).read()

# Use Python's StringIO constructor to convert the bytes into characters
γ = StringIO(β)

# Open the character string as a PIL image
δ = Image.open(γ)

# Create the Tkinter version of the image
ε = ImageTk.PhotoImage(δ)
```

This code first downloads the image and reads its binary contents into the variable β. PIL requires images to be represented as character strings, so we then use the `StringIO` constructor to convert the raw bytes into characters and open this as a PIL image using the `open` function from PIL's `Image` class. Finally, we convert the PIL image into the specific GIF-based 'photo image' format recognised by Tkinter, using the `PhotoImage` constructor from PIL's `ImageTk` class (not to be confused with the `PhotoImage` constructor in the `Tkinter` module). The resulting variable ε can then be used as the `image` attribute in a Tkinter `Label` widget or any other such widget that can display images.

Importantly, note that after the third step above you have an opportunity to reformat the PIL image, e.g., to resize it, crop it, or perform other transformations using the appropriate PIL methods. See the PIL documentation for more detail about these functions.

Updating images in widgets

One of the challenges in this task is to update the image associated with an existing widget. There are a number of ways to achieve this, but the simplest is to entirely "destroy" the existing widget and replace it with a new one containing the updated image. To do this you can use *widget*.`destroy()` to remove the old widget entirely. Alternatively, if you used the 'grid' geometry manager to place your widgets in the window you can also use *widget*.`grid_forget()` to remove the widget from the grid but without destroying the widget itself.

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, 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 strongly suggested that you use the following development process:

1. Decide what kind of periodical you want to create and search the Web for appropriate HTML or XML documents that contain the necessary text and images. Choose only documents that are updated regularly, preferably at least once a day.
2. Using the provided `downloader.py` application, download each document so that you can examine its structure. You will want to study the HTML or XML source code of the document to determine how the parts 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.

3. Using the provided `regex_tester.py` application, devise regular expressions which extract just the necessary elements from the relevant parts of the Web document. Using these regular expressions and Python's `urllib` module and `findall` function you can now develop a simple prototype of your "back end" solution that just extracts and prints the required data, i.e., the text and the addresses of the images, from the Web document(s) in IDLE's shell window. Doing this will give you confidence that you are heading in the right direction, and is a useful outcome in its own right.
4. Once you have got the data extraction functions working you can add a 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 which pages of your periodical to display, and extend your back-end code accordingly. Developing the GUI is the "messiest" step, and is best left to last.

Deliverables

You should develop your solution by completing and submitting the provided Python template file `diy_periodical.py` as follows.

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 developing Python code at the place indicated. You must complete your solution using **only the modules imported by the provided template**. You may not use or import any other modules to complete this program.
3. Submit **only a single, self-contained Python file**. **Do *not* submit multiple files. Do *not* submit an archive containing multiple files.** This means that all images you want to display must be downloaded from an online Web page, not from a local file.

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

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 will be created on Blackboard under *Assessment* for uploading your solution file close to the deadline (midnight on Wednesday, October 14th).

Appendix: Some RSS feeds that may prove helpful

For this task you need to find one or more documents on the Web that contain regularly-updated stories and links to images, and that have a fairly simple format you can search through. This appendix suggests some pages, but you are encouraged to find your own.

The best source of such data is *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. They usually 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.

There are many RSS Feeds for daily news, including the following.

- The Australian newspaper: http://www.theaustralian.com.au/help/rss
- SBS news: http://www.sbs.com.au/news/rss-list
- The New York Times: http://www.nytimes.com/services/xml/rss/index.html
- Yahoo7 (Channel 7) news: https://au.news.yahoo.com/rss
- The Sydney Morning Herald: http://www.smh.com.au/rssheadlines
- NDTV: http://www.ndtv.com/rss
- Sky News: http://news.sky.com/info/rss
- The Courier Mail: http://www.couriermail.com.au/help/rss

RSS Feeds are used for purposes other than general news, however, so you can find RSS sites dedicated to specialised topics that would make an ideal basis for a "magazine" style application. Some sources follow but there are many more. Not all such sites contain both stories and images, however. Also you may find that some RSS documents without images point to other pages with some, so you may want to consider developing a solution that follows the links to accumulate the necessary headlines, stories and images.

- NASA: http://www.nasa.gov/content/nasa-rss-feeds
- Apple: https://www.apple.com/rss/
- A general list of RSS feeds: http://www.uen.org/feeds/lists.shtml
- A list of feeds for software developers: https://github.com/impressivewebs/frontend-feeds

Most importantly, you are *not* required to use one of the sources above for this task. You are strongly encouraged to find online documents of your own, that contain material of personal interest.