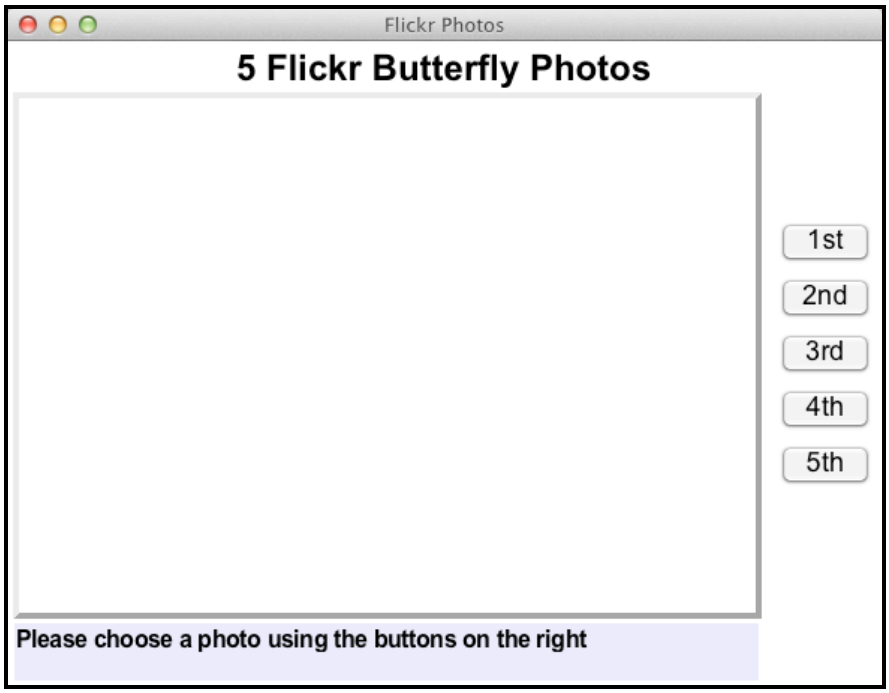
Take-Home Task 2: Photo Browser (15%, due May 20th)

Overview

There are many Web pages that give you access to collections of photographs. Here you will create an interactive *Photo Browser* application that makes it easy to browse such a collection by showing the photos to the user one at a time, via an intuitive user interface. To do this you will need to develop (a) a "back end" function which fetches a Web page and finds photos and associated captions in it, and (b) a "front end" Graphical User Interface that makes it easy for the user to navigate the results. To complete this task you will need to use Python, Tkinter, regular expressions and (probably) the Python Imaging Library.

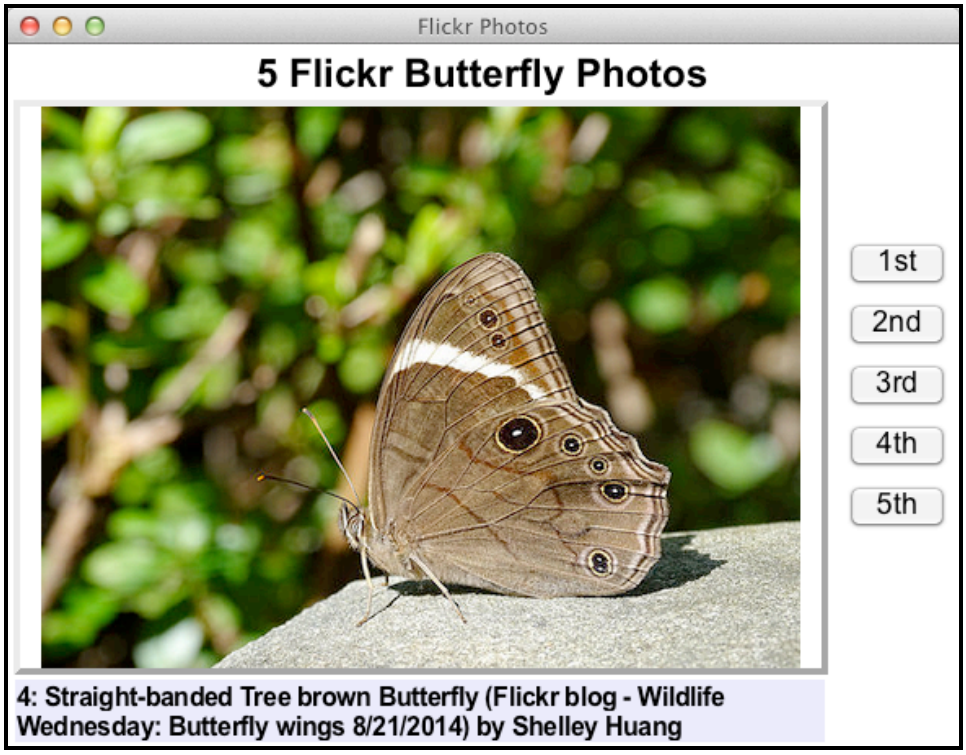
Example 1: A basic version

The example in this section shows a solution which meets the *minimum* requirements for this task. (The second version below is a better, more elaborate solution.) In this basic version the application opens with a blank interface as follows.



The user interface has a heading, a large empty area for displaying images, a text area at the bottom for displaying captions, and five push buttons on the right. When the first of the buttons is pushed, a photo and an associated caption are displayed as shown overleaf.

Similarly, pushing the fourth button produces a different photo.

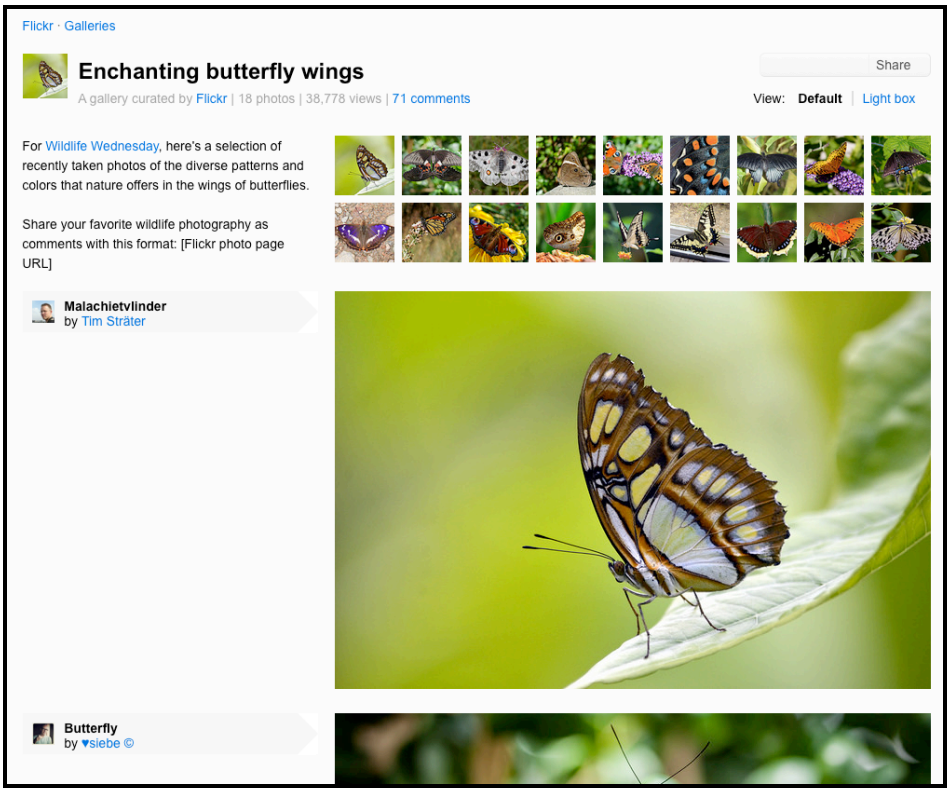


In this case the original photo has a different width and height than the first one, but it has been resized to fit within the same area.

But where do the photos and their captions come from? In this case they were downloaded from a *Flickr* page which can be found at the following address:

https://www.flickr.com/photos/flickr/galleries/72157646624780266/

The actual Web document begins as follows:



Here you can see the first of the photos produced by our photo browser application and its associated caption on the left. Importantly, if the original web page is updated with new photos then the images and captions displayed by our application change accordingly; our basic *Photo Browser* application reloads the contents of the web page each time it is started.

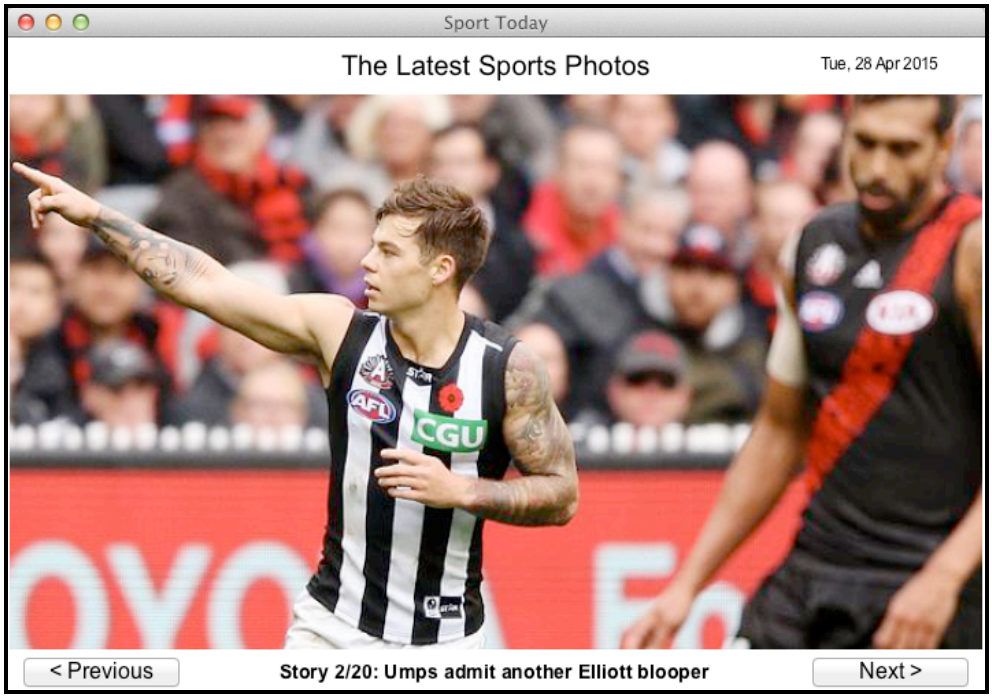
Example 2: A better version

Although the example above meets the minimum requirements for this task there are many ways in which it could be improved and you are free to embellish your solution in any way you like, as long as it still meets the basic specification.

As a second, better version of the solution, below is the opening screen for an application that allows us to browse the latest news photos from the world of sport. In this case there is an opening "splash" image, as well as the date taken from the Web document containing the photos. There is also a numerical indicator showing how many photos are available in total (20 in this case).

The user is provided with two buttons for navigation, Previous and Next, the first of which is initially disabled because there is no previous photo. Pressing the Next button produces the first photo and its caption from the Web page used:

Pressing the Next button again displays the second photo and also enables the Previous button in case we wish to go back to the previous image:

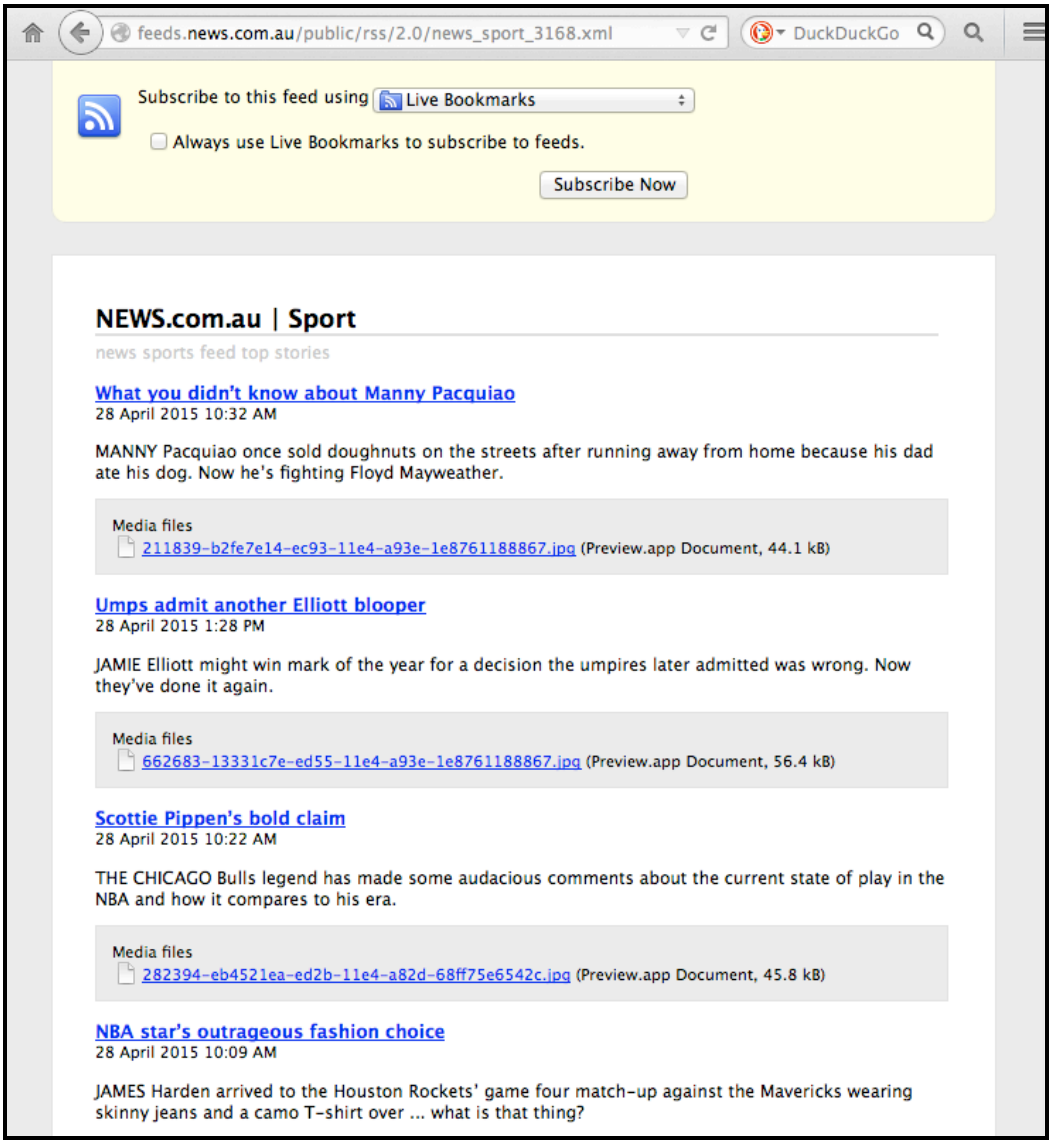


The images on the original Web page are of varying sizes in this case. However, our application resizes them to fit our window appropriately. In some cases this results in low-resolution images being greatly enlarged, as in the following example:

Importantly, this version allows us to browse *all* the images on the Web page, not just the first five. In this case the photos and captions were extracted from a Rich Site Summary (RSS), Web-feed document:

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

The original Web page appears as shown below when opened in a Web browser. Notice that, unlike the Flickr page above, this document provides *links* to the photos, but does not display them, so our *Photo Browser* application is a useful tool for readers wanting to explore these sports stories:



Although not obvious above, another improvement in our second *Photo Browser* solution is that the original Web page is reopened and reprocessed every time a button is pressed, meaning that the photos displayed are always up-to-date, no matter how long the application has been running for.

Specific requirements

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

- It must provide a Graphical User Interface with at least **four widgets**,
  - a **heading** to tell the user what the photos are about,
  - a **widget that supports images** for showing the photographs,
  - a **text** area to display the captions for the photos, and
  - one (or more) **interactive elements** to allow the user to browse the photos.

  The precise layout, colour and choice of the widgets is up to you. In the examples above we have used push buttons to select the photos, but you can use **any kind of selection mechanism** you like, such as radio buttons, lists, menus, "spinboxes", etc.
- It must present the user with up-to-date **photographs and captions** from a particular Web document. The category of photo is your choice, but could be one of the following:
  - News stories
  - Sporting event news
  - Television show or movie reviews or news
  - Webcam images (for weather, traffic, surf conditions, etc)
  - Any other regularly-updated collections of photographs

  The stories must come from a single Web page which is **updated regularly**, preferably 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.
- The chosen document must predominantly contain **full-size photos**. Web pages containing small 'thumbnail' sized images only are not suitable for our purposes.
- The photographs and captions displayed must be **"live"**, i.e., they must be freshly downloaded from the Web whenever your application is started. (Even better would be to download them each time the user interacts with your application.)
- Your application must allow the user to choose between **at least five** distinct photographs and captions from the Web page. (Ideally, it should allow the user to choose between *all* the photographs on the page.)
- The size, proportions and layout of your GUI **must not change** as new images are displayed, and all the photographs must be presented at the **same size**. It's very disconcerting for users if the position of widgets changes as they use an application so, even though the photos you download may be of different sizes, the GUI as a whole must not change its proportions as new images are shown.

- The photographs displayed must **retain their original proportions**, even if they have been resized for display in the GUI. In other words, the photos 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 page you use has been updated. For this reason it is unacceptable to "hardwire" your solution to the particular photographs and captions appearing on the page on a particular day. Instead you will need to use regular expressions, or some equivalent pattern matching method, to actively **search for the photographs and captions** in the document, regardless of any updates that have occurred since you wrote your program.
- The user interface must be **intuitive and easy to use** and the output must be presented in a clearly legible form. It would be unacceptable, for instance, to display HTML markup tags in the captions, or to expect the user to type in a URL or similar address to change the photograph 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 second 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 collection of photographs.

However, your solution is **not** required to follow precisely either of the examples shown above. Instead you are strongly encouraged to **be creative** in both your choice of photographs to display and the design of your Graphical User Interface.

Support tools

To get started on this task you need to download your web page of choice and work out how to extract two things from it:

- The addresses of the photographs it contains or references.
- The captions associated with these photos.

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 regular expressions and Python's `findall` function to extract the elements.

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 a Web page and stores it in a text file. Use it to get 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 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 use 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 since it's 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.

```
img = urlopen(url).read().encode('base64', 'strict')
tk_img = PhotoImage(data = img)
```

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 *tk_img* can then be used as the `image` attribute in a Tkinter `Label` widget or any other such widget that can display images. (Obviously you should choose more meaningful variable names than those used above!)

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 toolkit for manipulating image data in Python programs. (It has been stable since 2009.) It contains a large number of functions for converting images, resizing them, rotating them, 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 in the usual way. 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 open an image in a format other than GIF 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`', them we can convert it to a format ready for use in a Tkinter widget as follows.

```
raw_img = urlopen(url).read()
pil_img = Image.open(StringIO(raw_img))
tk_img = ImageTk.PhotoImage(pil_img)
```

This code first downloads the image and reads its binary contents into the variable *raw_img*. 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. Finally, we convert the PIL image into the specific GIF-based 'photo image' format recognised by Tkinter.

Importantly, note that after the second 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 delete 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 "live" photographs you want to display and search the Web for an appropriate HTML or XML document that contains the necessary photographs and captions. Choose only a document that is updated regularly, preferably once a day, and contains numerous full-sized photos.
2. Using the provided `downloader.py` application, download the document into a text file so that you can examine its structure in a text editor. 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 page, that uniquely identify the beginning and end of the photographs and captions you want to extract.

3. Using the provided `regex_tester.py` application, devise regular expressions which extract just the photographs and just the captions from the page. 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 captions and the addresses of the photographs, from the Web page 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 photographs 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 `photo_browser.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 an archive containing multiple files. This means that any graphics or text 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, May 20th).

Appendix: Some Web pages and RSS feeds that may prove helpful

For this task you need to find a document on the Web that contains regularly-updated photos and captions, and that has a fairly simple format you can search through. This appendix suggests some pages, but you are encouraged to find your own. Note that pages of small 'thumbnail' images are not suitable for this task; we want to display full-size images.

An obvious source of photographs is a Web site such as *Flickr*, which we used for our first example above. Some Flickr pages containing numerous photographs and captions include the following.

https://www.flickr.com/photos/flickr/galleries/72157646624780266/

https://www.flickr.com/photos/flickr/galleries/72157646624780266/

https://www.flickr.com/photos/flickr/galleries/72157649608126894/

https://www.flickr.com/photos/flickr/galleries/72157644585865271/

https://www.flickr.com/photos/flickr/galleries/72157644537473411/

Be warned, however, that the HTML source of these pages is quite complicated, making searching through them a challenge.

A simpler 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 presenting their data in the same way, making it relatively easy to extract specific elements from the document's source code.

Some examples of such pages, including the one we used for the second example above, are listed below.

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

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

http://feeds.news.com.au/public/rss/2.0/news_theworld_3356.xml

http://www.abc.net.au/news/feed/51120/rss.xml

http://www.abc.net.au/news/feed/46182/rss.xml

http://www.abc.net.au/news/feed/46800/rss.xml

However, you are **not required to use one of these pages**. You are strongly encouraged to find an online document of your own, that contains photographs of personal interest.