Assignment 2, Part A: Print Your Own Periodical

(17%, due 11:59pm Friday, October 21st)

Overview

This is the first part of a two-part assignment. This part is worth 17% of your final grade for IFB104. Part B will be worth a further 8%. Part B is intended as a last-minute extension to the assignment, thereby testing the maintainability of your solution to Part A and your ability to work under time pressure. The instructions for completing Part B will not be released until Week 12. Whether or not you complete Part B you will submit only one file, and receive only one mark, for the whole 25% assignment.

Motivation

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 develop a program that produces a customised periodical in HTML format, using data downloaded from the World-Wide Web. The program will have a Graphical User Interface that allows the user to control production of the periodical, which can then be viewed in a standard web browser. Most importantly, your publication will comprise up-to-date data sourced from online "feeds" that are updated on a regular basis. To complete this assignment you will need to: (a) download web pages in Python and use regular expressions to extract particular elements from them, (b) create an HTML file containing the extracted elements, and (c) use Tkinter to provide a simple Graphical User Interface.

Illustrative Example

For the purposes of this task you have a totally free choice of what kind of periodical to produce. It could be:

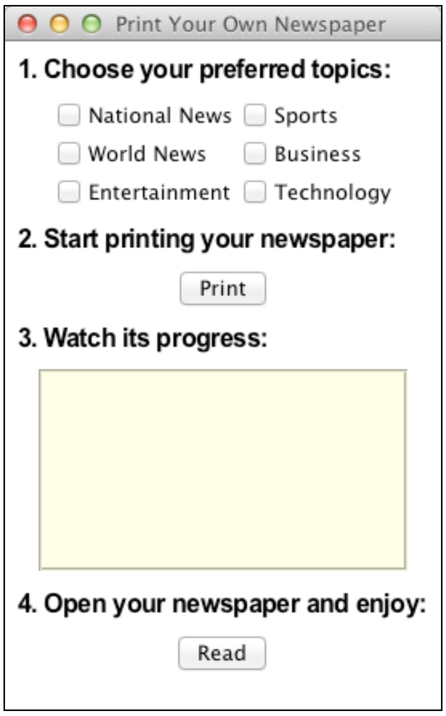
- a newspaper,
- a current affairs magazine,
- a fashion/lifestyle magazine,
- a newsletter for online gamers,
- a sports journal,
- a science and technology review,
- etc.

However, whatever theme you choose, you must be able to find at least four different online web pages that contain regularly-updated stories or articles in different categories under the overall theme. Each such story must contain a heading, a photograph, some text and a publication date. A good source for such data is Rich Site Summary (RSS) web-feed documents. The appendix below lists some such sites, but you are encouraged to find your own of personal interest.

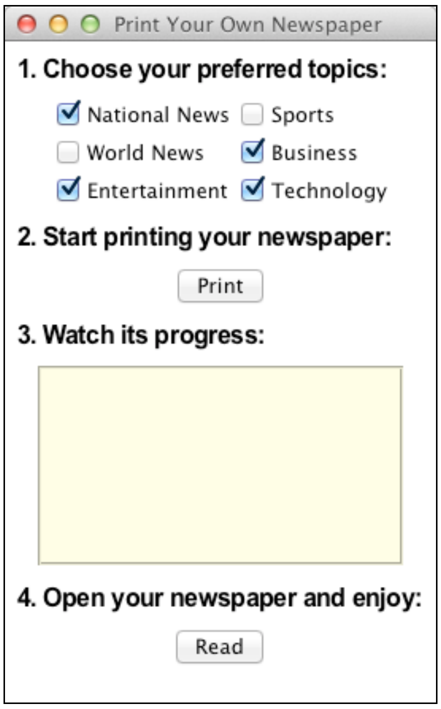
To demonstrate the idea, we will publish our own newspaper, using data extracted from News Limited's `news.com.au` web site. Our demonstration program allows users to select from several categories, National News, Sports, World News, Business News, Entertainment and

Technology. The program then downloads relevant data from the Web and uses it to produce an HTML document which can be read in a standard web browser.

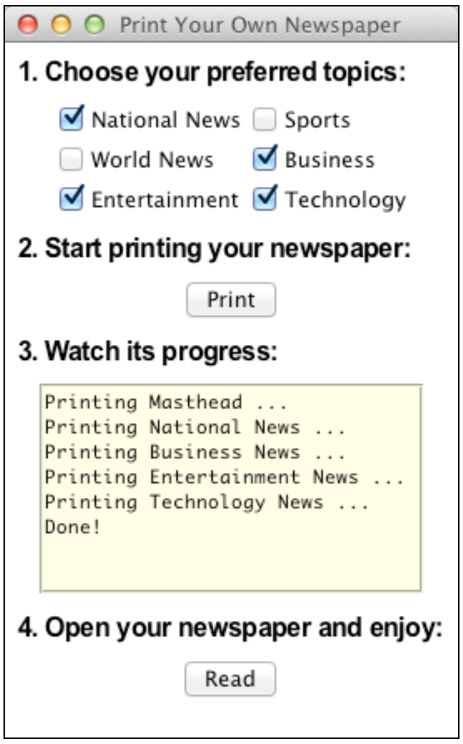
The screenshot below shows our example solution's GUI when it first starts.



The user is invited to select which categories of information they want included in their newspaper. In this case this is done by selecting check buttons, but other solutions are possible. Below the user has selected four news categories of interest.



When ready the user then presses the button to start "printing" the newspaper (i.e., to create an HTML file containing its contents). The system downloads current data from the `news.com.au` web site and generates the file. The user can follow the "printing" process's progress in the small text window.

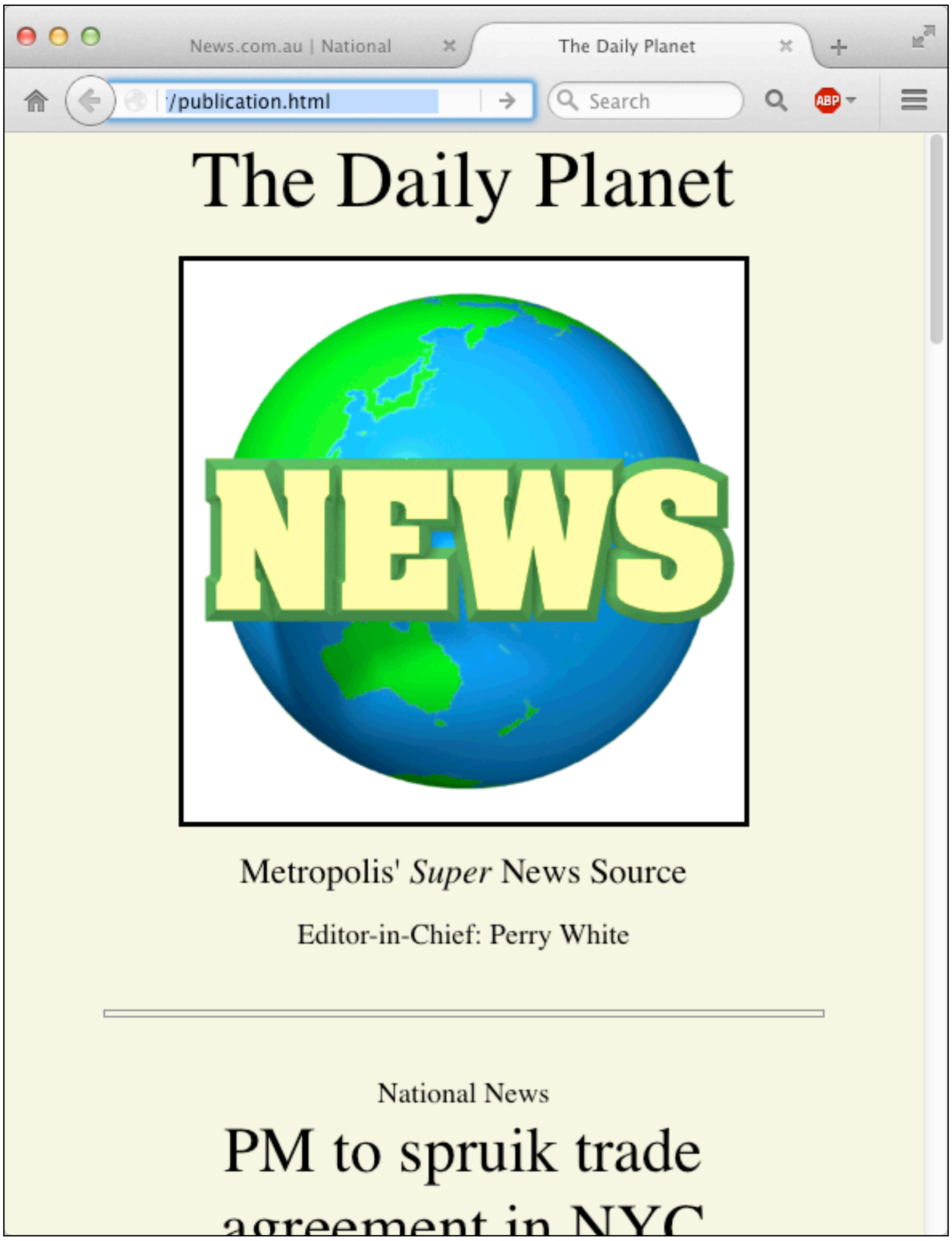


```
Printing Masthead ...
Printing National News ...
Printing Business News ...
Printing Entertainment News ...
Printing Technology News ...
Done!
```

As well as printing the latest top news items in each of the four categories, the system also generates a "masthead" which identifies the periodical.

Once the file has been created the user can open it in their preferred web browser. Alternatively, pressing the "Read" button in the GUI above will open the file in the host operating system's default browser.

The generated document contains the masthead and the current top story in each of the selected categories. It is shown overleaf as viewed in the Firefox browser.



The Daily Planet

Metropolis' *Super* News Source

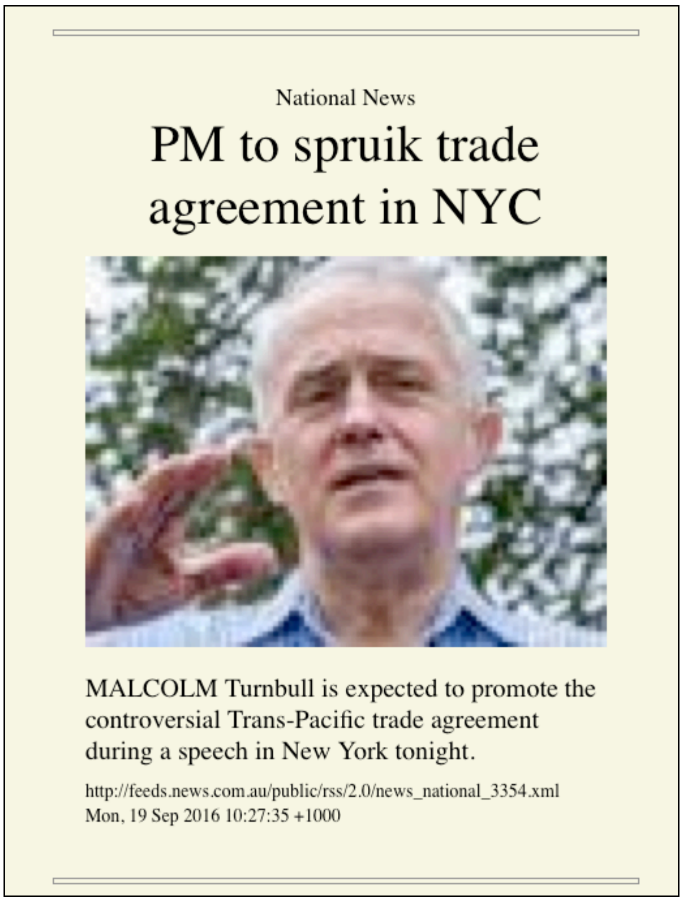
Editor-in-Chief: Perry White

National News

PM to spruik trade agreement in NYC

Above you can see the masthead with the name of the periodical, *The Daily Planet* in this case, and an image indicating the nature of its contents. (Our fictional newspaper's slogan and editor are also shown, but these features are optional.)

Scrolling down in the HTML document shows the current top news item in each of the four selected categories when the program was run. Three of these are shown below, as they were when this demonstration was run. (Some of the images downloaded at this time were small "thumbnails", hence their blurry appearance when enlarged.)

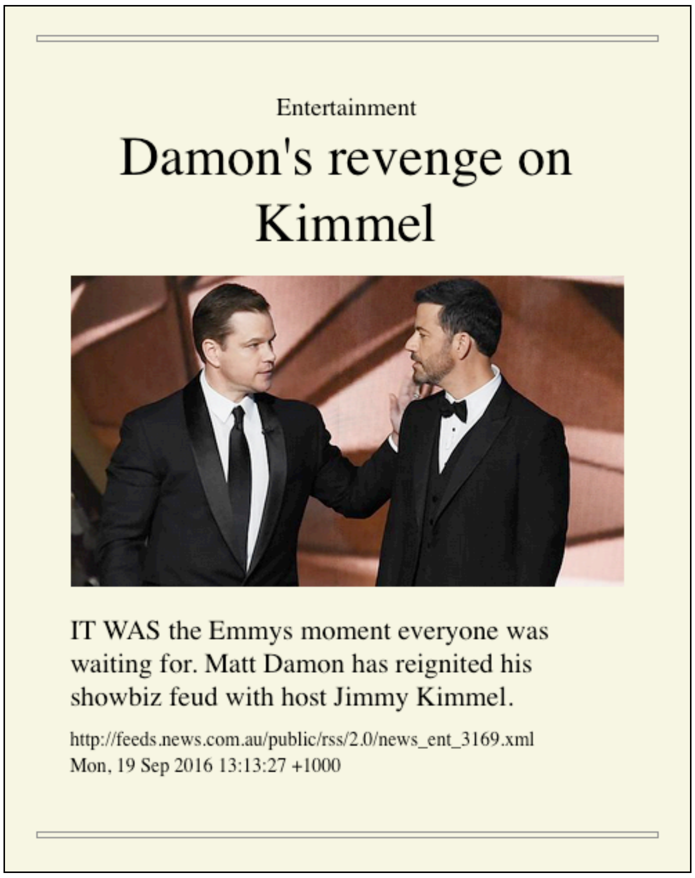


National News

PM to spruik trade agreement in NYC

MALCOLM Turnbull is expected to promote the controversial Trans-Pacific trade agreement during a speech in New York tonight.

http://feeds.news.com.au/public/rss/2.0/news_national_3354.xml
Mon, 19 Sep 2016 10:27:35 +1000

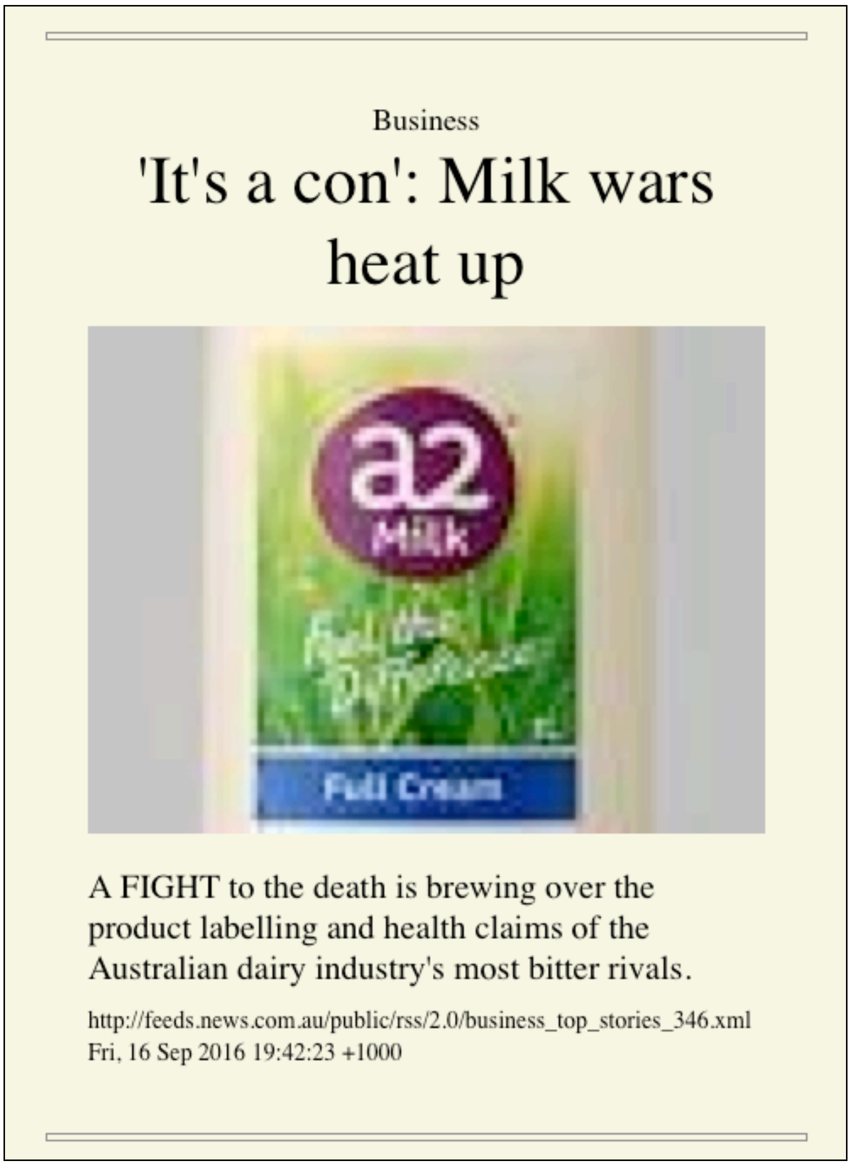


Entertainment

Damon's revenge on Kimmel

IT WAS the Emmys moment everyone was waiting for. Matt Damon has reignited his showbiz feud with host Jimmy Kimmel.

http://feeds.news.com.au/public/rss/2.0/news_ent_3169.xml
Mon, 19 Sep 2016 13:13:27 +1000



Business

'It's a con': Milk wars heat up

A FIGHT to the death is brewing over the product labelling and health claims of the Australian dairy industry's most bitter rivals.

http://feeds.news.com.au/public/rss/2.0/business_top_stories_346.xml
Fri, 16 Sep 2016 19:42:23 +1000

Notice that each top news item displayed above contains:

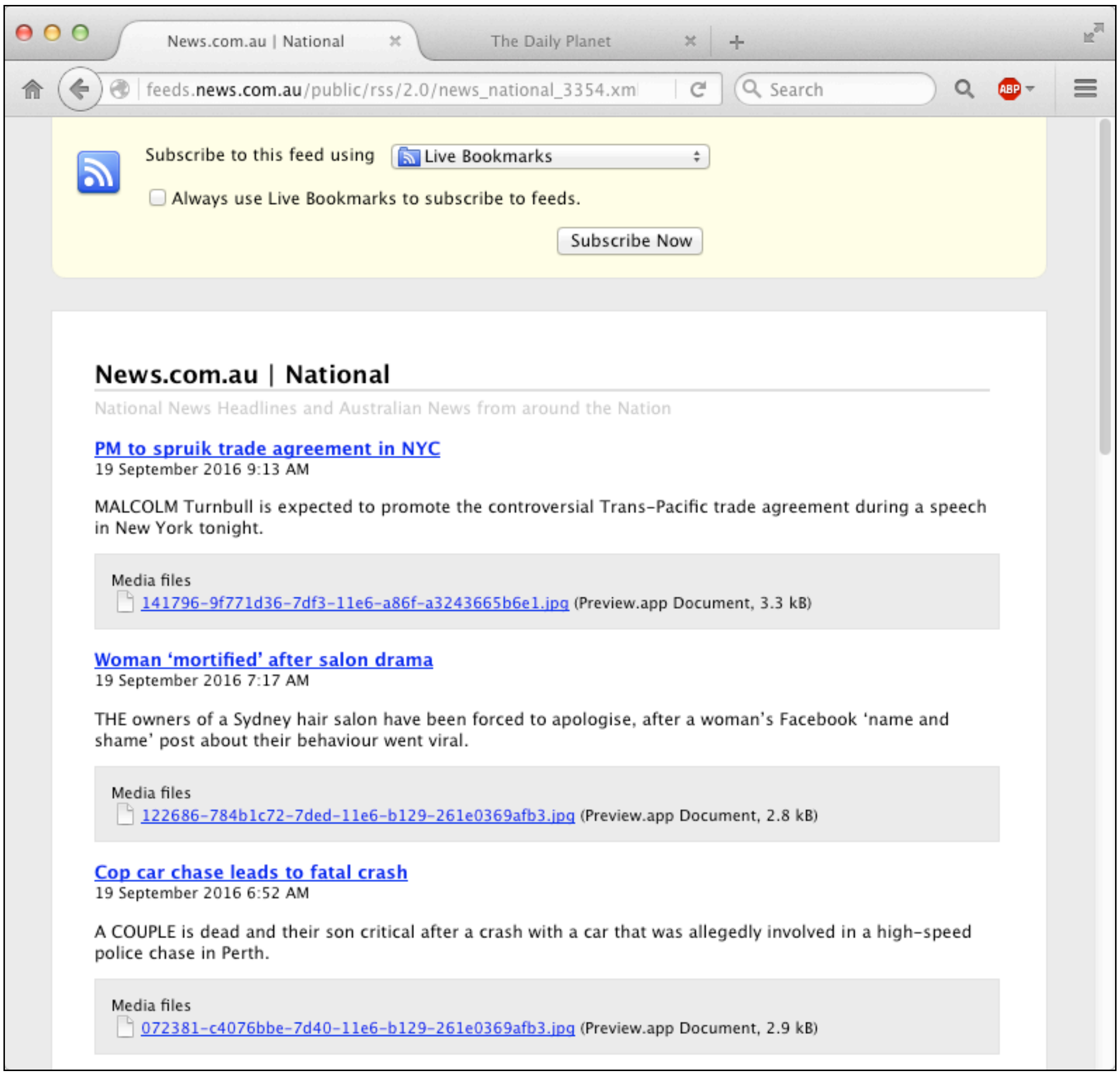
1. the category of story;
2. the URL where the original data was found;
3. the story's title (headline);
4. a photo illustrating the story;
5. a short summary of the story; and
6. the date and time the story appeared online.

Most importantly, items 3 to 6 are all extracted "live" from the online web document indicated. This was done by downloading the HTML source and using regular expressions to find the necessary elements needed to construct our own version of the story. The first part of the HTML code generated by our Python program is shown below (as displayed in the Firefox browser).

```

<!-- Custom-made newspaper -->
<html>

  <head>

    <title>
    The Daily Planet
    </title>

    <style>
    body {background-color: beige;}
    hr {border-style: solid; margin-top: 2em; margin-bottom: 2em;}
    </style>

  </head>

  <body>

    <!-- Masthead -->

    <!-- Newspaper title -->
    <p align = "center">
    <span style = "font-size:50px; font-family:Times">
    The Daily Planet
    </span>
    </p>

    <!-- Spinning world globe image -->
    <p align = "center">
    <img src = "http://manidaily.com/assets/world_news.gif"
         style="border:3px solid black">
    </p>

    <!-- Newspaper's slogan -->
    <p align = "center">
    <span style="font-size:22px; font-family:Times">
    Metropolis' <em>Super</em> News Source
    </span>
    </p>

    <!-- Person responsible -->
    <p align = "center">
    <span style="font-size:18px; font-family:Times">
    Editor-in-Chief: Perry White
    </span>
    </p>

    <hr width = 450px size = 5px>

    <!-- A news article -->
    <table align = "center">

        <tr> <!-- Newspaper section -->
```

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

To compose our HTML document, Rich Site Summary (RSS) web-feed files are downloaded from the `news.com.au` web site. RSS documents are XML files specifically intended to be machine-readable. They have a simple structure that makes it reasonably easy to extract their elements. An example of such a web document as it appears when examined in a web broswer is shown below.


News.com.au | National
The Daily Planet
feeds.news.com.au/public/rss/2.0/news_national_3354.xml
Search
Subscribe to this feed using
Live Bookmarks
Always use Live Bookmarks to subscribe to feeds.
Subscribe Now

News.com.au | National

National News Headlines and Australian News from around the Nation

PM to spruik trade agreement in NYC
19 September 2016 9:13 AM

MALCOLM Turnbull is expected to promote the controversial Trans-Pacific trade agreement during a speech in New York tonight.

Media files
141796-9f771d36-7df3-11e6-a86f-a3243665b6e1.jpg (Preview.app Document, 3.3 kB)

Woman 'mortified' after salon drama
19 September 2016 7:17 AM

THE owners of a Sydney hair salon have been forced to apologise, after a woman's Facebook 'name and shame' post about their behaviour went viral.

Media files
122686-784b1c72-7ded-11e6-b129-261e0369afb3.jpg (Preview.app Document, 2.8 kB)

Cop car chase leads to fatal crash
19 September 2016 6:52 AM

A COUPLE is dead and their son critical after a crash with a car that was allegedly involved in a high-speed police chase in Perth.

Media files
072381-c4076bbe-7d40-11e6-b129-261e0369afb3.jpg (Preview.app Document, 2.9 kB)

This was was the source of the data used to produce our National News story shown above. To compose the corresponding page for our newspaper we extracted the latest story's headline, story text, date and the address of the associated JPEG image. This data was then integrated into our HTML code.

To do this in Python 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, from a prior examination of the XML source of such documents we knew that each news item appears between `<item>` … `</item>` markup tags, and that the short summaries of each story appear between `<description>` … `</description>` tags, so we used this knowledge to help extract the necessary data. Our program does this whenever the "Print" button is pressed, so the generated HTML document will be updated with fresh data each time.

We also discovered that sometimes the downloaded text contained unusual characters that are not handled properly in Python strings, most notably "smart" quotes, so we replaced these with plain characters before "printing" our newspaper.

Requirements and marking guide

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

- **Generating a masthead (3%).** Your program must be able to generate an HTML file, `publication.html`, which begins with a 'masthead' identifying the nature of your periodical. When viewed in a web browser, the masthead part of the document must contain at least the following elements:
  - The name of the periodical.
  - An image evocative of the periodical's theme.

  The image must be sourced from online (you cannot attach image files to your solution). Since it will never change, the URL for this particular image can be "hard-wired" in your Python code. The HTML source generated by your Python program must be laid out neatly.
- **Generating four stories (8%).** Your Python program must be capable of generating at least four distinct "stories" as part of your periodical. Each such story must be derived from a different online web page, and must represent the latest story in a particular category at the time when the program runs. When viewed in a web browser, each story must contain at least the following elements:
  - the category of story,
  - the URL where the original data was found,
  - the story's title (headline),
  - an image illustrating the story,
  - a short summary of the story, and
  - the date and time the story appeared online.

  The last four of these items must all be extracted from the online 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 story elements is up to you and is determined by the design of your generated HTML code. The periodical must be easy to read. 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 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.

- **Providing an intuitive Graphical User Interface (4%).** Your program must provide an easy-to-use GUI. This interface must offer at least the following capabilities to its user:
    - It must allow the user to select up to four (or more) categories of story to print in the periodical. Any mechanism can be provided for selecting stories as long as it is intuitive and easy to use, e.g., push buttons, check buttons, menus, "spinboxes", etc.
    - It must allow the user to choose to "print" their periodical, i.e., to generate the `publication.html` file.
    - It must visually indicate to the user the progress being made on downloading data and generating their selected stories. Any clear mechanism can be used for doing so, e.g., a textual description, a progress bar, highlighting of GUI elements, etc.

  Optionally you may provide a mechanism for opening the generated HTML document in the local web browser as a local '`file://...`' document. However, as this is operating system and browser specific, care must be taken to achieve a portable solution. In particular, you will need to use the full path name of the file, and 'normalise' the file name to the host operating system's conventions. This can be done using Python's `getcwd` and `normpath` functions, as indicated in the supplied Python template file.

- **Code quality and presentation (2%).** Your Python program 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. In particular, each significant code segment must be *clearly commented* to say what it does, e.g., "Create the masthead", "Extract the first headline from the web page's source code", etc.

- ***Extra feature (8%).*** *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, in our example above we included a button in the GUI which opened the generated HTML document in the default web browser. We also supported more than four story categories.

You must complete the task using only basic Python features and the modules already imported into the provided template. In particular, you may not import any local image files. All displayed images and story text must be downloaded from online sources each time your program is run.

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 stories to display, the design of your Graphical User Interface, and the design of your periodical.

Support tools

To get started on this task you need to download various web documents of your choice and work out how to extract four things:

- The story's title.
- An image associated with the story.
- A short summary of the story.
- The date/time the story appeared online.

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 document elements in your solution. The solution to this problem is to use Python's 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 regular expressions, we have included two small Python programs with these instructions.

1. `downloader.py` is a small script that downloads and displays the source code of a web document. Use it to see a copy of your chosen web document in *exactly* the form that it will be received by your Python program. This is helpful because the version of a web document delivered by a web server to a Python program may *not* be the same as one delivered to a particular web browser! Worse, some web servers will entirely *block* access to web pages by Python programs in the belief that they are malware! In this case they usually deliver a short document containing an "access denied" message instead of the desired document.
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 perfect your regular expressions. There are also many online tools that do the same job, but the advantage of using our tool is that it is guaranteed to follow Python's regular expression syntax because it is written in Python itself.

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, owners of some other web pages 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. In short, when using `urllib` the process involves assigning a new value to the `urllib.URLopener.version` attribute and then using `urllib.urlopen` to request the web document as usual. We leave it to your own conscience whether or not you wish to do this, but note that the assignment can be completed 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 from the 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. 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/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 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. Design the HTML source code for your publication, with appropriate placeholders for the downloaded web elements you will insert. Keep the document simple and its source code neat.
5. Develop the necessary Python code to download the web elements and create the HTML file. This completes the "back end" functions of your solution.
6. 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 which pages of your periodical to "print", and extend your back-end code accordingly. Developing the GUI is the "messiest" step, and is best left to last.

Deliverable

You should develop your solution by completing and submitting the provided Python template file `publisher.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 do not need to use or import any other modules or files to complete this assignment.
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 and text you want to display must be downloaded from online web documents, not from local files.

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 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 will be created on Blackboard under Assessment for uploading your solution file before the deadline (11:59pm on Friday, October 21st). 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: Some RSS feeds that may prove helpful

For this assignment you need to find several web documents that contain regularly-updated stories and links to images, and that have a fairly simple source-code format. This appendix suggests some such pages, but you are encouraged to find your own. 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.

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.

For our example solution above we used News Limited's RSS feeds, but there are many other sites suitable for this assignment. Some examples of RSS sites (or pages that point to such sites) containing stories and embedded photo links include the following.

- CNN: http://edition.cnn.com/services/rss/
- NASA: http://www.nasa.gov/content/nasa-rss-feeds
- Apple: https://www.apple.com/rss/
- Netflix: http://dvd.netflix.com/RSSFeeds
- US ABC News: http://feeds.abcnews.com/abcnews/topstories
- New York Times: http://www.nytimes.com/services/xml/rss/index.html
- Yahoo7 (several feeds have images): https://au.news.yahoo.com/rss/
- Yahoo7 Entertainment News: http://d.yimg.com/am/entertainment.xml
- BBC News: http://www.bbc.com/news/10628494 or http://news.bbc.co.uk/2/hi/help/rss/default.stm
- Orlando Sentinel: http://www.orlandosentinel.com/news/breaking-news/rss2.0.xml
- Sky News: http://news.sky.com/info/rss

The following sites contain general lists of RSS feeds:

- http://www.feedage.com/
- http://www.uen.org/feeds/lists.shtml
- http://listofrssfeeds.com/

Not all such sites contain both stories and images in the same web document, however. The following sites contain story summaries but if you want to use them you may need to follow the story's links to the *next* web page to find the images.

- DTV: http://www.ndtv.com/rss
- The Sydney Morning Herald: http://www.smh.com.au/rssheadlines

- SBS news: http://www.sbs.com.au/news/rss-list
- The Courier Mail: http://www.couriermail.com.au/help/rss
- Australian government media RSS feeds (few with images): http://australia.gov.au/news-and- media/government-media-releases/media-release-rss-feeds
- Kitchen Window - recipes and food commentary from NPR: http://www.npr.org/rss/rss.php?id=4578972

Finally, you may find that some interesting RSS feeds only have images associated with some of their stories. Since we want the "latest" story in each chosen category, including an image, we'll allow you to use such sites as a special case, provided your program produces the latest *illustrated* story. You must be confident, however, that the feed will always have at least one illustrated story at any time. Furthermore, special care must be taken in this case to ensure that your program extracts *matching* text and images. Obviously it would not be acceptable to extract text from the first (non illustrated) story and the image from the fourth (illustrated) story in a news feed.