Assignment 2, Part A: Online Shopper

(19%, due 11:59pm Sunday, May 28th)

Overview

This is the first part of a two-part assignment. This part is worth 19% of your final grade for IFB104. Part B will be worth a further 6%. 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

The Internet has dramatically changed the way we conduct commerce. Online shopping is increasingly displacing traditional forms of retailing. Here you will develop an application that provides an online shopping experience, using data downloaded from the World-Wide Web. The program will have a Graphical User Interface that allows the user to choose how many products they want to buy from different categories. Having done so they will then be able to print an illustrated invoice summarising their purchases, which can be viewed in a standard web browser. Most importantly, the online shopping application will aggregate up-to-date data sourced from online "feeds" that are updated on a regular basis. To complete this assignment you will need to: (a) use Tkinter to create a simple Graphical User Interface; (b) download web pages using a Python script and use pattern matching to extract particular elements from them, and (c) generate an HTML document containing the extracted elements presented in an attractive, easy-to-read format.

Illustrative Example

For the purposes of this assignment you have a free choice of what products your online shopping application will offer. Categories of products could be:

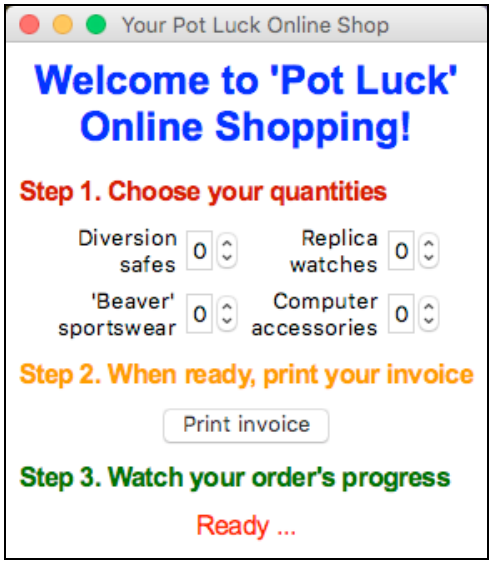
- clothing,
- electrical goods,
- books and magazines,
- motor vehicles,
- furniture,
- jewellery and cosmetics,
- etc.

However, whatever products you choose, you must be able to find at least three different on-line web sites that contain regularly-updated lists of such products. For each product there must be a textual description, a photograph, and a price. A good source for such data is Rich Site Summary (RSS) web-feed documents. Appendix A below lists some such sites, but you are encouraged to find your own of personal interest.

To demonstrate the idea, below is our own online shopping application, which uses data extracted from several different web sites. Our demonstration application allows users to select from four product categories, diversion safes (i.e., safes disguised as other household items),

replica watches, sportswear and computer accessories. The application allows users to choose how many items from each product category they want to buy. The program then downloads the necessary data from the various web sites and uses it to produce an illustrated invoice in the form of an HTML document which can be viewed in a standard web browser.

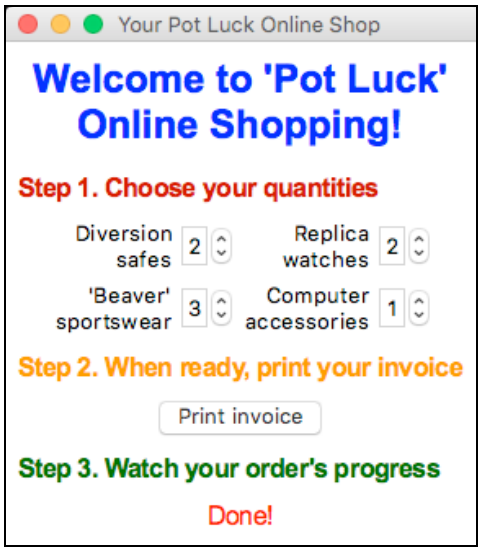
The screenshot below shows our example solution's GUI when it first starts. We've called our online shop *Pot Luck* because although the user can choose the quantities of goods they want to buy they have no control over which items they will get!



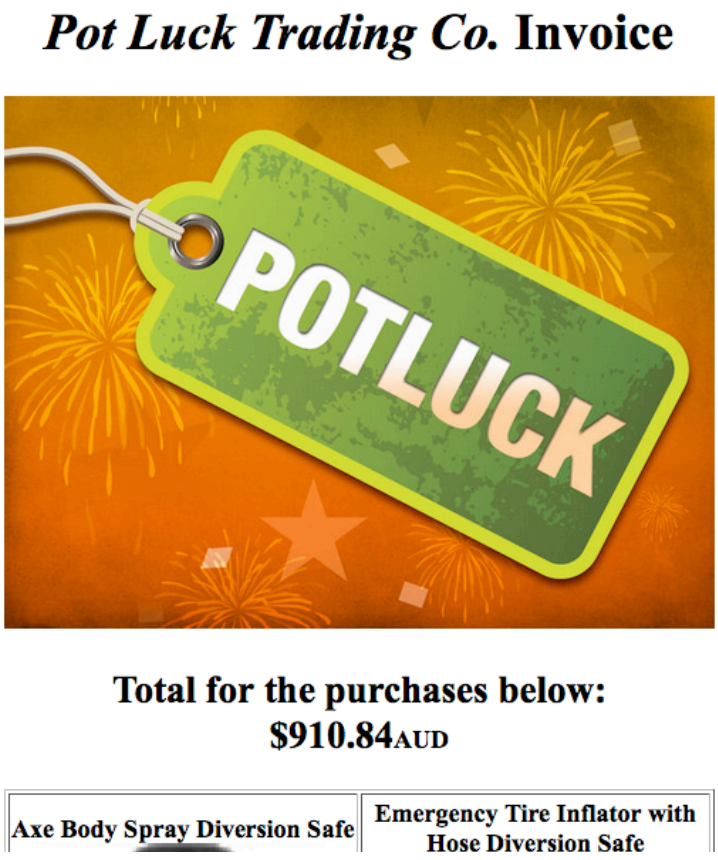
The user is invited to choose how many items in each category they want buy. In this case this is done using "spinbox" widgets, but other solutions are possible. Below the user has chosen to buy two safes, two watches, three items of sportswear and one computer accessory. Having pressed the button to start printing the invoice for these items, the user can watch their order's progress at the bottom of the GUI.

The application downloads information about the products currently for sale in each of these categories from four different online shopping sites and uses this information to generate an

HTML invoice. When the invoice is ready to be viewed the user is informed with a "Done!" message as follows.



The invoice appears in the file `invoice.html`, which the user can then open in any standard web browser. The invoice has several parts, and begins as follows.



Firstly there is a heading identifying our online shop. This is followed by an image, representative of our imaginary company. Like all data in the invoice, this image is itself downloaded

from the web, in this case via the following URL: http://www.connectgrowserveshare.com/dotAsset/9732.jpg.

Following this is the total cost for the user's eight purchases in this case. Importantly, this price is expressed in Australian dollars, even though the original web sites from which the product prices were obtained used United States dollars.

After this, each of the user's purchases is described. For each one there is a short textual description, a photo, and the price, again converted to Australian dollars. In this case the eight purchases were as follows.

Pot Luck Trading Co. Invoice

Total for the purchases below:
$910.84AUD

Axe Body Spray Diversion Safe  Our price: $30.52AUD	**Emergency Tire Inflator with Hose Diversion Safe**  Our price: $27.93AUD
Franck Muller Master Square Mens Large Unisex	**Swiss Army Infantry Vintage**

<table>
<tr>
<td>Wristwatch 6000 H SC DT R-10
 
Our price: $433.58AUD</td>
<td>Mens Wristwatch 241518
 
Our price: $340.48AUD</td>
</tr>
<tr>
<td>Bemidji State Beavers White cap
 
Our price: $13.30AUD</td>
<td>MacNaughton Cup Championship full color Tee
 
Our price: $26.53AUD</td>
</tr>
<tr>
<td>MacNaughton Cup Championship 1 color Tee
 
Our price: $22.54AUD</td>
<td>Colorado Big Mountains Mousepad
 
Our price: $15.96AUD</td>
</tr>
</table>

Like all the data displayed, the product photos have all been downloaded from different web sites. Those for the diversion safes and replica watches are only small "thumbnail" images, so appear blurry at this scale, but the sportswear and computer accessories sites provided our application with larger, more detailed images.

The final part of the invoice is due acknowledgement of the sources for all this product data. In this case the invoice provides hyperlinks to the four web sites used by our online shopping application, as shown below.

The *Pot Luck Trading Company* is proudly supported by:

- http://www.crimezappers.com/rss/catalog/category/cid/103/store_id/1/
- https://www.one-prices.com/feed/
- http://www.beaversportswear.com/rss.php
- https://feed.zazzle.com/rss?qs=computer-accessories

A special case, however, is what the invoice should contain if the user doesn't select any items, i.e., if the "Print invoice" button is pressed when all the product quantities are zero. In this case our demonstration application produces the following polite, "no charge" invoice.

***Pot Luck Trading Co.* Invoice**

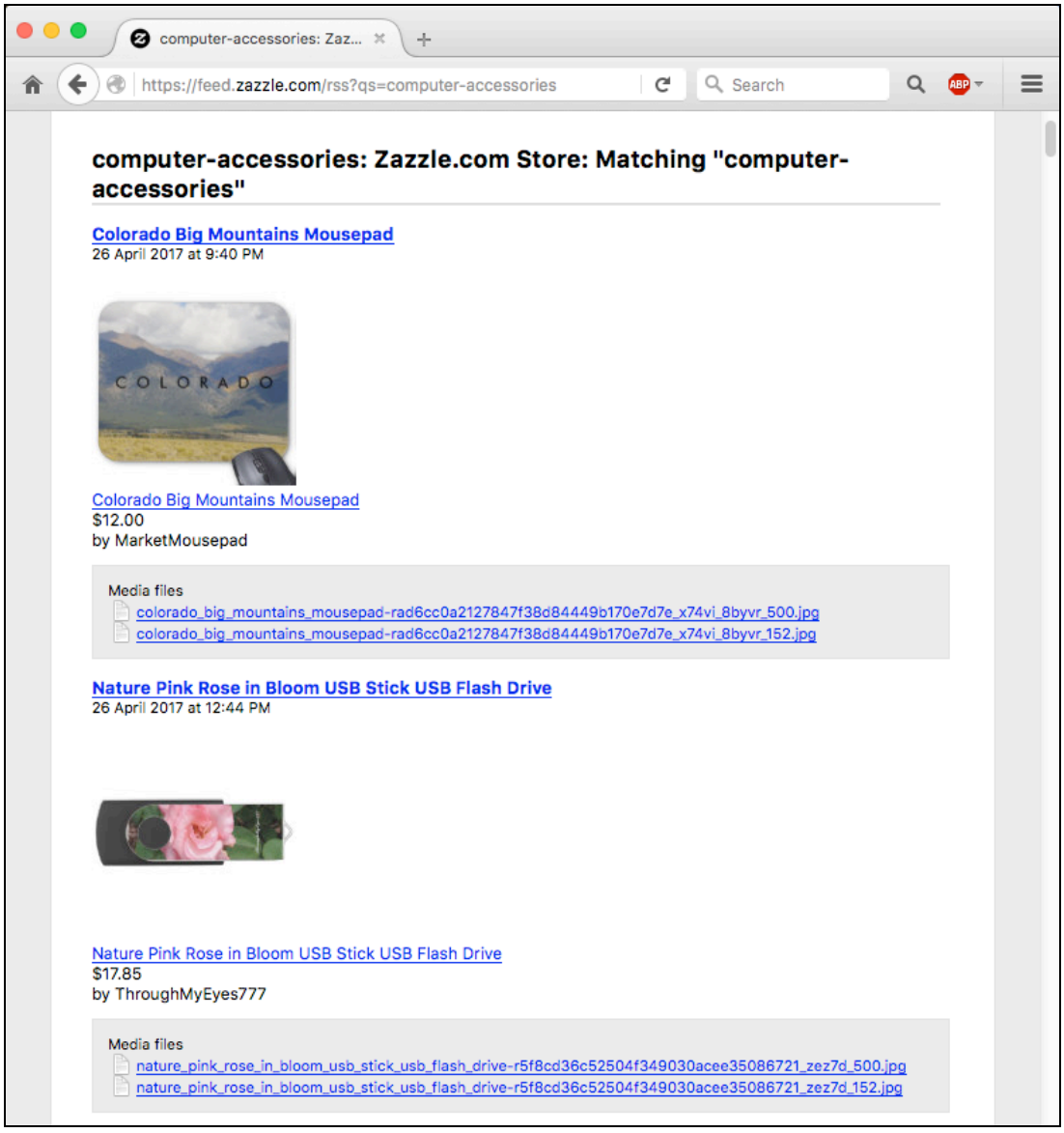
Thank you for browsing
Please call again

The *Pot Luck Trading Company* is proudly supported by:

- http://www.crimezappers.com/rss/catalog/category/cid/103/store_id/1/
- https://www.one-prices.com/feed/
- http://www.beaversportswear.com/rss.php
- https://feed.zazzle.com/rss?qs=computer-accessories

Online data sources

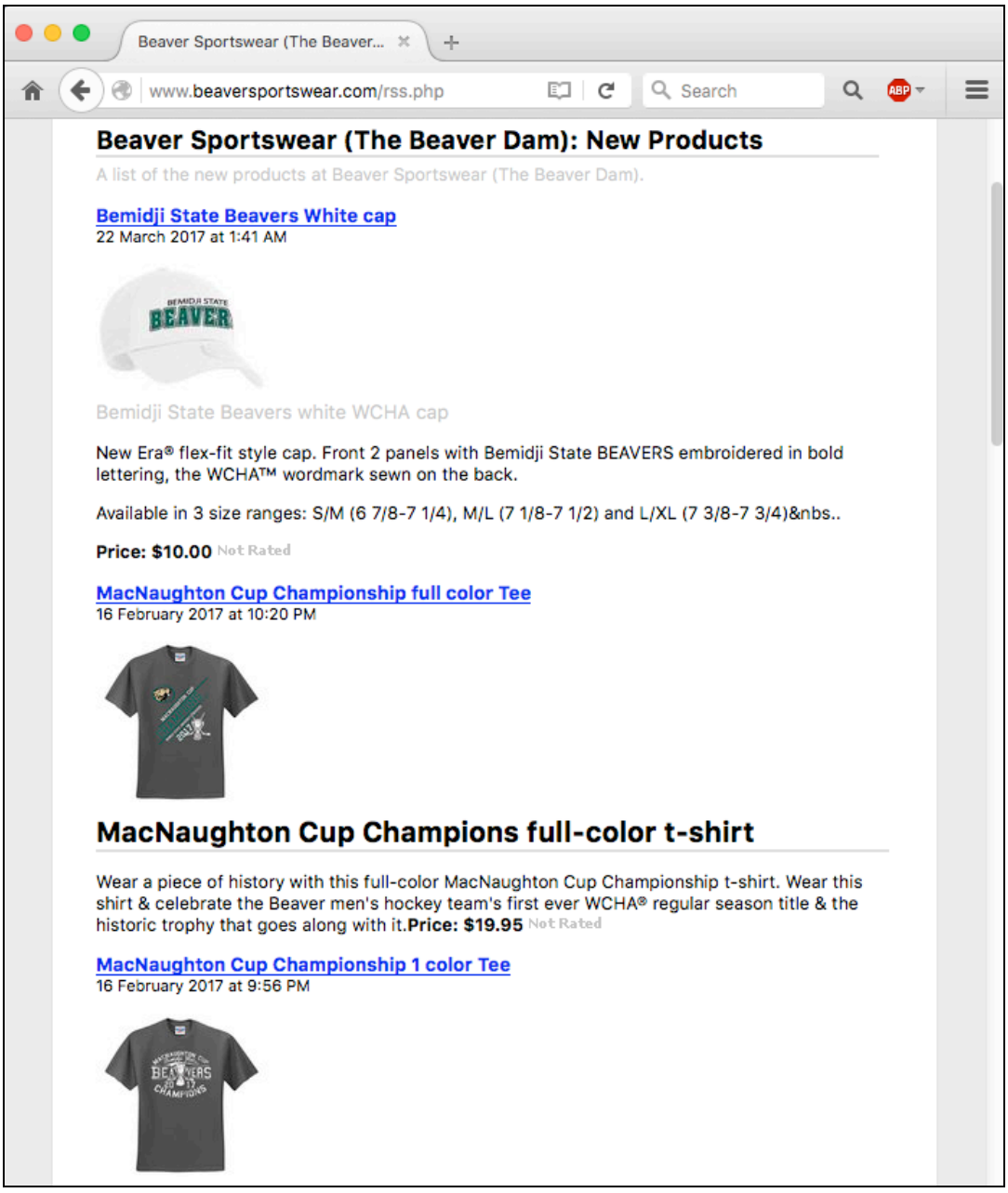
So where did the data about the products for sale actually come from? Most importantly, we downloaded all this data "live" from the four online web documents listed above. This was done by reading the HTML source files and using regular expressions to find the necessary elements needed to construct our invoice. For instance, we accessed the *Zazzle* online store for computer accessory products, using the following web document which, in full, lists dozens of products for sale.



This web document contains all the information we needed for our product invoice, descriptions, photos and prices (although in US dollars). Our application uses regular expressions to extract the necessary data, thereby allowing for the possibility that the products listed are changed. (Indeed this site *was* updated while we were developing our demo solution.)

Similarly, the sportswear items were extracted from the web site shown overleaf. As can be seen by the dates on the products, this particular web site is not updated often, but our appli-

cation must still allow for the possibility that new sportswear products will be added occasionally.



In each case our application extracts the topmost items in the product lists, regardless of what they are, as per the user's specified quantities. Our online shopping application must always produce the most recent items listed, even after the source web page has been updated.

Output format

Our Python application generates an HTML document in a file called `invoice.html` that can be viewed in a standard web browser. Although not intended for human consumption, the generated HTML code is nonetheless laid out neatly, and with comments indicating the purpose of each element. Part of the generated HTML for the invoice above is shown overleaf.

```
<!DOCTYPE html>
<html>

  <head>
    <!-- An automatically generated HTML document -->
    <title>Your 'Pot Luck' invoice</title>
    <!-- Fix the width of all elements and centre them -->
    <style>
        p {width: 500px; margin-left: auto; margin-right: auto}
        h1 {width: 500px; margin-left: auto; margin-right: auto}
        h2 {width: 400px; margin-left: auto; margin-right: auto}
        table {width: 500px; margin-left: auto; margin-right: auto}
        ul {width: 500px; margin-left: auto; margin-right: auto}
    </style>
  </head>

  <body>

    <!-- Titles and cost summary -->
    <h1 align="center"><em>Pot Luck Trading Co.</em> Invoice</h1>
    <p><img src="http://www.connectgrowserveshare.com/dotAsset/9732.jpg" alt="Pot Luck logo" width=500px></p>
```

To generate our HTML code we downloaded the source web documents as character strings and used a combination of Python string functions and regular expressions to isolate the elements we needed to construct our own HTML code. For instance, from a prior examination of the XML source code of the "replica watches" web site we knew that each watch's description appears between `<title>` … `</title>` markup tags so we used this knowledge to help extract the necessary data. Our application does this whenever the "Print invoice" button is pressed, so the generated HTML invoice 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, for instance "registered trademark" symbols, as shown in the sportswear product description below, so we removed these before adding the text to our invoice.

Requirements and marking guide

To complete this task you are required to develop an application in Python similar to that above, using the provided `online_shopper.py` template file as your starting point. Al-

though our demonstration allowed the user to select from four distinct product categories, you are only required to support three. Your solution must have *at least* the following features.

- **An intuitive Graphical User Interface (4%).** Your program must provide an easy-to-use GUI. This interface must have the following characteristics:
    - All of the widgets must be neatly laid out.
    - It must allow the user to select quantities from three (or more) categories of products on sale. The user must be able to select up to (at least) five items in each category. Any mechanism can be provided for selecting quantities as long as it is intuitive and easy to use, e.g., text entry boxes, radio buttons, spin-boxes, etc.
    - It must allow the user to choose to "print" their invoice, i.e., to generate the `invoice.html` file.
    - It must visually indicate to the user the progress being made on downloading data and generating their invoice. Any clear mechanism can be used for showing progress, e.g., a textual description, a progress bar, highlighting of GUI elements, etc.
- **The ability to generate the fixed invoice elements (2%).** Your program must be able to generate an HTML file, `invoice.html`, which contains the following fixed elements:
    - The name of your "online shop".
    - An image evocative of the shop's title. The image must be sourced from online (you cannot attach image files to your solution), but since it will never change, the URL for this particular image can be "hardwired" into your Python code.
    - Hyperlinks to each of the three (or more) web sites from which your application gets its product data. You must derive your product data from three distinct online shopping sites offering different products, not just three different product categories from the same online shop.
    - When the user has not bought any items an appropriate "no charge" message must be included.

  The HTML source code generated by your Python program must be laid out neatly.
- **The ability to calculate a total in Australian dollars (2%).** When the user has selected some quantities of items to buy, your application must download the prices for that many items, convert them to Australian dollars (if necessary), and display the total cost in the invoice. You do not need to use exact current exchange rates to perform this calculation but can use a reasonable approximation based on current rates. For instance, in our demonstration solution all of the prices downloaded were in US dollars, so we used a fixed multiplier of 1.33 to convert them to Australian dollars.
- **The ability to generate lists of products for each category (7%).** Your Python program must be capable of generating HTML code to display products downloaded from at least three distinct web sites in quantities as specified by the user. You must derive your product data from three different online shopping sites offering different

types of products, not just three different product categories from the same online shop.

For each quantity selected by the user you must add a description of that many products as derived from the online product lists. For each category of product there must be the current top-listed elements from the source web page, in the quantity specified by the user, and for each individual product there must be

- a textual description,
- a photo, and
- a price in Australian dollars.

In each case the description, photo and price must match correctly as shown on the source web page, e.g., you can't have the name of one product matched with the photo of another. (Mis-matched elements are an indication that your pattern matching solution is not working correctly.)

The top-listed products on the source web page must be used, regardless of any changes made to the web site since you developed your Python application. Your code for extracting online web elements cannot be hardwired to the source web sites as they were at a particular time in the past.

Each of the HTML elements used in your invoice must be extracted from the original document separately. It is *not* acceptable to simply copy large chunks of the original web document's source code. The HTML source code generated by your Python program must be laid out neatly.

When viewed in a web browser, your invoice must:

- layout all HTML elements neatly, irrespective of the size of the web browser's window.

The precise visual layout, colour and style of the invoice is up to you and is determined by the design of your generated HTML code. Nonetheless, the invoice must be attractive and easy to read. No HTML markup tags or other odd characters should appear in any of the text displayed in the invoice.

Data on the web changes frequently, so your solution must continue to work even after the source 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 certain day. Instead you will need to use pattern matching to actively find the text and photos in the documents, regardless of any product updates that may have occurred since you wrote your program.

- **Good Python and HTML code presentation (4%).** Both your Python program code and your generated HTML source code must be *presented in a professional manner* for both parts of the assignment. See the coding guidelines in the *IFB104 Code Presentation Guide* (on Blackboard under *Assessment*) for some 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., "Generate the invoice's title", "Show the total price", etc. in both the Python and HTML code.

- ***Extra feature (6%).*** *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 solution above we supported more than the required three product categories, and our application allows up to nine items to be ordered from each category, rather than the required five.

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 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 invoice.

Support tools

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

- The description of each listed product for sale.
- The photo of each listed product.
- The price of each product.

You also need to allow for the fact that the contents of the web documents from which you get your data can change unexpectedly, so you cannot hardwire the locations of the document elements in your solution. The solution to this problem is to use Python's `find` character string method and/or regular expression `findall` function to extract the necessary elements, no matter where they appear in the HTML/XML source code.

To help you develop your pattern matching solution, we have already provided two small Python programs.

1. `downloader.py` is a small script that downloads and displays the source code of a web document and was demonstrated in Week 7. 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 web browser! Worse, some web servers will entirely *block* access to web pages by Python scripts 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 data (see Appendix B).
2. `regex_tester.py` is an interactive program introduced in Week 8 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, 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 browsers such as Firefox, Safari, 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 (see Appendix B).

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 online 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 products you want to "sell" and search the web for appropriate HTML or XML documents that contain the necessary descriptions, photos and prices.
2. Using the `downloader.py` application from Week 7, 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 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.
3. Using the provided `regex_tester.py` application from Week 8, 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 elements, i.e., the text and the URLs of the photos, from the web documents 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 invoice, with appropriate placeholders for the downloaded web elements you will insert. Keep the document simple and its source code neat. The invoice must be well laid out when viewed in a web browser.
5. Develop the necessary Python code to download the web elements and generate the HTML file. This completes the "back end" functions of your solution.
6. Add the Graphical User Interface "front end" to your program. Decide which mechanism for allowing the user to choose product quantities you want to provide, e.g., text entry boxes, radio buttons, etc, 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 `online_shopper.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 is available on Blackboard under Assessment for uploading your solution file before the deadline (11:59pm on Sunday, May 28th). Note that you can submit as many drafts of your solution as you like. You are strongly encouraged to submit draft solutions before the deadline.

Appendix A: Some RSS feeds that may prove helpful

For this assignment you need to find several web documents that contain regularly-updated lists of products for sale, and that have a fairly simple source-code format so that you can easily extract elements from them. This appendix suggests some such pages, but you are encouraged to find your own. Note that you are *not* limited to using RSS feeds 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 products 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 by looking for the symbol above on shopping web sites. However, 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. To overcome this you can find various directories of RSS feeds online, as well as search engines specifically intended for finding RSS feeds. Explore!

For our example solution above we used four particular RSS feeds, but there are many other sites suitable for this assignment. Some examples of RSS feeds that could be used for this assignment include the following.

- A big site for Indian handicrafts: http://india-shopping.khazano.com/rss/
- A huge online "marketplace" with lots of "shops" that can be accessed as RSS feeds: https://www.etsy.com/au/c/. To find the RSS feed for a particular shop, e.g., "oktak", you just need to put the shop's name into a standardised URL as, in this case: https://www.etsy.com/shop/oktak/rss
- An online department store with lots for RSS feeds: https://feed.zazzle.com/rss. Use the "qs" qualifier to access different product categories, e.g., for products featuring cats use https://feed.zazzle.com/rss?qs=cats
- Another online shop with *lots* of feeds: https://www.rakuten.com/ct/rss/
- A site for buying property in the UK: https://www.foxtons.co.uk/buy/feeds.html
- Car accessories: https://www.seicane.com/rss
- Computer products and accessories (lots): http://www.tigerdirect.com/rss/index.asp
- Anti-theft technologies: http://www.crimezappers.com/rss/

- Dresses: http://www.joomlajingle.com/rss
- Shoes and handbags (and probably other stuff if you can figure out the necessary URLs): http://www.shoebuy.com/rss-sale-shoes and http://www.shoebuy.com/rss-sale-bags
- And many, many more you can find for yourself!

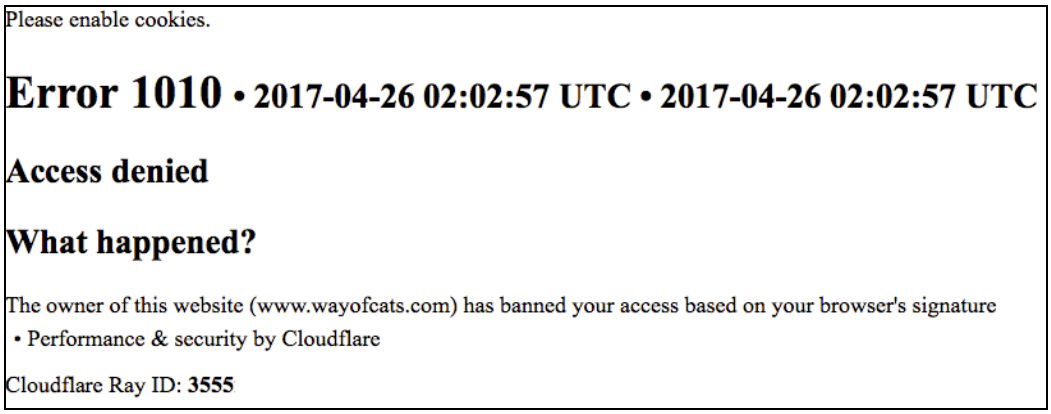
Not all such sites are easy to use, however. Some web sites are unreliable and, quite commonly, others don't put all the information needed for this assignment in one place. The following are some other sites we discovered while developing our demonstration solution for this assignment, but may be difficult to use.

- The following online department store site is *not* an RSS feed, but it has lots of pages of products and looks easily "scrapable": http://www.shopzilla.com/
- This site looks very promising at first because it lists lots of RSS feeds, but when we tried it they all took us to the same computer accessories page: https://www.newegg.com/RSS/Index.aspx
- This site has lots of great feeds, but appears to be *very* unreliable. Use it at your own risk! http://www.shop.com/rss-a.xhtml
- This menswear site has lots of feeds accessible from the categories on the first page below, but for each product you need to follow a link to find the prices:
  http://mensclothing2you.info/
  http://mensclothing2you.info/rss/clothing-men-dress.rss
  http://mensclothing2you.info/rss/mens-designer-suits.rss
  http://mensclothing2you.info/rss/mens-designer-pants.rss
  etc.
- Similarly, you need to follow the links to get the prices from this women's clothing site: http://womenclothing2you.info/rss.jsp
- Another online shop with lots of great stuff but in this case you need to follow more than one link to get full details: http://www.onlineshoppingaustralia.com.au/rss.xml

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 from Week 7 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 software "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**