Assignment 2, Part B: Save Your Order

(6%, due 11:59pm Sunday, May 28th)

Overview

This is the second part of a two-part assignment. This part is worth 6% of your final grade for IFB104. Part A which preceded it was worth 19%. This part 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. If you have a neat, clear solution to Part A you will find completing Part B much easier. For the whole assignment you will submit only one file, containing your combined solution to both Parts A and B, and you will receive one grade for the whole 25% assignment.

Motivation

One of the most common tasks in "Building IT Systems" is modifying some existing code. In practice, computer programs are written only once but are subsequently modified and extended often during their operational lifetime. Code changes may be required in response to internal factors, such as the need to correct errors, or external factors, such as changes in consumer requirements.

This task requires you to extend your solution to Part A of the assignment by adding an additional feature. It tests:

- Your ability to work under time pressure; and
- The quality and clarity of your code for Part A, because a well-written solution to Part A will make completing this part of the assignment much easier.

Goal

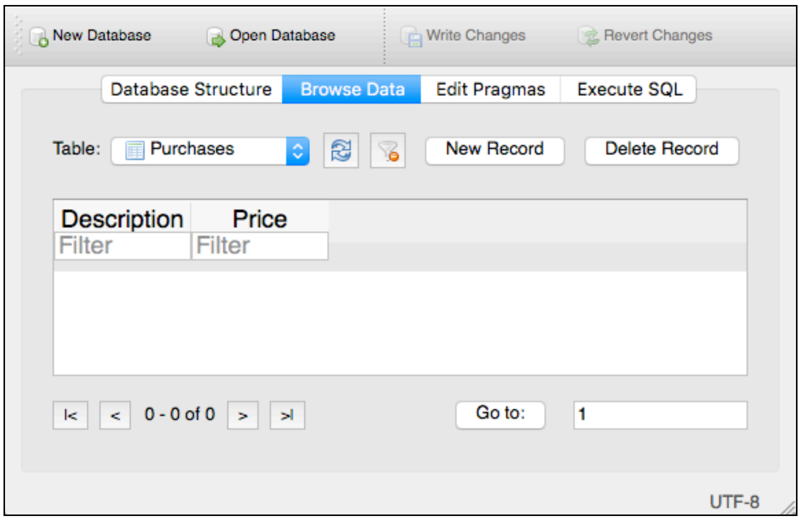
In Part A of this assignment you created a program which accesses data from multiple online shops and uses it to generate an invoice of purchases. However, another important aspect of buying goods is to keep records of your purchases. In this extension you will store a list of your purchases in a database, for later perusal using a tool such as the *DB Browser for SQLite* or for processing by some other financial or budgeting application. Your solution must have the following features.

- Some form of interactive widget must be added to your front-end GUI to allow the user to decide whether or not they want the details of their current invoice stored and to see that this has been done.
- When an invoice is generated and this function is activated, your back-end program must save details of all the most recent purchases in a supplied SQLite database, `shopping_trolley.db`. A copy of this database accompanies these instructions. It contains a single table, `Purchases`, which has two columns, `Description` and `Price`. Your program must populate these columns with the description and price (in Australian dollars) of all the items listed in the latest invoice.

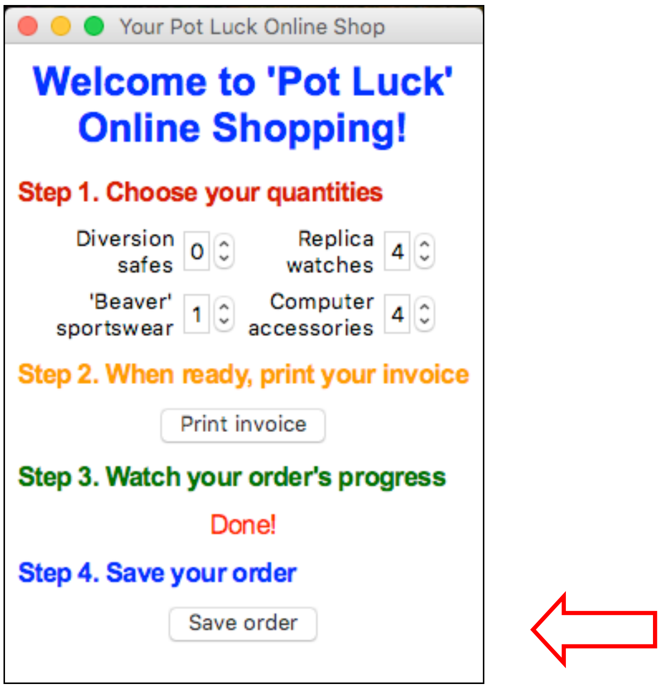
The data stored in the database should always be from the last time the user generated an invoice and any old data in the tables should be discarded.

Illustrative example

Supplied with these instructions is an empty SQLite database called `shopping_trolley.db`. If you open it in the *DB Browser for SQLite* or an equivalent tool you will see that it contains a single table with two columns, as shown below.



In the instructions for Part A we used a demonstration program which allowed the user to select quantities of products from four different categories. Here we extend its capabilities by adding an extra button for saving the user's current purchases in the database. Below is the updated GUI, showing the new widget.

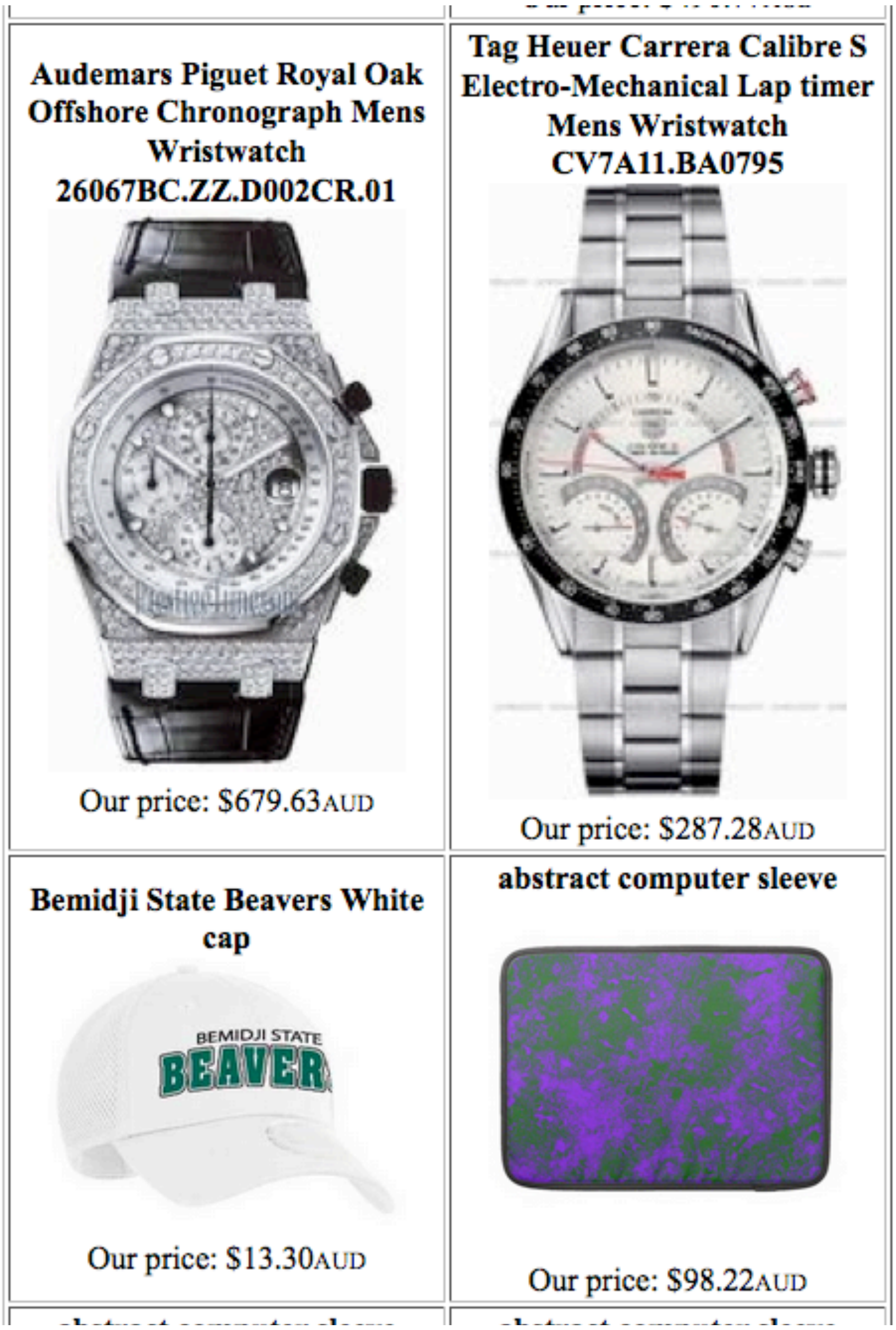

Your Pot Luck Online Shop
Welcome to 'Pot Luck' Online Shopping!
Step 1. Choose your quantities
Diversion safes 0
Replica watches 4
'Beaver' sportswear 1
Computer accessories 4
Step 2. When ready, print your invoice
Print invoice
Step 3. Watch your order's progress
Done!
Step 4. Save your order
Save order

In this example the user has chosen to buy four watches, one item of sportswear and four computer accessories, and has generated their invoice as can be seen by the progress display. The invoice thus created is illustrated by the sequence of screenshots shown below. They show the nine items purchased when our program was run on May 19th.

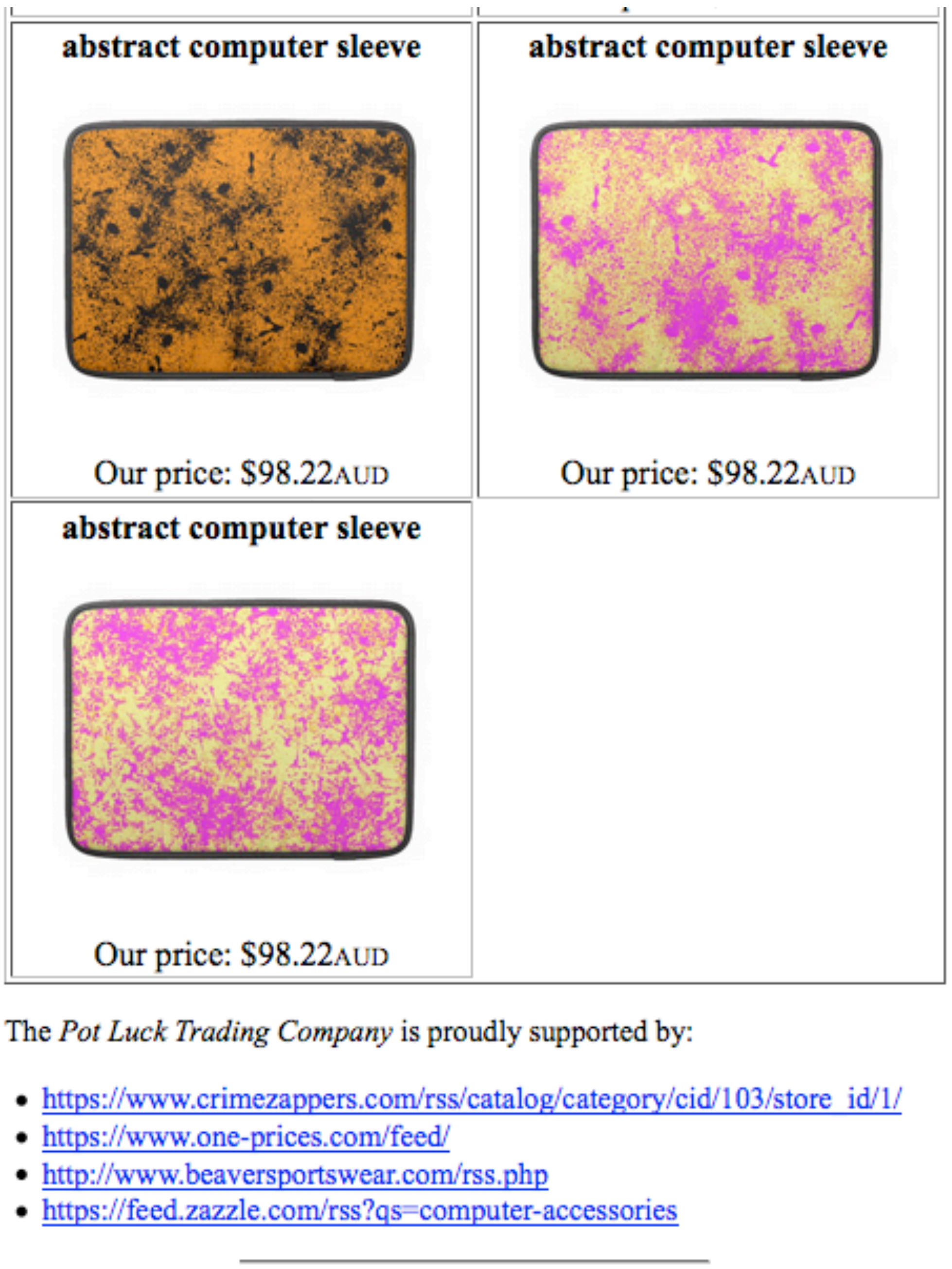
Pot Luck Trading Co. Invoice

Total for the purchases below:
$2217.64AUD

Swiss Army Chrono Classic Mens Wristwatch 241510 Our price: $353.78AUD	**Audemars Piguet Royal Oak Offshore Scuba Mens Wristwatch 15701ST.OO.D002CA.01** Our price: $490.77AUD



Audemars Piguet Royal Oak Offshore Chronograph Mens Wristwatch 26067BC.ZZ.D002CR.01 Our price: $679.63AUD	**Tag Heuer Carrera Calibre S Electro-Mechanical Lap timer Mens Wristwatch CV7A11.BA0795** Our price: $287.28AUD
Bemidji State Beavers White cap Our price: $13.30AUD	**abstract computer sleeve** Our price: $98.22AUD



abstract computer sleeve Our price: $98.22AUD	**abstract computer sleeve** Our price: $98.22AUD
abstract computer sleeve Our price: $98.22AUD	

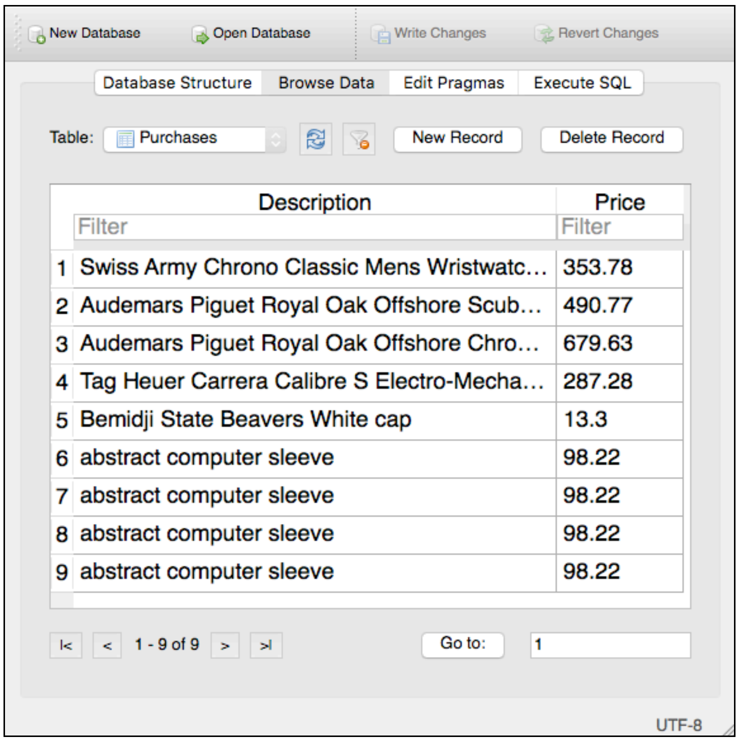
The *Pot Luck Trading Company* is proudly supported by:

- https://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

Importantly, notice that two of the web sites we used for our sample solution, selling replica watches and computer accessories, have **entirely updated their stock** since the Part A instructions for this assignment were written! The watches site appears to replace all of its product listings at the end of each week. Also, in this case the computer accessories listing from the *Zazzle* store was recently updated with a large number of identically-priced but differently-coloured computer sleeves. [Sic: The use of lower-case "abstract" rather than a capitalised "Abstract" is how these products were described on the source web site.]

Of course, we have no control over such updates, so it is an important requirement for the assignment that **your solution must continue to work even after the source web sites have been updated**. You **cannot hardwire your solution** to work only for the contents of the web sites at a particular time.

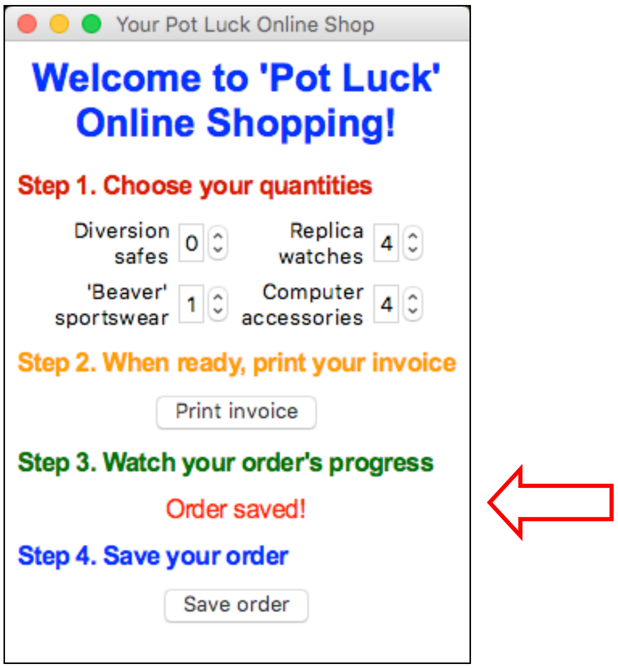
Having generated an invoice, the user of our sample solution can press the button to store their purchases in the database table as shown below.



	Description	Price
	Filter	Filter
1	Swiss Army Chrono Classic Mens Wristwatc…	353.78
2	Audemars Piguet Royal Oak Offshore Scub…	490.77
3	Audemars Piguet Royal Oak Offshore Chro…	679.63
4	Tag Heuer Carrera Calibre S Electro-Mecha…	287.28
5	Bemidji State Beavers White cap	13.3
6	abstract computer sleeve	98.22
7	abstract computer sleeve	98.22
8	abstract computer sleeve	98.22
9	abstract computer sleeve	98.22

Both the product descriptions and corresponding prices are stored. (Notice that due to the identical descriptions and prices for the computer sleeves on the source web site, we have allowed duplicate rows in the table by *not* defining a primary key for it. This is unusual for relational database tables, whose rows are normally meant to be unique, but necessary here.)

In addition, the user is given feedback that their purchases have been added to the database, as shown by the “progress” label below.


Your Pot Luck Online Shop
Welcome to 'Pot Luck' Online Shopping!
Step 1. Choose your quantities
Diversion safes 0
Replica watches 4
'Beaver' sportswear 1
Computer accessories 4
Step 2. When ready, print your invoice
Print invoice
Step 3. Watch your order's progress
Order saved!
Step 4. Save your order
Save order

Your task is to extend your solution to Part A of the assignment to provide a capability equivalent to that above. The shopper must be able to choose whether or not to store the contents of their latest invoice in the database and must receive some kind of visual feedback whenever this has been done.

Importantly, however, you do *not* need to copy our example above. This problem can be solved in other ways. For instance, rather than a push button you could provide the user with a checkbox which, when selected, means that the invoice will be stored in the database *automatically* whenever an invoice is generated. (Indeed, this may be a more convenient solution for the user.)

You also need to consider what to do if the user tries to save their purchases without having yet chosen how many to buy or without having yet generated an invoice. In this case our sample solution simply empties the contents of the database table to indicate that nothing has been purchased yet.

Requirements and marking guide

To complete this task you are required to further extend the provided `online_shopper.py` template file with your solution to Part B, on top of your solution to Part A, to provide a capability equivalent to that illustrated above.

Your submitted solution (for both parts of the assignment) will consist of a *single Python file.* The extension for Part B must satisfy the following criteria. Marks available are as shown.

- **Adding a "save order" widget and feedback to the GUI (2%).** Your program must provide some simple, intuitive feature to allow the user to choose whether or not to store their most recent purchases in the database. This could consist of a push button as shown in the example above or some other suitable widget such as a checkbox. Importantly your GUI should provide a sensible behaviour if this widget is activated when no purchases have yet been made. Also, whenever the "save order" function is completed the user must receive some kind of visual indication that the database has been updated.
- **Updating the database with the user's latest purchases (4%).** Whenever the "save order" feature is activated your program must store a list of the most recently purchased items in the database table `Purchases`. This must consist of each product's description and its price (in Australian dollars) as shown in the most recently generated invoice. Any data already in the table must be overwritten; only the most recent list of purchases should appear in the database. Your Python code can assume that the necessary SQLite database itself already exists in the same folder as your Python program.

You must complete this part of the assignment using only basic Python 2.7 features and the `sqlite3` module.

Supporting material

Accompanying these instructions we have provided a copy of the necessary SQLite database, `shopping_trolley.db`. It contains a single table, `Purchases`, which has two columns, `Description` and `Price`. As supplied the database table is empty. Your solution should assume that this database and its table *already exists in the same folder as your Python program*. Your GUI does not need to create the database or define the table.

Before you begin you should confirm that you can open this database with the *DB Browser for SQLite* or a similar database tool.

Development hints

- It should be possible to complete this task merely by *adding* code to your existing solution, with little or no change to the code you have already completed. You should be able to use exactly the same data that you use to generate the invoice to update the database.
- The SQLite statements needed to complete this task are quite simple. Similar examples can be found in the relevant lecture demonstrations and workshop exercises.
- If you are unable to complete the whole task, just submit whatever parts you can get working. You will receive *partial marks* for incomplete solutions.

Deliverable

You must develop your Part A and Part B solution by completing and submitting the provided Python 2.7 template file `inline_shopper.py`. You will submit one file only. **Do *not* submit any other files. Do *not* submit a copy of the SQLite database. Do *not* submit a compressed archive containing multiple files.**

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. *We will assume that submissions without a completed statement are not your own work and they will not be marked.*
2. Complete your solution by developing Python code in the provided template to meet the requirements of both Parts A and B.
3. Submit *only a single, self-contained Python 2.7 file*. All images and text you want to display for Part A must be extracted from online documents, not local files. We will use our own copy of the database to test your solution to Part B.

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. Only the last draft submitted will be assessed.