Due date: Sunday April 22, 2018 02:55

This assignment requires three short programs. For all three programs:

- use meaningful names for variables and functions.
- add any comments you think will help explain what your program is doing

Part 1: Sale Price Table

Write a program that calculates the sale prices for items normally priced from $9.95 to $49.95, where the sale price can be 5% to 50% off. Display the results in an appropriately formatted table. An example of the layout formatting is shown below

Normal price:	$9.95	$14.95	$19.95	$24.95	$29.95	$34.95	$39.95	$44.95	$49.95
%off: 5%	9.45	14.20	18.95	23.70	28.45	33.20	37.95	42.70	47.45
%off: 10%	8.96	13.46	17.95	22.45	26.95	31.46	35.96	40.46	44.96
%off: 15%	8.46	12.71	16.96	21.21	25.46	29.71	33.96	38.21	42.46
%off: 20%	7.96	11.96	15.96	19.96	23.96	27.96	31.96	35.96	39.96
%off: 25%	7.46	11.21	14.96	18.71	22.46	26.21	29.96	33.71	37.46
%off: 30%	6.96	10.46	13.96	17.46	20.96	24.46	27.96	31.46	34.96
%off: 35%	6.47	9.72	12.97	16.22	19.47	22.72	25.97	29.22	32.47
%off: 40%	5.97	8.97	11.97	14.97	17.97	20.97	23.97	26.97	29.97
%off: 45%	5.47	8.22	10.97	13.72	16.47	19.22	21.97	24.72	27.47
%off: 50%	4.97	7.47	9.97	12.47	14.97	17.48	19.98	22.48	24.98

You must use nested **for** loops, one inside the other. Solutions creating the results by adding lines like:

```
print(" 5% 9.45 14.20 18.95 23.70 28.45 33.20 37.95 42.70 47.45")
print("10% 8.95 13.46 17.95 22.45 26.95 31.46 35.96 40.46 44.95")
```

will be awarded zero marks.

Full marks will be awarded to those programs that produce output like that above with the decimal points all lined up, and the $ sign on the top line but not the others.

Marks will also be awarded for good program design. In particular, your code should not include “magic numbers”.

Part 2: A Movie Title Explorer

There is a file of the top 1000 or so movie titles called **movies.txt**. For this question you are to build a way of displaying a random movie title, finding movies, and building and saving a list of movies you would like to watch.

The program first displays a menu (something like that shown below) and carries out the appropriate action depending on which letter the user types, and then redisplays the menu:

```
*** Movie Title Explorer ***
l     - load file of movie titles r -
random movie
s  - search
sw - starts with
k - keep: save the last displayed movie title to your favourites f - favourites
display
c - clear favourites q - quit

command: ?
```

load Load a file of movie titles. Prompt for the name of a file. If no name is input (e.g. if using `input()` and the user presses the enter key), load from **movies.txt**, otherwise load it from the specified filename.

The file of movie titles is a text file is formatted with one movie per line.

random Randomly pick a movie title and display it.

search Prompt for a string, and then search for and display all movies that contain this string. Search case-independently, and separate the movies with a blank line. If the search string is empty, exit the search and redisplay the menu.

starts with Prompt for a string and then search for and display all movies that start with this string. Search case-independently, and separate movies with a blank line. If the search string is empty, exit the search and redisplay the menu.

keep Add the last displayed movie to a list of your favourites.

favourites Display the current favourites list.

clear Clear the favourites list.

quit Quit the program.

Don't forget to add appropriate diagnostics. E.g. you can't display a random or search before the movies file has been loaded.

IMPORTANT: for full marks, make sure that you use functions, one for each of the commands. You can of course, create any additional functions that will help to simplify or clarify your code.

Some suggestions on getting started:

1. Get the menu going by using a while loop. Inside the while loop display the menu and ask the user to input a command. Then use **if/elif** statements for each command. Initially, you can put a print statement inside each if statement—later you will call the function for that particular command.

2. Create a function for each command e.g. load, search,. . . . Initially, simply put a print statement inside each function, e.g. for the keep function, put **print("KEEP")**. You can test each command—it will simply print out a message.

3. Then get load going so you can load a text file into a list. We haven't yet covered file inpu/output, so you can just cut and paste the fol-lowing function to read the movies from a file and return a list of movies.

```
def read_file(filename):
    file = open(filename, "r") # open the file
    movies_list = file.readlines() # load each line into a list file.close() # close
    the file
    #  Now get rid of the "\n" at the end of each line using
    #  a "list comprehension". This is a construct we haven't
    #  yet covered, so for now just use the code. We'll explain
    #  later.
    movies_list = [movie.strip() for movie in movies_list] return
    movies_list
```

4. Implement the **random** command next. If you import the **random** module, you will find the **random.choice()** method useful here. It returns a randomly chosen element from a list. e.g.
   print(random.choice([1, 2, 3, 5, 9]))

5. Then get **quit** going, **search**, **startswith** search, and after that the **favourites** related commands.

Part 3

There is a link file called **stories.txt**. Copy and paste code into your program. These are the data you're going to use to do the following.

It consists of a list of lists, where each sub-list contains two elements, an ultrashort story and its author.

Create a program that lets you:

- Find stories that contain a certain pattern (e.g. sky)
- Find all stories by a certain author
- Find stories that are by a certain author and contain a certain word
- Find stories less than a certain number of words, using the **.split()** method to (roughly) split a line into words. You can ignore punctuation when doing this. If there are no stories, display a message stating this.
- Display all the stories

In the above, that matching is based on substring (i.e. pattern in line), so search for "wood" would match "Atwood". Exact author searches aren't necessary. The same goes for finding stories that contain a certain work.

When displaying the stories, format them with quotes around the story and the authors name on a following line, indented, like this:

"The baby's blood type? Human, mostly."
-- **Orson Scott Card**

Follow the model of using a menu outlined in the previous question.

Submission

Submit your Python programs, each named with **.py** extensions. You should have three **.py** files to submit, one for each part (eg **part1.py**, **part2.py**, and **part3.py**)