CSB110MC – Python Programming

Lauren Powell

The lab classes consist of tasks that are designed to be completed during the lab class. If you do not complete the tasks during the lab class then you should do them at home and have them ready to be checked in the next lab class.

Each task should be stored in its own `.py` file.

Do not use the interpreted mode for doing the tasks (only use the interpreted mode for experimentation). If you store your python code as `.py` files then you can go back and look at them at a later date.

CSB110MC Lab Class 3 — Monday 02/03/2023

☐ Task 3.1

Write a program which prints the numbers 10 to 1 (inclusive) using a **for** loop.

Hint: This can be done very easily using an appropriate step value.

Once you have done this, extend your program to print all the even numbers from 2 to 20 (inclusive) using a for loop.

☐ Task 3.2

Write a program which simulates tossing a coin 10 times. You can do this by having a for loop which:

1. Generates either a 0 or a 1 randomly.
2. If the number is 0, then print "Heads", if the number is 1 then print "Tails".

☐ Task 3.3

Write a program which asks the user to enter a series of integers (one after another). The program should store the input values in a **list**. The program should keep asking for integers until a sentinel value (of your choice) is entered.

Once the user has finished entering values, the program should output whole list of entered values sorted from smallest to largest, followed by the the number of even numbers and the number of odd numbers entered.

☐ Task 3.4

Write a program that reads a line of input as a string and outputs:

1. Only the uppercase letters in the string.
2. Every second character of the string.
3. The positions of all vowels in the string.

☐ Task 3.5

Write a program which reads numbers and adds them to a list if they are not already in the list. Once there are 5 numbers in the list, the program should print the contents of the list and terminate.

☐ Task 3.6

Write a program which asks the user to enter a series of sentences. The program should keep asking for sentences until user types `""` (the empty string, i.e., they just press enter). Once the user has typed `""` the program should output the total number of words that were entered and the total number of vowels.

Assume that any substring separated from others by spaces on both sides (or one side if it is at the start/end of the string) is a word.

☐ Task 3.7

Write a program that asks the user to enter some positive integer n.

The program should print the pattern:

```
    1
   212
  32123
 4321234
543212345
```

where the height of the pattern should be n. That is, the above pattern is printed when n is 5. The above pattern would be one row higher if n was 6.

☐ Task 3.8

The prices of tickets at a local cinema vary by seat and are shown by this table:

A	8.00	8.00	8.00	8.00	8.00	8.00
B	8.00	8.00	8.00	8.00	8.00	8.00
C	8.00	9.99	9.99	9.99	9.99	8.00
D	8.00	9.99	9.99	9.99	9.99	8.00
E	9.99	9.99	9.99	9.99	9.99	9.99
F	9.99	12.50	12.50	12.50	12.50	9.99
G	9.99	12.50	12.50	12.50	12.50	9.99
H	8.00	9.99	12.50	12.50	9.99	8.00
I	8.00	9.99	9.99	9.99	9.99	8.00
J	8.00	8.00	8.00	8.00	8.00	8.00

Write a program that stores the ticket prices (not the row letters) in an appropriate data structure.

The program should prompt the user to choose a seat in the format letter/number (e.g., B3 for the third seat from the left in the second row), then output the price of a ticket for that seat (8.00 for seat B3).

Once you have done this, extend your program so that it repeatedly asks for seat numbers until the user enters a sentinel value. The program should then print the **total** cost of all selected seats.

☐ Task 3.9

Write list comprehensions to generate:

1. `lst1`, the list of integers between -10 and 10 (inclusive).
2. `lst2`, the list of odd numbers in `lst1`.
3. `lst3`, the list of squares of negative numbers in `lst2`.
4. `lst4`, the list of numbers in `lst3` that contain a '9' as a digit.

□ Task 3.10

This task extends your Nim program.

Write a program that prompts the user for two integers, representing the number of tokens in two different piles. Print a representation of the two piles to the screen.

Ask the user if they want to go first or second. While there are tokens remaining in at least one of the piles, alternate between prompting the user to remove tokens from a pile and randomly removing tokens from a pile. After each turn, print the token piles to the screen.

When there are no tokens remaining, declare the winner of the game (player or computer), i.e., who took the last token.

An example run of the program might be:

```
How many tokens would you like in the piles?  6,3

pile 1: ******
pile 2: ***

Do you want to go first? (y/n)  y

Which pile do you want to remove tokens from?  1
How many tokens would you like to remove?  4

pile 1: **
pile 2: ***

I remove 2 tokens from pile 1!

pile 1:
pile 2: ***

Which pile do you want to remove tokens from?  2
How many tokens would you like to remove?  3

pile 1:
pile 2:

Player wins!
```

Hint: Neither player nor computer should be able to remove tokens from an empty pile.