COMPUTER SCIENCES: PROGRAMMING

Worksheet for Week 2: Functions and Advanced Lists / Data Persistence, Dictionaries and Tuples

JULY 13, 2020

TEESSIDE UNIVERSITY
Borough Road, Middlesbrough, TS1 4PA

Computer Sciences: Programming: Functions and Advanced Lists

Watch the week 2, part 1 video lecture (when available) before attempting these exercises. This session aims to familiarise you with functions, as well as some more advanced list methods.

Hint: In Python, if you create an empty function the program won't work. Every function needs at least one statement. Programmers often define their functions first, then fill them in, however. You can use the pass keyword to get around this. For example:

```
def empty_function():
    pass
```

Question 1: Shape Area Functions

You are tasked with drawing on work from week 1 to write functions that calculate the area of a sphere, cylinder and cone.

1. First, create three functions:
    a. `sphere(r)` that takes the radius of the sphere as `r` and returns its area.
    b. `cylinder(r, h)` that takes the radius of the cylinder as `r` and its height as `h`, returning its area.
    c. `cone(r, h)` that takes the radius of the cone as `r` and its height as `h`, returning its area.
2. Formulae for all 3 can be found here: https://www.bbc.com/bitesize/articles/ztqsdxs
3. Load these functions into the IDLE REPL using F5 and test each one out on the command line.

Question 2: Recursive User Input

Now, create a menu system using conditionals to ask the user to specify which shape they would like to calculate the area of. If they select an invalid option, use recursion to ask the question again. If you can't remember what recursion looks like, consult the lecture materials. Your menu should ask for input by number. For example:

```
Please type 1 for sphere, 2 for cylinder and 3 for cone: _
```

Question 3: Exam Results

You are tasked with writing some grade management software for a university. Download and edit the files `results.py` and `datafile.py` from Blackboard to proceed with this exercise. Don't be afraid to initially experiment with the files to see how they work!

1. Create a function that takes the results list as an argument and prints out a pass list, printing each student's name, score and grading (use distinction 70%, merit 60% and pass 40% grading schema).
2. Now code a function that takes the results list as an argument and prints out a list of all students that have failed with a mark below 40%, but where all students with a mark of 30-39 are entitled to a resubmission.
3. Now create a function that takes the results list and returns the following performance statistics, printing them out after the two results lists:
    a. Highest mark and student
    b. Marks range (highest and lowest)
    c. Average mark overall
4. Now create a function that returns how many students scored above the average and how many below, including failures and print these statistics.

Question 4: Slicing

You are writing an adventure game and you're required to implement some of its functionality. For this question, you'll need to read a tutorial on the Python random library such as this one: https://pythonspot.com/random-numbers/

1. Your adventurer has a backpack containing the following items:
    a. Water flask
    b. Cheese
    c. Gold coins
    d. Handkerchief
    e. Tinderbox
    f. Scrolls
    g. Dagger
    h. Rope
    i. Nuts
    j. Pipe,
    k. Tobacco
    l. Wine skin
    m. Herbs
    n. Axe
2. Create a new Python file and populate a list called `rucksack` with these items as strings.
3. Now, sort your rucksack using the `sort()` function and print out a list of the items in it.
4. Now print how many items you have in your rucksack.
5. Your adventurer finds a treasure chest which contains gems and a necklace. Add the chest items to your rucksack and sort it again, printing its contents again.
6. Your adventurer goes to sleep and a thief steals 5 random items from your rucksack. Remove 5 random items and list your rucksack's contents again.

Question 5: Dice

Create a program to throw 2 dice 100 times. Record how often each number from 2 to 12 is thrown in a suitable list and then print out a graph of your results like so:

```
Distribution Chart
Score     Rolls
2    1    *
3    5    *****
4    11   ***********
5    13   *************
6    15   ***************
7    16   ****************
8    18   ******************
9    8    ********
10   6    ******
11   4    ****
12   3    ***
```

Question 6: Coin Tossing

You’re tasked with writing a program to toss a coin.

1. Create a program to toss a coin 100 times, recording whether it lands heads or tails as “H” or “T” in a data list. Print your list in 5 rows of 20.
2. Now modify your program to find the longest continuous run of heads and the longest continuous run of tails.
3. Now modify your program to count how many times three heads were tossed in a row and output your result.

Question 7: List Management

You are tasked with writing a list management program.

1. Create a program to generate 20 random numbers from 0 to 99 and store them in a list.
2. Locate second largest and smallest numbers Now modify your program to determine the second largest and the second smallest number in the list and output your results.
3. Now modify your program to create a list of 20 random numbers, output the list, and then remove all entries greater than the third highest number in your list and output the resulting list after this operation.
4. Re-organise list data Create a program to generate a new list of 20 random numbers from 0 to 99.
5. Insert number 50 in the middle position of the list. All numbers in the first half of the list that are greater than 50 should be removed from the first half and appended to the second half.
6. Then do the same with all numbers in the second half of the list that are smaller than 50, appending them to the end of the first half of the list before the number 50.

Computer Sciences: Programming: Data Persistence, Dictionaries and Tuples

Watch the week 2, part 2 video lecture (when available) before attempting these exercises. This session aims to familiarise you with writing to, reading from, appending to and deleting from files.

Hint: In Python, you can search within strings in the same way you can search within a list, using the `in` keyword.

```
>> "sand" in "sandwich"
True
>> "abc" in "abacus"
False
```

Question 1: Sandwich Menu

You are tasked with writing an interactive sandwich menu for a restaurant.

1. First, in Notepad, create a file that contains the names of different types of sandwiches, one per line (e.g. "Tuna Mayo"). Call this file `menu.txt`.
2. Write a program that asks the user for the name of a sandwich, then politely tells them whether or not there is a sandwich with that name in `menu.txt`.

Question 2: Smart Sandwich Menu

You are tasked with making the interactive menu from the previous question smarter!

1. Extend your answer to the previous question to not only search for a complete string match but search *within* each item to see if the ingredient is present.
2. For example, a search for "Tuna" should return "Tuna Mayo", "Tuna Sweetcorn" and any other sandwich with tuna in its name.
3. Make the search ignore uppercase and lowercase letters (read about Python's case functions `lower()` and `upper()` here: https://www.geeksforgeeks.org/isupper-islower-lower-upper-python-applications/)
4. See this week's hint (above) for a pointer on how to achieve this.

Question 3: Loading Cars

You are tasked with writing an application to load information about cars from a file and search that information.

1. Download `cars_exercise_3.txt` from Blackboard and open it in Notepad. Notice its structure:
   a. 3 lines describe one car—line 1 is make, line 2 is model and line 3 is year.
   b. There is information about 3 cars present in the file.

2. Now, write a script to load these 3 cars as dictionaries from the file. Each dictionary should have 3 keys: `make`, `model` and `year`
3. Now, print out the make, model and year information for the oldest and newest car.

Question 4: Splitting the Car Database

You are tasked with writing a program which splits the car database from question 3 into smaller files.

1. Write a script that will read in the car database used in question 3 and write out 3 files named as `make_model_year.txt` containing information about one car only. For example, `mitsubishi_outlander_2015.txt` should contain:

   ```
   Mitsubishi
   Outlander
   2015
   ```
2. Now add another car to the original file and run your program again. The program should still work as expected, producing 4 files this time instead of 3. If it does not do this, modify it so that it does.

Question 5: Removing the Years

You are tasked with writing a program which will remove all years from the car database used in questions 3 and 4.

1. Write a script that will read in the car database used in questions 3 and 4 and write it out again with all years removed.

Question 6: Moving the Years

You are tasked with writing a program which will combine the year associated with a car with its model number:

1. Adapt your answer to question 5 to prepend the year to the model number instead of deleting it from the file completely. The resulting file should look like this:

   ```
   Hyundai
   2006 Tucson
   Mitsubishi
   2015 Outlander
   Ford
   1998 Focus
   ```

Extension Activity: A More Sophisticated App

You are tasked with writing a car management app for a car showroom.

Note: This activity is designed to be challenging! If it seems difficult, that's because it is! Don't give up and approach it with curiosity, rather than dread. It's worth it I promise!

1. Produce an app which first reads all cars from `cars_exercise_3.txt` used in previous exercises into a list of dictionaries as in question 3, then asks the user if they'd like to add or remove a car.
2. If the user specifies that they'd like to add a car, ask them for the make, model and year of the car and add it to the list of dictionaries, then write the list of dictionaries back to the text file, formatted in the same way as the original file (3 lines per car: make, then model, then year). **Be careful here not to lose data!**
3. If the user specifies that they'd like to remove a car, ask them to specify a make, model and year. Check the list of dictionaries for a car that fulfils these criteria, ignoring letter case (i.e. in a case-insensitive way—see question 2).
4. If one is found, ask the user if they're sure they want to delete it. If they choose yes, remove the car in question from the list of dictionaries and write out the file again. If they choose no, skip to point 6.
5. If one is not found, tell the user that the app could not find a car that satisfies the criteria.
6. Loop back around to point 1 and ask the user if they'd like to add or delete a car again. This should continue until the user types 'exit'.