Modules 2 and 3 - Working with Data Types and Operators in Python and Writing Functions in Python

Author: Jessica Cervi

Expected time = 1.5 hours

Total points = 135 points

Assignment Overview

In this assignment, you will work with the Python data types you learned in Module 2, such as integers, floats, booleans, lists, tuples, and dictionaries. You will also practice using comparison and logical operators. In the second half of the assignment, you will apply what you have learned in Module 3 to use built-in and user-defined functions. You will also work with loops and conditional statements. Before you begin the assignment, review the instructions below and complete Question 0 to practice using Vocareum.

This assignment is designed to build your familiarity and comfort coding in Python while also helping you review key topics from each module. As you progress through the assignment, answers will get increasingly complex. It is important that you adopt a data scientist's mindset when completing this assignment. **Remember to run your code from each cell before submitting your assignment.** Running your code beforehand will notify you of errors and give you a chance to fix them before submitting. You should view your Vocareum submission as if you are delivering a final project to your manager or client.

Vocareum Tips

- Do not add arguments or options to functions unless you are specifically asked to. This will cause an error in Vocareum.
- Do not use a library unless you are expicitly asked to in the question.
- You can download the Grading Report after submitting the assignment. This will include feedback and hints on incorrect questions.

Learning Objectives

- Use Jupyter Notebooks to begin programming in Python.
- Identify Python data types, including strings, integers, floats, booleans, lists, and tuples.
- Explain the operations that can be performed on a given data type.
- Examine the benefits and limitations of using lists, tuples, and dictionaries to store and organize your data.
- Explain the benefits of functions in performing specified tasks in Python.
- Use built-in functions to perform common tasks in Python.
- Use methods to perform analyses on specific data types.
- Create custom functions to perform custom analyses relevant to the dataset or research question.

Index:

Module 2: Working with data types and operators in Python

Module 3: Writing functions in Python

Instructions:

All the questions in the assigments will be graded automatically and you will have the chance to fix your solutions many times in order to get the expected output. Make sure to follow the steps carefully (and don't delete or change anything unless you're asked to).

Question 0 is designed and solved for you to understand how to complete this and the next assignments.

Below you can see the structure of the questions. In the markdown cell just before the question, you will see the question number and the description of what has been asked.

Next, there is a code cell starting with the comment: `### GRADED`. This means that the cell will be autograded.

The line `###BEGIN SOLUTION` tells you where to write your solution. You can use as many auxiliary variables and additional codes as you need. Note that we have already initialized the answer variables that you will need for you final answer. These variables and assigned them to `None`. This is to avoid errors when typing the variable names that contain the results. You can replace the value `None` with the actual answer or redefine it below.

Finally, you will see a second code cell with the autograded solution. **DO NOT CHANGE ANYTHING IN THAT CELL**

Question 0:

Assign the String `"hello"` to the variable `var`:

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
var = "hello" ## edit code here, i.e., set var = "hello"
```

Module 2: Data types

In the first part of this assignment, we will be testing your knowledge of the topics covered in Module 2, such as basic data types (`int`, `floats`, and `str`) as well as lists, tuples, and dictonaries.

Floats

As you have learned, float are data types designed to store decimal numbers.

Back to top

Question 1

5 points

Assign the value of 175 multiplied by 16.28 to the variable `ans1` and make sure it's stored as a float.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans1 = 175*16.28
```

Back to top

Question 2

5 points

Assign the value of 572 divided by 8.73 to the variable `ans2` and make sure it's stored as a float.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans2 = 572/8.73
```

Strings

Strings are data types designed to contain single characters or a series of them, such as words or sentences.

Back to top

Question 3

5 points

Assign the sentence "I am a student" as a string to the variable `ans3`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans3 = 'I am a student'
```

Converting Data Types

As you have learned, int are data types designed to store integer numbers.

Back to top

Question 4

5 points

Assign the value of 25.927 to the variable `ans4` and convert it to an integer.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans4 = int(25.927)
```

Lists

In Python, a list is a collection of elements which is ordered and **changeable**. Lists can contain integers, floats, strings, other lists, and so on.

Back to top

Question 5

5 points

Create a list with the numbers `1,4,9,16,25` and assign it to a variable called `list_1`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
list_1 = [1,4,9,16,25]
```

Back to top

Question 6

5 points

Create a list with the elements: `1, "Hello", 14, 5.6` and assign it to a variable called `list_2`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
list_2 = [1, "Hello", 14, 5.6]
```

It is also possible to define nested lists (i.e., a list inside another list). Additionally, lists can have repeated elements.

Back to top

Question 7

5 points

Given `list_3 = [3,4,5]`, create a nested list that includes the elements `"Columbia", 11, 24.5, 3, list_3` and assign it to a variable called `list_4`.

In [ ]:
```
### GRADED
list_3 = [3,4,5]

### YOUR SOLUTION HERE
list_4 = ["Columbia", 11, 24.5, 3, list_3]
```

You can access the list items by referring to the index number. Remember, Python starts counting from 0, **not 1**. Additionally, negative indexing means beginning from the end: -1 refers to the last item, -2 refers to the second last item, etc.

Back to top

Question 8

5 points

Given `list_5 = [3,4,5,6,7,8,9]`, access the last element of the list and assign it to the variable `list_5_elem`.

In [ ]:
```
### GRADED
list_5 = [3,4,5,6,7,8,9]

### YOUR SOLUTION HERE
list_5_elem = list_5[-1]
```

Back to top

Question 9

5 points

Define `list_6 = [10,20,30,"Hello", 5.82,1,"student", "Python"]` and delete the third last element on the list.

In [ ]:
```
# GRADED


### YOUR SOLUTION HERE
list_6 = [10,20,30,"Hello", 5.82,1,"student", "Python"]
del list_6[-3]
```

Tuples

A tuple is a collection which is ordered and **unchangeable**. In the same way as lists, tuples allow duplicate members.

Back to top

Question 10

10 points

Create a tuple with elements from 10 to 20 and assign it to the variable 'tuple_1'. Get the first 3 values and assign them to a variable called `sliced_tuple_1`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
tuple_1 = tuple(range(10,21))
sliced_tuple_1 = tuple_1[:3]
```

Dictionaries

A very important data type in Python is dictonary. A dictionary is a collection which is unordered, changeable, and indexed.

A dictionary holds a `key:value` pair. Each key-value pair in a dictionary is separated by a `:`, whereas each key is separated by a `,`. Keys in a dictonary must be unique, but values can repeat themselves.

Back to top

Question 11

10 points

Create a dictionary with a key "name" with value "Andrew" and a key "age" with value 25. Assign it to a variable called `dict_1`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
dict_1 = {"name":"Andrew", "age":25}
```

Back to top

Question 12

5 points

Create a new key `"city"` in `dict_1` and assign the value `"Seattle"` to it.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
dict_1["city"] = "Seattle"
```

Back to top

Question 13

5 points

Change the value of `"city"` to `"New York"` in `dict_1`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
dict_1["city"] = "New York"
```

Module 3: Writing functions in Python

In the second part of this assignment, we will be testing your knowledge of the topics covered in Module 3, such as Python built-in functions, user-defined functions loops, and conditional statements.

Methods Scripts

A function is a bundle of code which only runs when it is called. Python comes with many built-in functions. You can find a list of the functions that come with Python here.

Back to top

Question 14

5 points

Assign -25.82 to the variable ans14 and take the absolute value.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans14 = -25.82
ans14 = abs(ans14)
```

Back to top

Question 15

5 points

Assign the string "I live in the US" to the variable `ans15`. Next using a string method, change all the characters to lower-case letters.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
ans15 = 'I live in the US'
ans15.lower()
```

Conditional if statements

Sometimes you want your function to do one thing under one condition and another thing under another condition. We can specify this in the function body using the keywords if, else, and elif.

Back to top

Question 16

10 points Given `num_1`, `num_2` and the list `values`, write a program that appends a string `"True"` to `values` if `num_1` is less than 0 and appends "False" to values if `num_2` is greater than 0.

In [ ]:
```
### GRADED

num_1 = -738.9
num_2 = 11
values =[]

### YOUR SOLUTION HERE
if num_1 < 0:
    values.append("True")
if num_2 > 0:
    values.append("False")
```

Foor loops

For loops work on iterable objects. An iterable object is an object that returns its members one at a time.

Back to top

Another basic operator in Python is the remainder function. The corresponding symbol is `%`. This function returns the remainder when first operand is divided by the second

Question 17

10 points

Create a program that iterates using a **for loop** over all values of `list_mod`. The program must check each value and append the string `"True"` to the list `result` if the value is even or append the string "False" to the list result otherwise.

In [ ]:
```
### GRADED

list_mod = list(range(1,15))
result = []

### YOUR SOLUTION HERE

for number in list_mod:
    if number % 2 == 0:
        result.append("True")
    else:
        result.append("False")
```

Back to top

Question 18

10 points

Given the list called `time` with elements from 7 to 24 use a **for loop** to create a program that appends `"Good morning"` to `result_time` if an element in the list `time` is less than or equal to 12, appends `"Good afternoon"` if an element is *less than 20*, and otherwise appends `"Good night"` to `result_time`.

In [ ]:
```
### GRADED

time = list(range(7,25))
result_time = []

### YOUR SOLUTION HERE

for hour in time:
    if hour <= 12:
        result_time.append("Good morning")
    elif hour < 20:
        result_time.append("Good afternoon")
    else:
        result_time.append("Good night")
```

User-defined Functions

As you write more complicated code, built-in Python functions won't be enough. For this reason, Python allows you to create user-defined function. Function are defined using the `def` keyword followed by the name of the function and the arguments you need to pass.

Arguments are specified after the function name, inside parentheses and a colon is needed after the arguments. You can add as many arguments as you want, just separate them with a comma. Finally, after you wrote the block of instructions you want your function to perform, return a value using the `return` keyword. Note that the bundle of instructions and the return statement need to be **indented**.

Back to top

Question 19

10 points

Create a function called `validate` that checks whether the variable `x` is even. If it's even, return the string `"even"`; if it's odd, return the string `"odd"`.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
def validate(x):
    if (x % 2 == 0):
        return "even"
    else:
        return "odd"
```

Back to top

Question 20

10 points Create a function called `abs_value` that takes a number and returns the absolute value of it.

In [ ]:
```
### GRADED

### YOUR SOLUTION HERE
def abs_value(x):
    if x >= 0:
        return x
    else:
        return -x
```