due: 02.20.2022 Sunday 11:59 pm

• do not hardcode the test cases into your solutions (this is counterproductive anyway, since we will be testing your code using different test cases)

Grading

Grading: a problem is correct if all our tests pass, otherwise it will be considered as False.

Each Questions Worth 20 points

*** Do not forget to include *"independent completion form"* ***

Instructions

Each step indicates the relevant section of the primer_3.0, where you can find help on this issue.

To work on HW2:

• build a HW2 directory under your home directory (cs103sp22)

• start Jupyter Notebook and from dashboard navigate to hw2 folder

• create hw2.ipynb in hw2 folder

• for each question, you need to first define the proper function and call the function with the given test cases

• in the notebook, edit the first markdown cell, add your full name and blazerid and Run it

HW1, CS103 Fall 2021

name:

blazerid:

• in the notebook, create the myName and myBlazerID functions which returns your name and your blzerid (the same function you had in hw1)

• once you are confident that your code works, submit on Canvas

Practice problem

p(x) Write the function `p(x)` that takes a float value **x**, and returns **0** if `x < 0`, **1** if `x ∈ [0,1]`, and **0** if `x > 1`. In signal processing, this is an important function called the unit pulse.

Correct answer:

```
def p(x):
     if x<0 or x>1:
          return 0
     else:
          return 1
# Call the function
print(p(5))
```

Mandatory Functions

The following functions should be implemented, and the return statements should be modified with the correct credentials. Do not forget to call the functions

```
def myName():
    return "James Bond"
def myBlazerID():
    return "jbon12"

# Call these functions
print("My Name is =", myName(), " and my BlazerId is =",myBlazerID())
```

HW2 problems

`isOdd (n1)` Write the function `isOdd()` that takes a single parameter **`n1`** as input and returns the Boolean value **`True`** if n is an odd integer and **`False`** otherwise. *Note that the parameter may be any type, but the return value is False whenever it is not an int*.

For example, ***isOdd(5) is True and isOdd(5.1) is False.

The **parity of an integer** is an important property and a good introduction to modulo arithmetic. Modulo arithmetic evolves into finite fields, which are used in cryptography. Finite fields are a discrete form of periodic functions, like the cosine function, which are important throughout science, such as in signal processing and complex analysis.

`paintTheRoom(length, width, height)`

Write the function **`paintTheRoom(length, width, height)`** that takes three floats for **`length, width,`** and **`height`** of the rectangular room, all in feet. Using this information and the nominal coverage for a given paint, compute the amount of paint needed to cover the four walls and ceiling, assuming no doors or windows. The function returns the whole number of gallons of paint needed. Assume one gallon of paint will cover 400 square feet. You need to *round up* the required gallon number.

`Sample Code Run`

Assume a room of `length 10 feet, width 12 feet and height 8 feet.`
The area of the walls would be:
10 ft x 8 ft = 80 sq. ft. (2 walls this size)
12 ft x 8 ft = 96 sq. ft. (2 walls this size)
The area of the ceiling would be:
10 ft x 12 ft = 120 sq. ft.
The total area would then be:
2 x 80 sq ft. + 2 x 96 sq. ft. + 120 sq. ft = 472 sq. ft.
472 sq. ft / 400 sq. ft per gallon = **1.18 gallons thus 2**

sumToMagic(listOfInt,magicSum)

Write the function `sumToMagic(listOfInt,magicSum)` that takes a list of integers(`listOfInt`) and an integer value(`magicSum`). The function will return the indices of two numbers such that their sum is equal to the `magicSum`. Assume that the list only contains the integer values, and the size of the list is not fixed. Additionally, assume that each input will have exactly one solution (you don't use the same element twice). The order of the answers is not important.

Sample Input:	Expected Output
`listOfInt=[8,16,22,35]` `magicSum=30`	`[0, 2]`
`listOfInt=[7,4,22]` `magicSum=11`	`[0, 1]`
`listOfInt=[1000,99,11,1,24]` `magicSum=100`	`[1, 3]`

tupleCounter (t) Write the function "**tupleCounter**" that takes a tuple **t** and returns an integer. The function will check each element of the tuple and returns the total number of odd elements. Assume the tuple contains only integers.

Sample Input:	Expected Output:
***t = (2,4,6,7,9,121,1)	4
t = (1,3)	2
t = (10,30,44)	0

`smallerThanIndex(listOfNumbers)`

Write the function `smallerThanIndex(listOfNumbers)` that takes a list of integers(`listOfNumbers`) and return an **integer**. The function will check every number's value and their indices. Count the number of integers in the list whose value is smaller than index and return the total.

Sample Input:	**Expected Output**
`listOfNumbers=[10,20,1,2,30]`	`2` **Hint (1 and 2)*
`listOfNumbers=[1,2,0,44,29,309]`	`1` **Hint (only 0)*
`listOfNumbers=[-4,-3,2,1,0]`	`4` **Hint (-4,-3,1,0)*

Submission:

Make sure that all of your code is correct and producing the correct output. Then, upload your hw2.ipynb file into Canvas. Do not forget to sign and upload your independent completion form