due: 09.28.2025 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_2.0, where you can find help on this issue.

To work on HW2:

• build a HW2 directory under your home directory (cs103fa25)

• 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

name:

blazerid:

• in the notebook, create the myName and myBlazerID functions which returns your name and your blazerid (the same function you had in hw1, you can copy-paste them)

• 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 "jbon007"

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

HW2 problems

`isOddOrEven(n)`

Write a function `isOddOrEven(n)` that returns "**Even**" if n is an even integer, "**Odd**" if n is an odd integer, and **False** for all other types.

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.

Sample Function Calls

```
>>> isOddOrEven(6)
"Even"
>>> isOddOrEven(7)
"Odd"
>>> isOddOrEven(6.5)
False
>>> isOddOrEven("10")
False
```

`floorPaintingCost(length, width, costPerSquareFoot, discount, extraCoat=False)`

Write a function floorPaintingCost(length, width, costPerSquareFoot, discount, extraCoat=False) that calculates the total cost to paint a rectangular floor.

Details:

1. length and width are the dimensions of the floor in **feet** (floats).
2. costPerSquareFoot is the cost of painting **one square foot** of the floor (float).
3. discount is a **percentage discount** applied to the total cost (float between 0 and 100).
4. extraCoat is a **boolean** (default False). If True, assume the floor needs **two coats of paint**, so multiply the total area cost by 2 **before applying discount**.
5. Return the **final cost after discount** as a float rounded to 2 decimal places.

Sample Function Calls

```
>>> floorPaintingCost(10, 12, 2.5, 10, extraCoat=False)
270.0   # 10*12*2.5 = 300, 10% discount -> 270
>>> floorPaintingCost(5, 5, 3, 0, extraCoat=True)
150.0   # 5*5*3*2 = 150, no discount
>>> floorPaintingCost(20, 15, 4, 25, extraCoat=True)
900.0   # 20*15*4*2 = 2400, 25% discount -> 1800
```

`findPairsWithSum(listOfInt, targetSum)`

Write a function `findPairsWithSum(listOfInt, targetSum)` that returns **all pairs of indices** where the sum of the two numbers equals `targetSum`.

Sample Function Calls

```
>>> findPairsWithSum([1, 2, 3, 4], 5)
[[0, 3], [1, 2]]
>>> findPairsWithSum([10, 20, 10, 30], 40)
[[0, 3], [2, 3]]
```

`sumOfSquaresUpToN(n)`

Write a function `sumOfSquaresUpToN(n)` that returns the sum of squares of all integers from 1 up to `n`.

- If `n` is not a positive integer, return `False`.

Sample Function Calls

```
>>> sumOfSquaresUpToN(5)
55   # 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55
>>> sumOfSquaresUpToN(0)
False
>>> sumOfSquaresUpToN(3.5)
False
```

`countGreaterThanPrevious(listOfNumbers)`

Write a function countGreaterThanPrevious(listOfNumbers) that counts how many numbers are **greater than the previous number** in the list.

- The first number is ignored since it has no previous number.

Sample Function Calls

```
>>> countGreaterThanPrevious([1, 3, 2, 4, 5])
3   # 3>1, 4>2, 5>4
>>> countGreaterThanPrevious([5, 4, 3, 2, 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