CS128 – Homework #8. Functions Spring 2022

Task1 (printing function, no return statement)

Create a function named **pizza_shares** that takes 2 parameters:

- slices : the number of pizza slices
- people: the number of people who will share the pizza.

The function should print how many slices each person receives. If there are leftover slices, the function should print that as well.

Use descriptive variable names for the parameters, along with any local variables you create inside the function.

Add function calls after the function definition to call the pizza_shares function several times to test your code:

Function Call	Expected Output
pizza_shares(15,5)	15 slices split among 5 people = 3 slices per person.
pizza_shares(22,4)	22 slices split among 4 people = 5 slices per person. 2 slices leftover.

What happens if you call the function with 0 as the second parameter? Your program will bomb due to a division by zero error. Add a call **pizza_shares(8,0)** to confirm the error:

```
---------------------------------------------------------------
ZeroDivisionError                         Traceback (most re
cent call last)
<ipython-input-18-e0e407c7589d> in <module>()
     17 pizza_shares(33,12)
     18
---> 19 pizza_shares(8,0)
```

Fix the function to prevent this error by testing whether the number of people is positive before computing the slices per person. Print an error message if a value less than 1 is passed as the second parameter. The code should only compute the slices per person and leftover slices if the number of people is positive. Run the code to confirm the error message is displayed:

pizza_shares(8,0)	Number of people must exceed 0

TASK2a (printing function, no return statement)

A pedometer treats walking 2,000 steps as walking 1 mile. Write a function named **pedometer** that takes 2 parameters:

- steps: the number of steps walked
- goal: the miles for the daily goal.

The function should compute the number of miles walked based on the number of steps passed to the first parameter. The function should also compute the remaining steps required to meet the daily goal that was passed to the second parameter.

Use descriptive variable names for the parameters, along with any local variables you create inside the function.

Print a congratulations message if the daily goal is met or exceeded, otherwise print the additional steps required to meet the goal.

Test your code with the following function calls:

Function Call	Expected Output
pedometer(1500, 1)	1500 steps = 0.75 miles Walk 500 more steps to meet 1 mile goal
pedometer(2000,1)	2000 steps = 1.0 miles Congratulations, you met the 1 mile goal
pedometer(2000,1)	2500 steps = 1.25 miles Congratulations, you met the 1 mile goal
pedometer(4250, 3)	4250 steps = 2.125 miles Walk 1750 more steps to meet 3 mile goal

TASK2b (function returns a value)

Copy the code from task2a to task2b. Remove the print statements in the function. Instead of printing, the function should return a value. If the number of steps remaining to reach the goal is positive, return that value. If the goal was met or exceeded, return 0 to indicate no additional steps are required.

Update the function calls to use assertions to test the return value. Add a print statement after the assertions that only displays if all assertions pass.

```
#Main algorithm.  Add your function calls here

assert(pedometer(1500, 1) == 500)  #goal not met, need 500 more steps
assert(pedometer(2000, 1) == 0)  #goal met, no more steps needed
assert(pedometer(2500, 1) == 0)  #goal exceeded, no more steps needed
assert(pedometer(4250, 3) == 1750)  #goal not met, need 1750 more steps
assert(pedometer(6000, 3) == 0)  #goal met, no more steps needed

print('Task2b tests passed')
```

TASK3a (printing function, no return statement)

Create a function named **driving_costs** that takes 3 parameters:

- distance traveled
- mpg (miles per gallon)
- cost per gallon

The function should compute and print the number of gallons required to travel the distance based on the mpg, along with the cost for that quantity of gas given the price per gallon. The function should perform error checking of the mpg variable to prevent division by zero.

Call the **driving_costs** function several times to test your code:

Function Call	Expected Output
driving_cost(100,20,2.50)	5.00 gallons needed for 100 miles based on 20 mpg 5.00 gallons at $2.50 per gallon = $12.50
driving_cost(510,33,2.75)	15.45 gallons needed for 510 miles based on 33 mpg 15.45 gallons at $2.75 per gallon = $42.50
driving_cost(100,0,1.75)	mpg must be positive

TASK3b (function returns a value)

Copy the code from task3a to task2b. Remove the print statements in the function. Instead of printing, the function should return a value. If the mpg is positive, return the calculated cost. Return an error message if the mpg is not positive.

Update the function calls to use assertions to test the return value.

```
#Main algorithm.  Add your function calls here

assert(driving_cost(100,20,2.50) == 12.50)

assert(driving_cost(510,33,2.75) == 42.50)

assert(driving_cost(100,0,1.75) == "mpg must be positive")

print('Task3b tests passed')
```

SUBMIT TO CANVAS

Submit hw8.ipynb. Save your notebook. Select File/Download As/Notebook.

Submit hw8.pdf. Select File/Print Preview. Right click/Print/Save as pdf.