Problem Set II: Functions

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo
Due: 1/18/2019

Purpose

To understand the purpose of functions and practice defining and calling them.

Description

Functions are one of the most import building block of a program, which:

- Provide an easy means of reusing segments of code
- Allow the programmer to conceptually encapsulate parts of long programs
- Simplify testing and debugging by allowing the programmer to focus on one area

In Python, functions are **defined** using the `def` keyword. A function is only ever defined once for a program.

A function can be **called** by referring to its name along with a list of arguments. It can be called any number of times; however, all function calls must occur after its definition.

```
def add_one(x):
      return x + 1

add_one(0)       # evaluates to 1
add_one(1)       # evaluates to 2
add_one(99)      # evaluates to 100
```

Testing

You are required to write at least 3 tests for each of the specified functions. Tests are written using assert statements; for a function `f`, an example would be

```
assert f(2) == 8
```

where the expression after the `assert` keyword must be a Boolean expression.

Since we are emphasizing test-driven development, you should read the descriptions for each of the functions below and write tests for them first. In doing so, you will have a better

understanding as to what the functions take as input and produce as output, which makes writing the function definition easier.

In your pset2.py file, insert the following line and put all your function and class definitions above it (each definition should start from the beginning of a line) and put your test cases including assert statements and function calls below the line with each statement indented by 4 spaces:
If __name__ == '__main__':

By the say, the line above instructs python to execute code inside the 'if' statement only when the program file is run as the main file.

Implementation

~~Create a separate file called `tests.py` with at least 3 tests for each of the functions below (unless otherwise stated).~~

I/O Functions

Write a function `print_hello` that takes a string `name` and uses `print` to display the message `"Hello " + name`. This function should ~~**not**~~ have a `return` statement but will still require **exactly one** test case.

Next, write a function `get_numbers` that takes no arguments and prompts the user twice for a number using the `input` function and returns the sum of these numbers. This function must **not** have a test case. (Consider why a test case would be difficult to use.) Instead, you should test this function using the Python interpreter.

Finding the Cube

Write a function `cube` that returns the cube of its input. For example, if its input is 2 then the value returned should be 8.

Calculating a Triangle's Hypotenuse

Write a function `get_hypotenuse` that takes two arguments corresponding to the lengths of the sides adjacent to the right angle of a triangle and returns the hypotenuse of the triangle. To perform the square root operation, you may take a value to its ½ power.

Converting Math Notation to Code

Write a function `do_math` that performs the same calculation as:

$$\frac{3x^2 + 4y}{2x}$$

Detecting Positive Numbers

Write a function `is_positive` that takes a single number as an argument and returns `True` when the argument is positive (zero exclusive) and `False` otherwise. You must write this function using a relational operator (without any sort of conditional statement).

Detecting Two Positive Numbers

Write a function `both_positive` that takes two numbers as arguments and returns `True` when both are positive and `False` otherwise. **This function must make two calls to the `is_positive` function above.**

Computing the distance between 2 points in 2D coordinate.

Define a class Point, which defines a point in 2D cartesian coordinate system (Please refer the lecture slide). Write a function `get_distance` that takes two Point objects and calculate the euclidean distance between the two and returns it as a floating point number rounded up to 2 digits after the decimal point. If you are not familiar with Euclidean Distance please look it up on Wikipedia or your Math text. To round a floating point number, you can use Python builtin function **round()** which takes two arguments: the first one is a floating point number to be rounded, the second one is a number of digit after the decimal point. To compute square root, you can use **math.sqrt()** function. Add the following line at the top of your file:
import math

Submission

Zip `pset2.py` ~~`and tests.py`~~ and submit to Polylearn.