Lab 3: Functions

Purpose

To be able to write and call functions.

Description

Functions are one of the most important building blocks 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
```

Implementation

Include a module docstring and function docstrings (one docstring per function under each function header) in your program file lab3.py.

I/O Functions

Write a function `print_hello` that takes no arguments, prompts the user for their name using the input function, and displays the message `"Hello " + name` on the screen.
This function shall not `return` any values explicitly. You do not need to write a test for this function. (Consider why a test case would be difficult to use.) Instead, you should test this function manually by calling the function and typing inputs on your keyboard.
Start by writing the signature (the header with the description of the input and output) of the function: `def print_hello():`
Then, write the docstring as follows:

```
def print_hello():
    """Asks the user for their name and prints hello + name.
    """
```

Next, write a function `get_numbers` that takes no arguments, prompts the user twice for a number using the `input` function, and displays the sum of these numbers on the screen.
This function shall not `return` any values explicitly. You do not need to write a test for this function. (Consider why a test case would be difficult to use.) Instead, you should test this function manually by calling the function and typing inputs on your keyboard.
Start by writing the signature (the header with the description of the input and output) of the function: `def get_numbers():`
Then, write the docstring as follows:

```
def get_numbers():
    """Asks the user to input two numbers and prints the sum.
    """
```

Finding the Cube

Write a function `cube` that returns the cube of its int argument (parameter) `num`. For example, if its argument is 2 then the value returned should be 8.
Start by writing the signature (the header with the description of the input and output) of the function in the following format: *def function_name(argument:type)->return_value_type:*
For example,

```
def cube(num:int)->int:
```

Then, write some examples in the docstring before writing the function body. Example usage of the function shall be written after the three greater-than-signs >>> and one space character. Write the expected return value in the next line without including spaces after the value. For the function docstring, you may use either the Google-style docstring (recommended in this course), the PyCharm-style docstring, or Numpy-style docstring, but you need to be consistent.

```
def cube(num:int)->int:
    """Cubes the num. This is a Google-style docstring.

    Args:
        num (int): The number to be cubed.

    Returns:
        int: the cubed number.

    Examples:
```

```
        >>> cube(1)
        1
        >>> cube(2)
        8
        >>> cube(3)
        27
    """
```

```
def cube(num:int)->int:
    """Cubes the num. This is a PyCharm-style docstring.

    :param num: a number to be cubed.
    :type num: int
    :return: the cubed number.
    :type: int

    >>> cube(1)
    1
    >>> cube(2)
    8
    >>> cube(3)
    27
    """
```

```
def cube(num:int)->int:
    """Cubes the num. This is a Numpy-style docstring.
    Parameters
    ----------
    num : int
        a number to be cubed.

    Returns
    -------
    int
        the cubed number.

    Examples
    --------
    >>> cube(1)
    1
    >>> cube(2)
    8
    >>> cube(3)
```

```
    27
    """
```

Calculating a Triangle's Hypotenuse

Write a function `get_hypotenuse` that takes two arguments corresponding to the lengths of the sides (e.g. a and b) adjacent to the right angle of a triangle and returns the hypotenuse of the triangle ($\sqrt{a^2 + b^2}$). To perform the square root operation, you may take a value to its ½ power.
Start by writing the signature (the header with the description of the input and output) of the function: `def get_hypotenuse(a:float, b:float)->float:`
Then, write some examples in the docstring before writing the function body.

Converting Math Notation to Code

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

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

Start by writing the signature (the header with the description of the input and output) of the function: `def do_math(x:float, y:float)->float:`
Then, write some examples in the docstring before writing the function body.

Detecting Positive Numbers

Write a function `is_positive` that takes a single number as an argument and returns `True` when the argument is positive (excluding zero) and `False` otherwise. You must write this function using a relational operator (without any sort of conditional statement).
Start by writing the signature (the header with the description of the input and output) of the function: `def is_positive(num:int)->bool:`
Then, write some examples in the docstring before writing the function body.

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.**

Start by writing the signature (the header with the description of the input and output) of the function: `def both_positive(num1:int, num2:int)->bool:`
Then, write some examples in the docstring before writing the function body.

Test

Test by running your program. To run your program add the following lines of code at the bottom of your file:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print_hello()
    sum_numbers = get_numbers()
    print(sum_numbers)
```

Then run the program.
If any of your functions do not return the expected output as specified in your example, you will see an error reported on the screen. Fix your program until you see no errors.

Submission

Show your work to the instructor or TA first. Then, upload `lab3.py` to Canvas.