

ISTA 130

Programming Assignment 2

Functions

(submit via D2L Assignments)

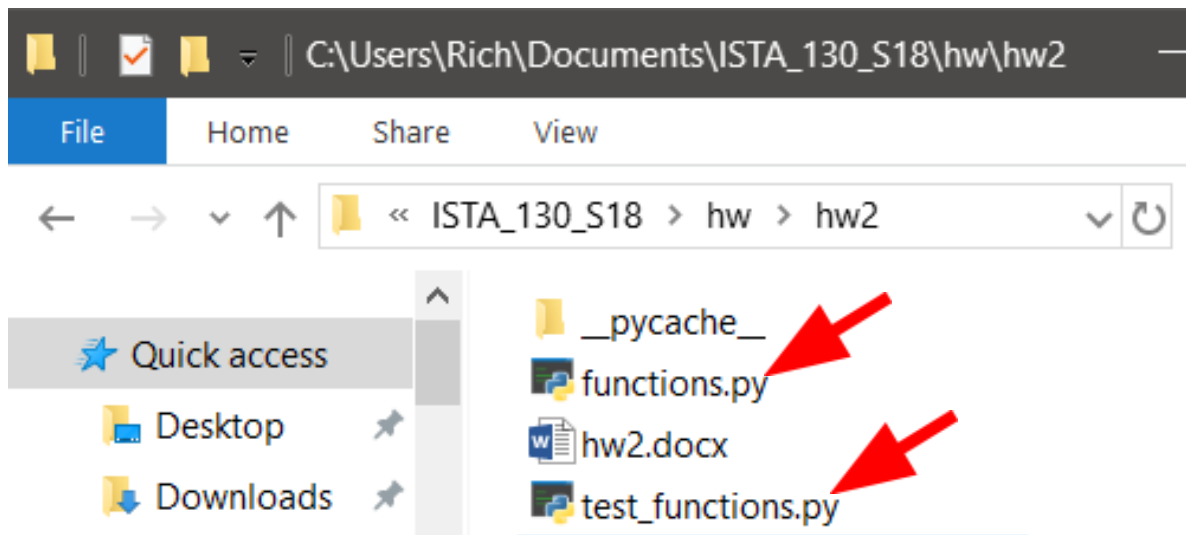
Please read the instructions below carefully and follow them closely. All problems (and parts of problems) are required except as noted in the instructions below.

Important: your filenames must be identical to the filenames given below. For any functions you are asked to write, the function signature (header) must be exactly as described in the instructions. That is, you must use the exact function names given in the instructions, you must have the parameters the instructions ask for, and the parameters must be in the order the instructions give.

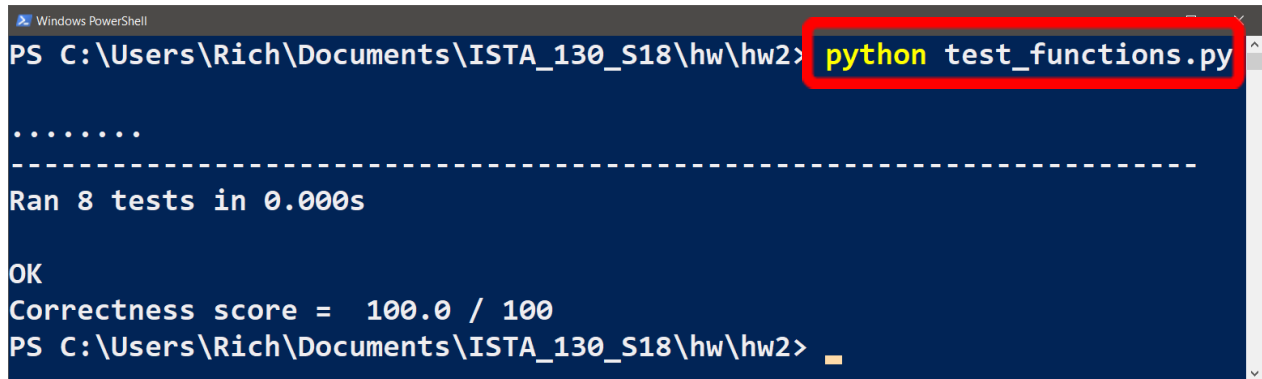
Also important: make sure you always save a backup of your work somewhere safe (such as UA Box or Google Drive).

Writing and Testing Your Code

For this assignment, you will create six functions that do not need turtle graphics. Download the file called `functions.py`. This file will contain your code. Download `test_functions.py` and put it in the same folder as your `functions.py` module:



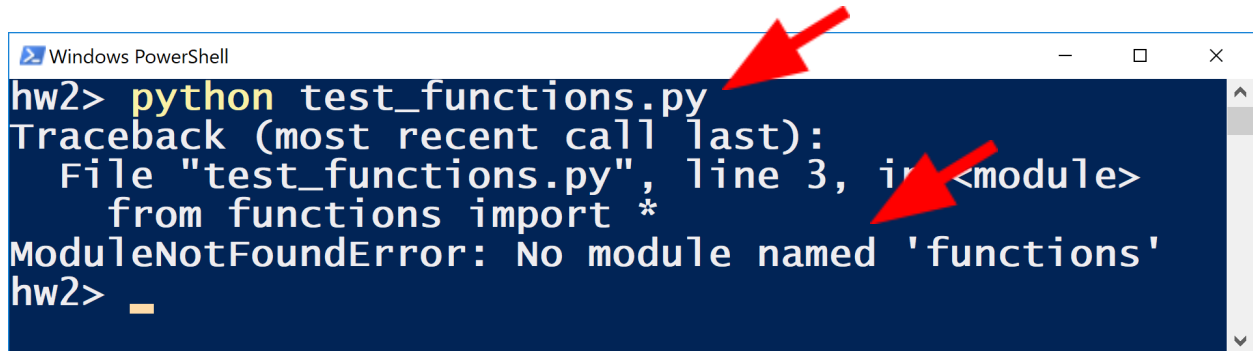
Run `test_functions.py` from the command line to see your current correctness score:



```
PS C:\Users\Rich\Documents\ISTA_130_S18\hw\hw2> python test_functions.py
.....
-----
Ran 8 tests in 0.000s

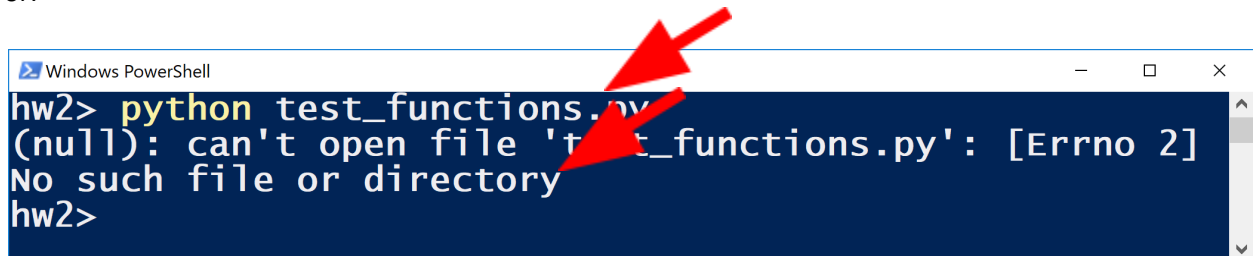
OK
Correctness score = 100.0 / 100
PS C:\Users\Rich\Documents\ISTA_130_S18\hw\hw2>
```

If the command line current working directory is not the folder containing the two files, you will get a file not found error. If there is a typo in either file name, you will get a file not found error. If both files are not in the same directory, you will get a file not found error. Examples:



```
hw2> python test_functions.py
Traceback (most recent call last):
  File "test_functions.py", line 3, in <module>
    from functions import *
ModuleNotFoundError: No module named 'functions'
hw2>
```

or:



```
hw2> python test_functions.py
(null): can't open file 'test_functions.py': [Errno 2]
No such file or directory
hw2>
```

An extremely common error, **especially on macs** (because of the way drag-and-drop works): having two copies of a file in different directories. The symptom of this is that you keep changing the code in your script to fix an error, but nothing changes when you run the test – because you're actually testing a different file than the one you are changing. Check your paths!

If you get a syntax error on a line, but it looks perfect, it almost always means you forgot a parenthesis on the previous line:

```
Windows PowerShell
hw2> python test_functions.py
Traceback (most recent call last):
  File "test_functions.py", line 3, in <module>
    from functions import *
  File "C:\Users\Rich\Documents\ISTA_130_S18\hw\hw2\functions.py", line 26
    return None
    ^
SyntaxError: invalid syntax
```

Once you get rid of all of the syntax errors, the test will run and it will also provide you with error messages. If you get a name not defined error in the output from the test, you have a typo in one of your variable names:

```
Windows PowerShell
-----
Traceback (most recent call last):
  File "test_functions.py", line 9, in test_print_word
    self.assertIsNone(print_word(3, 'banana'))
  File "C:\Users\Rich\Documents\ISTA_130_S18\hw\hw2\functions.py", line 24, in print_word
    for i in range(numbr):
NameError: name 'numbr' is not defined
-----
```

There are many other common mistakes. As you learn your own patterns of mistakes, debugging will become faster and faster. And you will get better and better at understanding the tests' error messages.

Many, many times your code will not pass because the text you print doesn't exactly match the spec:

```
Windows PowerShell
FAIL: test_erf_plus_gamma (__main__.TestFunctions)
-----
Traceback (most recent call last):
  File "test_functions.py", line 54, in test_erf_plus_gamma
    self.assertEqual('Result: 3.0\n', buf.getvalue().lower())
AssertionError: 'Result: 3.0\n' != 'result: 3.0\n'
- Result: 3.0
? ^
+ result: 3.0
? ^
```

The line starting with the – is what your code should produce (red arrow). The line starting with the + is what your code actually produces (blue arrow). The carets on the lines starting with ?'s show where

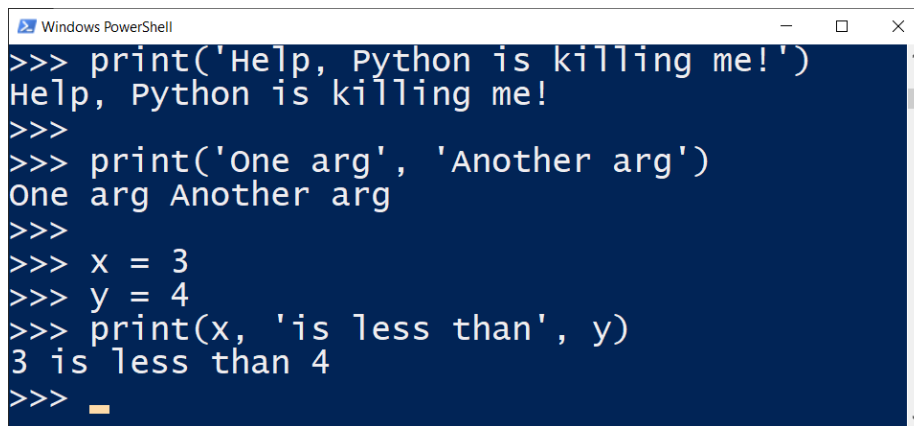
strings differ. In this case, we can see that I should be printing a capital R but I'm actually printing a small r. The mismatch is also shown in the red rectangle. This version is really handy when there are whitespace issues like a space hidden behind a tab, which will be a problem in a later homework, because it shows the actual whitespace characters (notice the newline character, `\n`).

Each of the functions is worth 10% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the homework specification (aka the spec).

The other 40% of your grade is for answering some questions as described below after the function specifications.

Function Writing Drills (100 points)

To do this homework, you need to understand how the built-in `print` function works. Open a Python shell with the `python` (`python3` on macs) command. Imitate this:

A screenshot of a Windows PowerShell window with a dark blue background. The window title is "Windows PowerShell". It shows a series of Python commands and their outputs. The commands are: `print('Help, Python is killing me!')`, `print('One arg', 'Another arg')`, `x = 3`, `y = 4`, and `print(x, 'is less than', y)`. The outputs are: "Help, Python is killing me!", "One arg Another arg", and "3 is less than 4". The prompt `>>>` is visible at the start of each line of code, and a cursor is at the end of the last line.

```
>>> print('Help, Python is killing me!')
Help, Python is killing me!
>>>
>>> print('One arg', 'Another arg')
One arg Another arg
>>>
>>> x = 3
>>> y = 4
>>> print(x, 'is less than', y)
3 is less than 4
>>> _
```

If we pass one argument, Python prints it. If we pass more than one argument separated by commas, Python prints them with spaces between them. Mess with this until you understand `print` so you can complete the following functions.

- 1.) Write a function called `chases` that has two string parameters. The first is a predator, the second a prey animal. You will use these to print a line per the examples below. You must match this output format (i.e. if you call your function with the same arguments as in the examples your output should look like the output in the examples). The code executed is in blue, the output produced is in green):

```
chases('dragon', 'human')
The dragon chases the human

chases('coyote', 'roadrunner')
The coyote chases the roadrunner
```

Your code should just fill in the predator and the prey in the correct spots. The rest prints the same every time.

- 2.) Write a function called `sum3` that takes three numerical arguments and prints their sum in this format (the code executed is in `blue`, the output produced is in `green`):

```
sum3(1, 2, 3)
The sum of the arguments is: 6

sum3(4, 5, 6)
The sum of the arguments is: 15
```

- 3.) Write a function called `forecast` that has no parameters. You can see that barebones `functions.py` that we provided imports the `random` module. Use `random.randrange(101)` to get an integer between 0 and 100. This is your forecasted chance of rain, which will be different every time. Print it in the following format (the code executed is in `blue`, the output produced is in `green`):

```
forecast()
Chance of rain today: 59 %

forecast()
Chance of rain today: 74 %
```

- 4.) Write a function called `radians` that takes an angle in degrees and prints it converted to radians. The formula for turning an angle in degrees into radians is:

$\text{radians} = \text{degrees} * \pi / 180.$

Use `math.pi` for π . You should round the result to three decimal places using the built-in `round` function. The documentation is [here](#). I suggest messing around in the shell with `round` to see how it works. Here are some examples of calling the function with different arguments. (The code executed is in `blue`, the output produced is in `green`):

```
radians(200)
The angle in radians is: 3.491

radians(30)
The angle in radians is: 0.524
```

- 5.) Write a function called `decimal` that takes a number. Print the decimal part in the format below. To get the decimal part, subtract the floor of the number from the number using `math.floor`. Round the result to three decimal places.

Here are some examples of calling the function with different arguments. (The code executed is in `blue`, the output produced is in `green`):

```
decimal(5.983)
```

Decimal part: 0.983

`decimal(5.9839)`

Decimal part: 0.984

- 6.) Write a function called `erf_plus_gamma` that takes a number and prints the error function of the number plus the gamma function of the number in the format shown below. Use `math.erf` and `math.gamma` and round the result to three decimal places.

Here are some examples of calling the function with different arguments. (The code executed is in blue, the output produced is in green):

`erf_plus_gamma(0.6)`

Result: 2.093

`erf_plus_gamma(0.22)`

Result: 4.395

Now it is time to use your code to answer some questions and fill in the answer section of `functions.py` (up near the top):

```
11  """
12  .... *****
13  .... Answers to questions:
14  .... 1)
15  .... 2)
16  .... 3)
17  .... 4)
18  .... 5)
19  .... 6)
20  .... 7)
21  .... 8)
22  .... *****
23  .... """
```

We are going to call some of your functions interactively and get the answers. Start the Python interpreter from the command line with the `Python` command or `Python3` on macs. Once the shell is started, import all of your functions:



```
Windows PowerShell
>>>
hw2_2020> python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

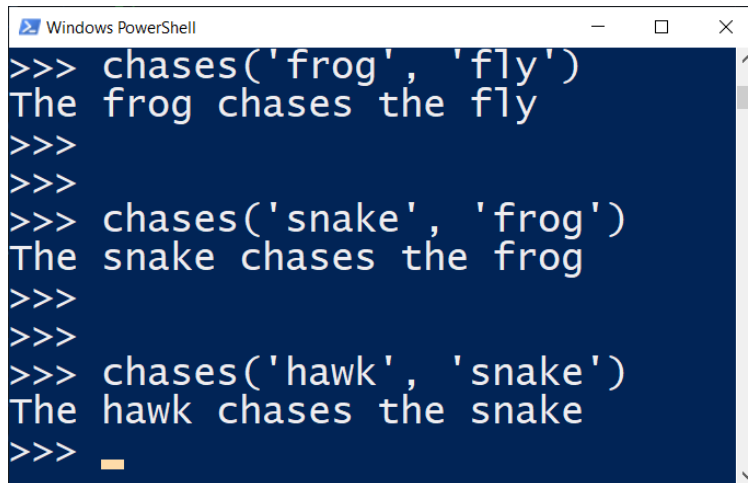
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license()" for more information.
>>> from functions import *
>>>
```

As a warmup, try calling `chases` with the following pairs of arguments:

```
"frog", "fly"  
"snake", "frog"  
"hawk", "snake"
```

Your console should look like (I hit `enter` a few times to put space between the calls):



```
Windows PowerShell  
>>> chases('frog', 'fly')  
The frog chases the fly  
>>>  
>>>  
>>> chases('snake', 'frog')  
The snake chases the frog  
>>>  
>>>  
>>> chases('hawk', 'snake')  
The hawk chases the snake  
>>> 
```

Ok, now answer these questions by calling your functions and passing in the appropriate arguments. Write the answers next to the corresponding number in the `functions.py` answer section.

- 1) What is the sum of 11034567, 999, and 76576?
- 2) Enter the command `random.seed(42)`. Call `forecast`. What is the chance it will rain?
- 3) Better verify that. Call `forecast` again. What's the new prediction?
- 4) What is 90° in radians?
- 5) 180° ?
- 6) What is `erf_plus_gamma` of 0.5?
- 7) What is `erf_plus_gamma` of 0.25?
- 8) What is `erf_plus_gamma` of 0.1?

Verify that your documentation makes sense and that you've added documentation to each of your functions, per the examples in `docstrings.py` or any of the posted example code.

Verify that your program works by running `test_functions.py`.

Upload your file to the Hw2 Assignments folder on D2L.