ISTA 130: Spring 2018
Programming Assignment 2
Functions and Loops

Due: Thursday, February 15th by 11:59 pm
(submit via D2L Assignments)

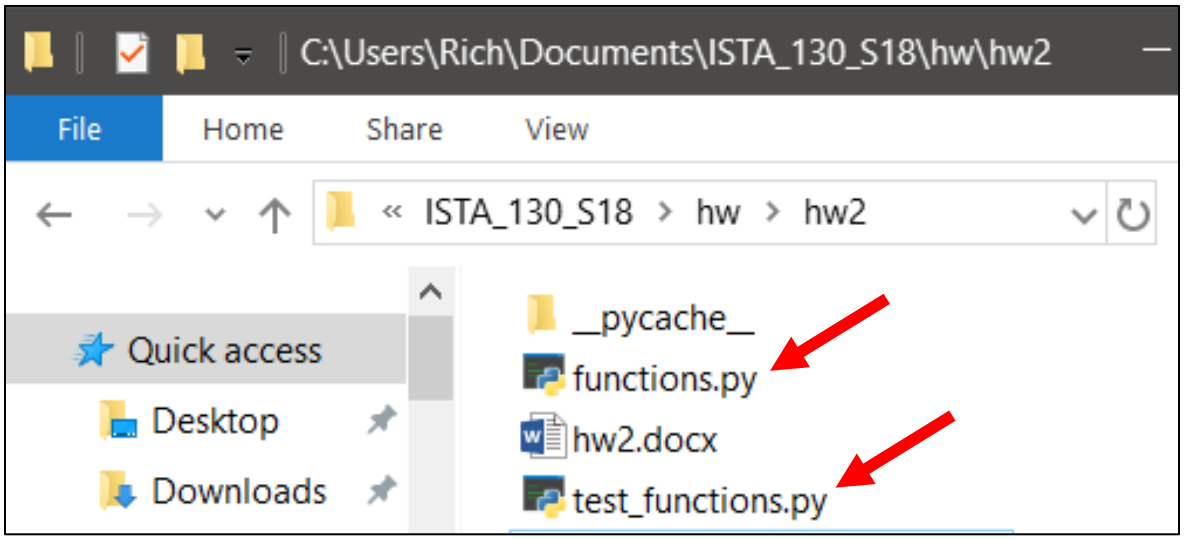
Please read the instructions below carefully and follow them closely. All problems (and parts of problems) are required except as noted in the instructions below.

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.

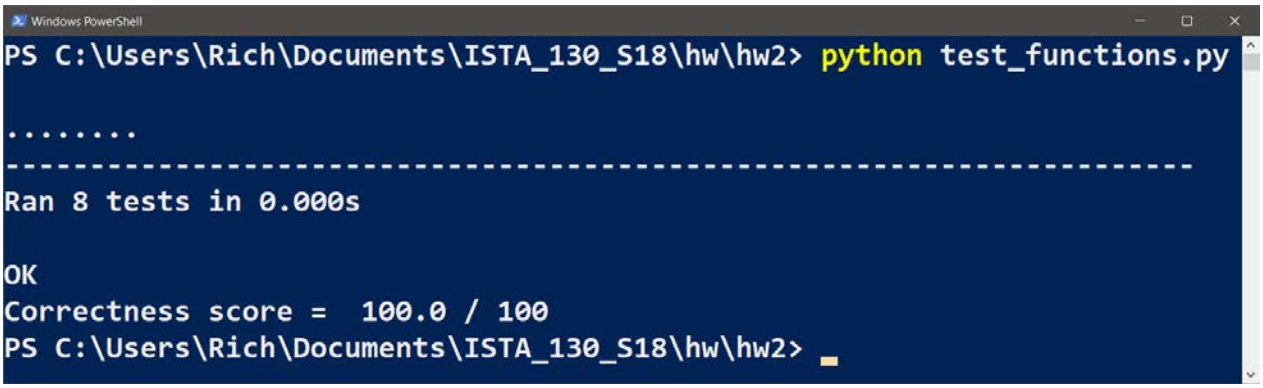
Also important: make sure you always save a backup of your work somewhere safe (such as UA Box or Google Drive).

Writing and Testing Your Code

For this assignment, you will create eight functions that do not need turtle graphics. Create a new file called `functions.py`. This file will contain your code. If you're starting with the template, delete the `turtle.getscreen().exitonclick()` line in `main()`. 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:



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.

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!

Each of the functions is worth 12.5% 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 test file we will use to grade your program may be different, but any code that passes this test will pass the grading test.

Function Writing Drills (100 points)

1.) Write a function called `print_word` that has two parameters. The first is for an integer (assume it will always be non-negative). The second is for a string. The function will print the string on the given number of lines, each time preceded by a line number and an arrow. See the examples below for the format of the output (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).

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

```
print_word(3, 'banana')
1 --> banana
2 --> banana
3 --> banana

print_word(4, 'mississippi')
1 --> mississippi
2 --> mississippi
3 --> mississippi
4 --> mississippi
```

2.) Write a function called `bacteria` that will print out the number of bacteria in a Petri dish as time goes by. The function has two parameters. The first is an integer giving the number of minutes it takes for a bacterium to split into two new bacteria. The second is an integer giving the number of bacterial generations to include in the output. Assume you always begin with a single bacterium in the dish and every bacterium always splits into exactly two bacteria at the end of each time period.

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

```
bacteria(10, 5)
after 10 minutes:  2 bacteria
after 20 minutes:  4 bacteria
after 30 minutes:  8 bacteria
after 40 minutes:  16 bacteria
after 50 minutes:  32 bacteria

bacteria(21, 3)
after 21 minutes:  2 bacteria
after 42 minutes:  4 bacteria
after 63 minutes:  8 bacteria
```

3.) Write a function called `convert_to_copper` that has three integer parameters. The first represents a number of gold coins. The second represents a number of silver coins. The third represents a number of copper coins. The function will print the numbers of each type of coin followed by the total value of all of the coins when converted to copper. The exchange rate for coins is:

5 copper pieces (cp) = 1 silver piece (sp)
10 silver pieces = 1 gold piece (gp)

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

```
convert_to_copper(5, 10, 7)
5 gp, 10 sp, 7 cp converted to copper is: 307 cp

convert_to_copper(15, 23, 12)
15 gp, 23 sp, 12 cp converted to copper is: 877 cp
```

4.) Write a function called `convert_from_copper` that takes a single integer argument representing a number of copper pieces. The function prints out the number of gold pieces (gp), silver pieces (sp), and copper pieces (cp) you would end up with if you first converted as many of the initial copper pieces to gold as possible and then converted as many of the remaining copper pieces as possible to silver pieces.

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

```
convert_from_copper(200)
200 copper pieces is: 4 gp, 0 sp, 0 cp

convert_from_copper(1107)
1107 copper pieces is: 22 gp, 1 sp, 2 cp

convert_from_copper(3242)
3242 copper pieces is: 64 gp, 8 sp, 2 cp
```

5.) This function requires a bit of preparation first:

Try entering each of the following in a Python shell:

```
print('Hobbit' * 10)
print('Hobbit' * 2)
print('Hobbit' * 1)
print('Hobbit' * 0)
```

Using what you just learned, write a function called `repeat_word` that has three parameters. The first is for a word (a string). The second is for an integer representing a number of rows. The third is for an integer representing a number of columns. The function prints the word in a number of rows equal to the value of the rows parameter and each row contains the word repeated a number of times equal to the columns parameter.

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

```
repeat_word('Goblin', 3, 5)
GoblinGoblinGoblinGoblinGoblin
GoblinGoblinGoblinGoblinGoblin
GoblinGoblinGoblinGoblinGoblin

repeat_word('Kobold', 5, 3)
KoboldKoboldKobold
KoboldKoboldKobold
KoboldKoboldKobold
KoboldKoboldKobold
KoboldKoboldKobold
```

6.) Using what you learned in the previous question, write a function called `text_triangle` that takes an integer parameter and prints X's in a triangle shape.

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

```
text_triangle(3)
X
XX
XXX

text_triangle(10)
X
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX
XXXXXXXX
XXXXXXXXX
XXXXXXXXXX

text_triangle(0)
```

There should be no spaces in front of each X. The last one is the tricky one! If the argument is <= 0, print one blank line. Make sure you print the correct amount of X's.

7.) Write a function called `surface_area_of_cylinder` that takes two arguments. The first is a float representing the radius of a cylinder. The second is a float representing the height of a cylinder. The function calculates and prints the surface area of a cylinder with the given radius and height. You will need to import the `math` module and use the `math.pi` constant. The formula is:

$$SA = 2\pi r^2 + 2\pi rh$$

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

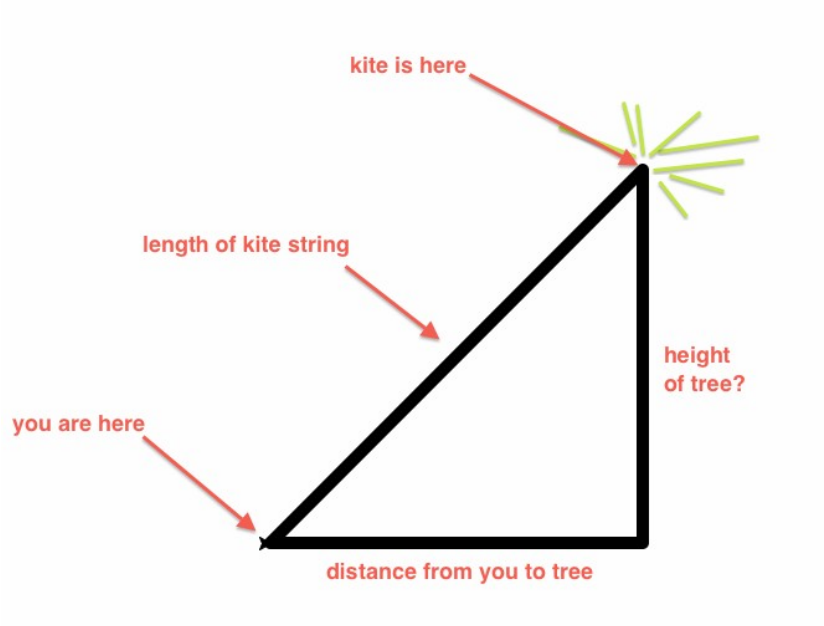
```
surface_area_of_cylinder(10.0, 10.0)
The surface area of a cylinder with radius 10.0 and height 10.0
is 1256.6370614359173

surface_area_of_cylinder(0.0, 1.0)
The surface area of a cylinder with radius 0.0 and height 1.0 is
0.0
```

8.) The `math` module has a function called `sqrt` that returns the square root of its parameter. It is used like this: `math.sqrt(100)`.

Imagine you are flying a kite and the kite gets caught in the top of a perfectly straight palm tree. The string pulls free from the kite leaving you with the full length of string and your kite stuck in the tree. You measure the distance from you to the base of the tree. Given the length of the kite string and the distance from you to the base of the tree you can calculate the height of the tree using the Pythagorean Theorem:

$$a^2 + b^2 = c^2$$

In this case *a* is the distance from you to the tree, *b* is the unknown height of the tree, and *c* is the length of the kite string.



Write a function called `tree_height` that takes two arguments. The first is a float representing the distance from you to the base of the tree. The second is a float representing the length of the kite string. The function will calculate and print the height of the tree as shown in the examples below. The function will calculate and print the height of the tree.

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

```
tree_height(300, 500)
Kite string: 500
Distance: 300
Height: 400.0
```

```
tree_height(100, 141.421356)
Kite string: 141.421356
Distance: 100
Height: 99.99999966439368
```

Since we are not using turtle graphics in this program, you will lose points if your `main` function still has this line (you don't actually need a `main` function for this assignment):

```
turtle.getscreen().exitonclick()
```

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 Homework 2 Assignments folder on D2L.