ISTA 130: Spring 2020
Programming Assignment 4
Returning Values and Simple Conditions

Due: Thursday, February 27th 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 your D2L locker, dropbox.com, or UA Box).

This too is important: You are not just graded on whether your code produces the same result as the examples show. You should be using what you've learned to do your best to write good code. For example, things like poorly named variables/functions, duplicating code where you could instead use a loop, and missing documentation will cost you points.

Function Writing Drills (100 points)

In this section you will write a number of functions that do not need turtle graphics. You can test the correctness of your code by placing `test_conditions.py` in the same folder as your `conditions.py` module (see below) and running the test code.

Create a new file called `conditions.py`. If you're using the template, get rid of the `turtle.getscreen().exitonclick()` line. Implement these 10 functions (10 points each):

1.) This function takes a bit of preparation first. Try entering the following in a shell:

```
len('mumps')
len('Yellow-bellied Sapsucker')
len('')
phrase = 'That is silly!'
length = len(phrase)
print(length)
```

Using what you just learned about the `len` function, write a function called `word_length` that has two parameters. The first will be a string and the second will be an integer. The function first prints a message about the relationship between the length of the word and the integer, as shown in the following examples.

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

```
word_length('liversnaps', 7)
Longer than 7 characters: liversnaps

word_length('earwax', 5)
Longer than 5 characters: earwax

word_length('chickenfat', 10)
Exactly 10 characters: chickenfat

word_length('Gross!', 13)
Shorter than 13 characters: Gross!
```

2.) Write a function called `stop_light` that determines whether a stop light should change color, and, if so, what color it should change to. It takes two arguments. The value of the first will be either `"green"`, `"yellow"`, or `"red"`. This represents the stop light's current color. The second parameter tells the function how long this color has been showing. If green has been showing longer than 60 seconds, return `"yellow"`. If yellow has been showing longer than 5 seconds, return "red". If red has been showing longer than 55 seconds, return "green". If the color hasn't been showing long enough (e.g. green has been showing for 17 seconds), return the current color.

Here are some examples of calling the function with different arguments. (The code executed is in **blue**, the value returned by the function is in **green**):

```
stop_light('green', 61)   -->   'yellow'
stop_light('yellow', 5)   -->   'yellow'
stop_light('yellow', 6)   -->   'red'
stop_light('red', 12)     -->   'red'
stop_light('red', 56)     -->   'green'
```

3.) Write a function called `is_normal_blood_pressure` that has two integer parameters. The first represents systolic blood pressure (the top number in a blood pressure reading). The second represents diastolic blood pressure (the bottom number in a blood pressure reading). The function should return `True` if systolic is less than 120 and diastolic is less than 80 (i.e. blood pressure is normal). Otherwise it returns `False`.

Here are some examples of calling the function with different arguments. (The code executed is in **blue**, the value returned by the function is in **green**):

```
is_normal_blood_pressure(120, 80)   -->   False
is_normal_blood_pressure(119, 80)   -->   False
is_normal_blood_pressure(119, 79)   -->   True
is_normal_blood_pressure(120, 79)   -->   False
```

4.) Write a function called `doctor` that has no parameters. The function will ask the user to enter his/her systolic blood pressure reading. It will then ask for the diastolic reading. The function then prints either "Your blood pressure is normal." or "Your blood pressure is high." depending on the values entered. This function should use the function you wrote in the previous question.

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

```
doctor()
Enter your systolic reading: 119
Enter your diastolic reading: 79
Your blood pressure is normal.

doctor()
Enter your systolic reading: 133
Enter your diastolic reading: 79
Your blood pressure is high.
```

5.) Write a function called `pants_size` that has a single parameter (the value will be an integer) representing a person's waist size in inches. The function returns a string. The string returned will be either `"small"`, `"medium"`, or `"large"` depending on the parameter value. Waist measurements that are 34 inches or larger should return large. Measurements that are 30 inches or larger, but not large enough to be in the large category, should return medium. Anything smaller should return small.

Here are some examples of calling the function with different arguments. (The code executed is in **blue**, the value returned by the function is in **green**):

```
pants_size(38)     -->    'large'
pants_size(34)     -->    'large'
pants_size(33)     -->    'medium'
pants_size(29)     -->    'small'
pants_size(-20)    -->    'small'
pants_size(2000)   -->    'large'
```

6.) Write a function called `pants_fitter` that takes no arguments. The function should first ask the user for his/her name. It then greets the user by name. Next it asks the user for his/her waist size in inches (a positive integer). It then asks the user how many pairs of pants he/she would like to buy (a positive integer). Next it asks what type of pants the user wants to buy (either "regular" or "fancy"). Next it calculates the cost of the pants (integer). Regular pants cost $40 per pair. Fancy pants cost $100 per pair. Finally it prints out the number of pairs, the size, the type, and the total cost. The following examples show the format that your prompts and output should be in. This function should use the function you wrote in the previous question.

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

```
pants_fitter()
Enter your name: Ziggy
Greetings Ziggy welcome to Pants-R-Us
Enter your waist size in inches: 34
How many pairs of pants would you like: 2
Would you like regular or fancy pants? fancy
2 pairs of large fancy pants: $ 200

pants_fitter()
Enter your name: Elmer
Greetings Ziggy welcome to Pants-R-Us
Enter your waist size in inches: 31
How many pairs of pants would you like: 10
Would you like regular or fancy pants? regular
10 pairs of medium regular pants: $ 400

pants_fitter()
Enter your name: Minnie
Greetings Ziggy welcome to Pants-R-Us
Enter your waist size in inches: 12
How many pairs of pants would you like: 1
Would you like regular or fancy pants? fancy
1 pairs of small fancy pants: $ 100
```

7.) Write a function called `digdug` that takes a single argument number (assume it will always be a positive integer). For every integer from 1 up to and including number, the function will print a message if warranted. If the integer is evenly divisible by 3 the function will print `"dig"`. If the integer is evenly divisible by 5 it prints `"dug"`. If the integer is evenly divisible by both 3 and 5 it prints `"digdug"`. If the integer is not divisible by either 3 or 5 it does not print anything.

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

```
digdug(2)

digdug(3)
3 : dig

digdug(5)
3 : dig
5 : dug

digdug(15)
3 : dig
5 : dug
6 : dig
9 : dig
10 : dug
12 : dig
15 : digdug
```

8.) Write a function called `beef_type` that takes a single parameter, `percent_lean`. If the value of `percent_lean` is less than 78%, return `"Hamburger"`. If it is at least 78% and less than 85%, then return `"Chuck"`. At least 85% but less than 90% return `"Round"`. 90-95% inclusive return `"Sirloin"`. If `percent_lean` doesn't fall within one of these ranges, return `"Unknown"`.

Here are some examples of calling the function with different arguments. (The code executed is in **blue**, the value returned by the function is in **green**):

```
beef_type(91.2)     -->    "Sirloin"

beef_type(78)       -->    "Chuck"

beef_type(87)       -->    "Round"

beef_type(95.1)     -->    "Unknown"
```

9.) Write a function called `species_height` that takes 2 arguments. The first is either `"Human"` or `"Klingon"`. The second is a positive float representing the height (in inches) of this human or Klingon. In this homework assignment, the average human height is 67 inches. The average Klingon height is 71 inches. For the parameters given, print out if it is above, below or at the average height for its species.

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

```
species_height("Human", 62.1)
Below Average

species_height("Klingon", 73)
Above Average

species_height("Klingon ", 71)
Average
```

10.) Write a function called `sooner_date` that has 4 integer parameters. The first is a number between 1 and 12 (inclusive) that represents a month. 1 is January, 2 is February, etc. The second is a number between 1 and 31 (inclusive) that represents a day. The third parameter is another integer representing a month and the fourth is another integer parameter representing a day. So essentially you have 2 dates (the first 2 parameters and the second 2 parameters). Figure out which date would come sooner, then print out that date in the format month / day.

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

```
sooner_date(1, 1, 1, 2)
1 / 1
```

```
sooner_date(2, 5, 1, 3)
1 / 3

sooner_date(8, 25, 7, 30)
7 / 30
```

11.) Verify that your program works by running `test_conditions.py`.

12.) Upload your file to the Hw4 Assignments folder on D2L.