Python Functions Worksheet

Instructions:

1. Log into Grok
2. Go to [https://groklearning.com/learn/python-turtle-playground/1/0/](https://groklearning.com/learn/python-turtle-playground/0/0/)
3. Read the directions for each task very carefully before you begin working on it.
4. Every time the directions say “take a screenshot” of something, do it and then copy it into the proper space in this document.

This worksheet is intended to be both an introduction to functions and to refresh your memory on what you have learned so far in python. Work through each task with the help of a partner (being sure to type/run each bit of code yourself).

# Problem 1:

Use the code below to answer the following questions and complete the required tasks.

```

def do\_something\_1(x, y):

z = x + y

return z

result = do\_something\_1(5, 6)

print(result)

```

**BEFORE** running the program, answer the following questions:

1. How many inputs to the `do\_something\_1()` function are there? What is the type (e.g., string or int) of each input?
2. What is the output/result of the function (if any) and what is its type?
3. What do you think will be printed to the screen when you run the code?
4. What would be a better name for the function than `do\_something\_1()`?
5. If you didn’t have the last two lines of the program, what would happen when you run the code?

**NEXT,** copy the code into Grok and run it. Look under the **output tab** for the results.

1. Were your predictions correct? If not, what mistakes did you make?
2. Modify the code **inside the function** so it does multiplication rather than addition. Also, add some code **inside the function** so that if the result of the multiplication is greater than 1000, the program prints out “whoa, big number!”
3. A “function call” is where you call/invoke the function (I do this on the second to last line of the program above). Create two “function calls” of your own below my function call. One or your function calls should result in the “whoa” being printed out and one should not.
4. Take a screenshot of **your code in Grok** and paste it here.

# Problem 2

Use the code below to answer the following questions and complete the required tasks.

```

def do\_something\_2(x):

output = ""

for i in range(0, x, 2):

output = output + str(i) + ", "

return output + "who do we appreciate?"

result = do\_something\_2(10)

print(result)

```

**BEFORE** running the program, answer the following questions:

1. How many inputs to the `do\_something\_2()` function are there? What is the type (e.g., string or int) of each input?
2. What is the output/result of the function (if any) and what is its type?
3. What do you think will be printed to the screen when you run the code?
4. What would be a better name for the function than `do\_something\_2()`?
5. If you didn’t have the last two lines of the program, what would happen when you run the code?

**NEXT,** copy the code into the Grok window and run it.

1. Were your predictions correct? If not, what mistakes did you make?
2. Modify the code so that it **starts** the cheer at the number 2.
3. Modify the code so that it **includes** the number that was given as the input to the function in its cheer.
4. This code is an example of the “accumulator pattern”. Why?
5. Take a screenshot of **your code in Grok** and paste it here.

# Problem 3

Use the code below to answer the following questions and complete the required tasks.

```

from turtle import \*

def make\_shape(x):

for i in range(4):

forward(x)

left(90)

make\_shape(30)

make\_shape(60)

```

**BEFORE** running the program, answer the following questions:

1. How many inputs to the `make\_shape()` function are there? What is the type (e.g., string or int) of each input?
2. What is the output/result of the function (if any) and what is its type?
3. What do you think will be printed to the screen when you run the code?
4. What would be a better name for the input variable than `x`?

**NEXT,** copy the code into the Grok window and run it.

1. Were your predictions correct? If not, what mistakes did you make?
2. Alter the code to make it create a triangle. What did you change?
3. Alter the code to make it create a hexagon. What did you change?
4. Alter the function so it has another parameter (num\_sides) that determines the number of sides a shape should have. Then, use that parameter to create the desired shape. For example, `make\_shape(100, 8)` should make an octagon that has sides of length 100. Note: the trick is to turn 360 / num\_sides at each side. Test your function with 2 calls.
5. Take a screenshot of **your code in Grok** and paste it here.
6. (Optional: 5 bonus points on quiz) Call the `make\_shape()` function in a loop so that you create multiple shapes with 3 - 20 sides. Take a screenshot of the **resulting image** and paste it here.

Problem 4 (Optional: 5 bonus points on quiz)

Functions can call themselves. This is called recursion. Use the *recursive* code below to answer the following questions and complete the required tasks.

```

def call\_me(num):

if num == 0:

print("All done")

else:

print(num)

call\_me(num - 1)

call\_me(10)

```

**BEFORE** running the program, answer the following questions:

1. How many inputs to the `call\_me()` function are there? What is the type (e.g., string or int) of each input?
2. What is the output/result of the function (if any) and what is its type?
3. Based on the definition of recursion above, why is the `call\_me()` function considered recursive?
4. What do you think will happen when you run the code?

**NEXT,** copy the code into the Grok window and run it.

1. Was your prediction correct? If not, what mistake did you make?
2. The bit of code that checks for n == 0 is called the “base case”. Why is the base case necessary?
3. Alter the code so that it counts **UP** to a given number. Take a screenshot of **your code in Grok** and paste it here.