Python and Mathematics for Machine Learning

Python function definition:
Python functions (or methods) are segments of code that are placed in a program "container" so that it can be called from within a main programming sequence. Functions are created primarily for efficiency if a segment of code is expected to be used in many places, for example, in calculating a distance based on two sets of coordinates.

Python function header:
The syntax of a Python function header, also known as the Application Programming Interface or API, includes the "def" keyword, then the function name and optional list of input parameters to the function enclosed within parentheses, then a colon following the parenthesis. The programmer simply calls the function by typing the name of the function and passing the parameters, if any, enclosed in parentheses.

Python function scope:
The scope of the function defines what can be done within the function, that is, when the function is called, and is identified by any code that is indented relative to the function header. The function may define its own variables, called local variables, which are only accessible within the function and not outside once the function has returned to the main program. The function may access global variables, which are ones that are accessible and updatable by all code in a Python file. This means the main code and all code within the scope of a function may access and update a global variable.

Return types:
Functions may perform operations and print out results to the console or to a file, may perform operations on and update global variables, or may perform operations on local variables and return the results to the code which called the function. If the function simply performs an operation and prints out a result or assigns it to a global variable, it may be defined as "void" which means there is no return value. If the function returns values, a "return" statement must be present in the code with the parameter or parameters that are returned. The code which calls the function then would assign a variable to the function name (i.e. the function call) and if there are multiple return values, it would assign multiple variables to the function name.

Doctest:
Doctest is a library that is included in Python to allow generation of testing scenarios (a.k.a. test vectors) for your functions. This functionality is extremely useful and important for any engineer or computer scientist developing code as it allows module level (or unit) testing of functions and methods. The tests are output from the standard Python interpreter and cut/pasted into "docstrings". Here's a reference link. To insert a doctest in your code, you'll need to import doctest. Then to insert a test use three double quotes to delimit the test (e.g. put 3 quotes before and after the test code. You may insert comments using the standard comment hashtag symbol (#) to document the tests you are running. Then use three greater-than symbols to identify the test code (e.g. setting up variables and calling the function. Finally, list the expected

output and run doctest.testmod(). A message will then be printed indicating the pass or fail condition of the test.

Examples:

The following code shows some simple function examples illustrating the function header syntax and various parameter lists, and single and multiple return values. Please familiarize yourself with these syntax formats. Also note the use of doctest at the end of the code to test a couple of the functions.

```
28 import doctest
29
30 def helloWorld():
31   print("Hello World!!")
32
33 def helloStudent (name):
34   print('Hello ', name)
35
36 def helloStudents (name1, name2):
37   print('Hello', name1, 'and',name2)
38
39 def helloTeacher (name):
40   return 'Hi, ' + name + ', how are you?'
41
42 def helloPrincipal (name1, name2):
43   return 'Hi, ' + name1, 'Hi, ' + name2
44
45 helloWorld()
46 helloStudent('Jim')
47 helloStudents('Jim', 'Jane')
48 print(helloTeacher ('Jim'))
49 response1, response2 = helloPrincipal('Jim', 'Jane')
50 print(response1)
51 print(response2)
52
53 """
54 #Test the following functions:
55 #helloStudent(name), helloStudents(name1, name2)
56    >>> helloStudent('Jim')
57    Hello Jim
58    >>> helloStudents('Jim', 'Jane')
59    Hello Jim and Jane
60 """
61
62 doctest.testmod()
```

```
Hello World!!
Hello  Jim
Hello Jim and Jane
Hi, Jim, how are you?
Hi, Jim
Hi, Jane
**********************************************************************
File "__main__", line 5, in __main__
Failed example:
    helloStudent('Jim')
Expected:
    Hello Jim
Got:
    Hello  Jim
**********************************************************************
1 items had failures:
   1 of   2 in __main__
***Test Failed*** 1 failures.
TestResults(failed=1, attempted=2)
```

Please note the doctest.testmod() call on line 62. This call executes the functionality in lines 54-60 which define the function calls, input conditions (i.e. the test vectors), and the expected outputs. Here we see the three double quotes to start and end the test module. The hashtags identify commented code. The >>> execute the function call and the following lines are the expected outputs. Now note the output from running the code cell. Here we see the "TestResults" output identify that one test failed and 2 were attempted (i.e. 1 failed and 1 passed). The failed test is listed above and shows the expected and returned result. Here we see there is an extra whitespace inserted between the 'Hello' and 'Jim' in the received output.

Assignment:

1. Create a Python program which performs the following:
    a. Contains a function with the API: *def ageCategory1 (age):*
    b. Prints a message inside the function (your choice of an appropriate message) for the following age categories:
        i. Less than 5
        ii. Between 5 and 10 inclusive
        iii. Between 11 and 20 inclusive
        iv. Between 21 and 64 inclusive
        v. Greater than 64
    c. Calls the function from a main program.
    d. Upload a screenshot of your code AND your code output.

2. Create a Python program which performs the following:
    a. Contains a function with the API: *def ageCategory2 (age):*
    b. Returns a string with the messages you created for the categories in #1.
    Note, your function must implement a return statement (fruitful function)
    c. Calls the function from a main program and prints the returned output.
    d. Upload a screenshot of your code AND your code output.

3. Create a Python program which performs the following:
    a. Contains a function with API: *def ageCategory3 (*profile):* where *profile contains a comma-separated list of names (Strings) and ages (integers)
    Note, your function must implement arbitrary arguments
    b. Returns two strings, each with a message you created for the categories in #1 that includes the input name and associated message.
    c. Calls the function from a main program and prints the returned outputs
    d. Upload a screenshot of your code AND your code output here.
    e. For example, your main program should call *ageCategory3 (Jim, 22, Jane,3)* and your output could be something as follows:
    Jim, you are of legal age!
    Jane, you are a toddler!
    Note, you can assume the inputs contain 2 pairs of name, age. (we will see how to do this with lists in the next module). Hint, use the len function to determine how many pairs are passed to the function, or index into the profile variable using [0] for item 1, [1] for item 2, etc.

4. Add doctest functionality to test each of your functions from parts 2 or 3. Rerun your program and verify your functions match the test vectors (the returned values match the expected results you built into your doctest functionality. Upload a screenshot of your doctest console output.