CS5001: Lab 4. Due on Friday, Oct-6-2023.

Name(s): Xujia Qin

Email(s): qin.xuj@northeastern.edu

You can work on this lab either individually or in small group of two or three students. If working in a group, include names of all the students in the submission PDF.

Getting credit for this lab. This lab handout has several empty boxes that prompt you to answer a question. As part of the lab, you are to write the answers to these questions inside the boxes/blanks. When you are finished, you should create a PDF and upload it on Canvas. If you don't finish, you have until 11:59 PM on Friday, Oct-6-2023 to submit.

What computer to use? If your primary computer is a laptop, bring it to the lab to work on, as lab is an excellent opportunity to get started with Python on your machine. You should follow the instructions on the course website. Ask a TA for help if you have problems with your installation. If you prefer, you could also use one of the machines in the lab room to work on this lab assignment.

Lab Materials. Lab materials can always be found on Canvas under the appropriate lab posting.

For today's lab, you need this handout (which is also online), plus the files DemoMath.py.

The following questions are about the Python file DemoMath.py:

```
# DemoMath.py
# CS 5001 (ad.mishra@northeastern.edu)
# 2023
""" Examines a function that computes approximate square roots."""


import math

def sqrt(x):
    """Returns an approximate square root of x as float.

    Performs five steps of rectangle averaging.

    Precondition: The value of x is a positive number."""
    # As explained in lecture, imagine you have an x-by-1 rectangle,
    # which will thus have area x.
    length = float(x)

    # As explained in lecture, we change the length of our rectangle
    # (and then, implicitly, the width to keep the area the same), to make a
    # "more square" rectangle with the same area.
    length = (length + x/length)/2
    length = (length + x/length)/2
    length = (length + x/length)/2
    length = (length + x/length)/2
    length = (length + x/length)/2

    # If the "rectangle" with area x were now a square, then the length
    # of its side would be the sqrt.
    return length



def fourth_root(x):

    """Returns an approximate fourth root of x as float.

    Precondition: The value of x is a positive number."""
    return x # change this for question 6!
```

```
# Application Script
if __name__ == '__main__':
    """ A keyboard input framework for checking out sqrt.
    """

    x = float(input('Enter a number whose square root you want: '))
    y1 = math.sqrt(x)
    y2 = sqrt(x) # question 2: change this to "y2 = DemoMath.sqrt(x)"

    print ('\n\n          x = %5.2f' % x) # question 7: unindent this
    print ('math.sqrt(x) = %15.12f' %y1)
    print ('     sqrt(x) = %15.12f' %y2)
    z1 = math.sqrt(math.sqrt(x))
    z2 = fourth_root(x)
    print ('\n  True 4th Root = %16.12f' %z1)
    print ('  fourth_root(x) = %15.12f' %z2)
```

(1) What is displayed if you ask for the square root of 9? Is this program's sqrt(9) correct according to math.sqrt(9)? What is displayed if you ask for the square root of 900? Is this program's sqrt(900) correct according to math.sqrt(900)? **(5 + 5 points)**

if I ask for the square root of 9, the output is:

```
        x =  9.00
math.sqrt(x) =  3.000000000000
    sqrt(x) =  3.000000001397

 True 4th Root =   1.732050807569
 fourth_root(x) =  9.000000000000
```

This program's sqrt(9) is not correct, but really close according to math.sqrt(9)

if I ask for the square root of 900, the output is:

```
        x = 900.00
math.sqrt(x) = 30.000000000000
    sqrt(x) = 38.054079122334

 True 4th Root =   5.477225575052
 fourth_root(x) = 900.000000000000
```

This program's sqrt(9) is not correct, and the variation is not admissable according to math.sqrt(9)

(2) We used “import math” instead of “from math import *” or “from math import sqrt”. This question justifies our choice, and explores the implications of the various ways to use import statements.

What happens if you say y2 = DemoMath.sqrt(x) instead of y2 = sqrt(x) in the Application Script? Explain in a way that demonstrates that you understand “what Python thinks you are referring to” (and why it is unsuccessful in finding it) when encountering the word “DemoMath”. **(10 points)**

```
Traceback (most recent call last):
  File "/Users/cathyqin/Desktop/DemoMath.py", line 46, in <module>
    y2 = DemoMath.sqrt(x) # question 2: change this to "y2 = DemoMath.sqrt(x)"
         ^^^^^^^^
NameError: name 'DemoMath' is not defined
```

If I want to use a function named sqrt from a custom module named DemoMath instead of the built-in math.sqrt function, it needs to make sure that DemoMath is properly defined and imported in the application script first, otherwise the program cannot find sqrt function without DemoMath predefined.

From the above, you should understand: why would we get a “name ’math’ is not defined” error for the line y1 = math.sqrt(x) if we had used either of the two alternate import statements mentioned above? **(10 points)**

Importing the math module first will make its functions and objects available for use in the script.
Without the import statement, Python doesn't know about the math module, leading to the "name 'math' is not defined" error.

(3) What happens if you comment out the return statement in sqrt? Explain — specifically, write down (and remember for future programming experience) the error message you (should) get, and explain why variable y2 has the value None instead of some float value, and so the print statement that is expecting a float causes an error. **(10 points)**

```
          x =  9.00
 math.sqrt(x) =  3.000000000000
 Traceback (most recent call last):
   File "/Users/cathyqin/Desktop/DemoMath.py", line 50, in <module>
     print ('     sqrt(x) = %15.12f' %y2)
           ~~~~~~~~~~~~~~~~~~~~~~~~~~^~~
 TypeError: must be real number, not NoneType
```

In this example, sqrt doesn't have a return statement.
When we call sqrt(), it will execute the code inside the function but won't explicitly return any value.
As a result, the value of result will be NoneType.

Basically because if we expect a function to return a specific value, we should include a return statement with that value.
If we omit the return statement, the function will still run, but it won't provide a meaningful return value (in this case, i.e. NoneType), which can lead to unexpected behavior in the code output.

(4) Why is it necessary to have the statement length = float(x) in the function body of sqrt? **(10 points)**

Because this initialization relaxes the x variable to be the constant input, and can avoid being updated when going into the iterative code below. Instead, we necessarily assign the initial value to variable length, and let it iterate with a fixed x, converging to an approximate square root of x in the end.

(5) How could sqrt be modified so that it could handle the input 0? **(10 points)**

```
We can set a base case check inside the sqrt function
"""The value of x is a non-negative number."""
    if x == 0:
        return 0.0  # Handle the case where x is 0
```

(6) Redo the body of fourth root so that it makes effective use of sqrt and returns an approximate fourth root of x. **(10 points)**

```
def fourth_root(x):
    """Returns an approximate fourth root of x as float.

    Precondition: The value of x is a positive number."""
    return sqrt(sqrt(x))
```

(7) Unindent the line that has ”question 7” in a comment and run the program. Look at the line number of the error message you get. Why is Python reporting a problem with a line after the one you un-indented? **(15 points)**

```
  File "/Users/cathyqin/Desktop/DemoMath.py", line 49
    print ('math.sqrt(x) = %15.12f' %y1)
IndentationError: unexpected indent
```

Because here's an issue with inconsistent indentation in the print statements after I un-indented the line with x = %5.2f. Python uses indentation to determine the structure of the code block, in this case, all print statements should be under the application’s main script with the same indentation, otherwise it will cause an error with the new code block unindented.

(8) Why does the program no longer output anything if you comment out the line if __name__ == __main__ but doesn’t give an error, either? (Hint: the program isn’t actually executing the remaining lines. Why?) **(15 points)**

Python does not produce an error because commenting out if __name__ == '__main__': simply means that the code within that block won't execute when the script is run directly, because the program logic is not triggered. It doesn't violate any Python syntax rules, so it doesn’t give an error in the output.