Python Programming

Homework 2

Instructor: Dr. Ionut Cardei

The purpose of this programming assignment is to:
1. write simple algorithms and implement them in Python
2. write Python programs using strings and functions that manipulate strings
3. practice writing functions

For full credit:
- make sure you follow exactly all requirements listed in this document,
- the program must have all required features.

Write your answers into a new Word document called **h2.doc**.
Write your full name at the top of the file and add a heading before each problem solution.

Don't forget to read the **Important Notes** at the end of this file.

Homework 2 has three problems.

Problem 1. Pythagorean Numbers

The lengths (a,b,c) of the three sides of a right triangle are related by this well-known formula:

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

A Pythagorean triple consists of three **positive integer** numbers bound by the above relation. An example of a triple is (3,4,5). Others are (k*a, k*b, k*c), for any positive integer k and some other valid Pythagorean triple (a,b,c).

a) Write an algorithm that reads from the terminal a positive integer *n* and that computes and displays to the terminal **all possible** Pythagorean triples (a,b,c), where 0<a,b,c<=n.
Write your algorithm in file h2.doc in pseudo-code. For this part of the problem do not write a program, but just the algorithm.
To get full credit make sure the algorithm has ALL the features required, as explained in the class material.

b) Implement the algorithm from part a) in a Python program called **p1.py**. Write code that tests the algorithm. Insert the program in file h2.doc.

Problem 2. Duplicated Substrings

A substring is a contiguous sequence of characters from a string. For example, “cde” is a substring of the string “abcdefg”. We say that substring *s1* is duplicated in string *s* if *s1* shows us in *s* at least two times, without overlapping. For example, if s == “a**bcd**ef**bcd**gh” then there is a duplicated substring of length 3: “bcd”. There is no duplicated substring of length 4 and higher.

a) Write an algorithm in pseudo-code (in file h2.doc) in a function **find_dup_str(*s, n*)** that determines whether a string *s* contains a duplicated substring of a given length *n* and that returns the **first occurring substring of length *n* that is duplicated** (if any) or the empty string, otherwise. For the example above (s == “a**bcd**ef**bcd**gh”) with substring length 3, the substring to be returned is “bcd“. With length 2, the algorithm returns “bc”, and not “cd”. For this value of parameter *s*, if the length parameter is 4 or more, the algorithm returns “”.
Write an algorithm, not a Python program.
To get full credit make sure the algorithm has ALL the features required, as explained in the class material.

b) Create a new file p2.py where you write a Python function **find_dup_str(s, n)** that implements the algorithm from part a). Parameter *s* is the original string and *n* is the length of the substring to look for. Your implementation must use string slicing.
In file p2.py write after the function definition some code that reads a string and a number from the terminal, and then calls *find_dup_str*(*s, n*) and prints the result. Use this for testing.

c) Write in file h2.doc after part b) an algorithm **find_max_dup(s)** that takes a string *s* as parameter and that determines the **longest substring that is duplicated** in *s*. For instance, *find_max_dup*(“abcdefbcdgh”), the function should return “bcd”. If the string *s* has no duplicated substrings, then the algorithm should return the empty string, e.g. *find_max_dup*(“0123456”) should return “”.
This algorithm MUST use the function written for part b), i.e. must make calls to *find_dup_str*(…) and use its result.
To get full credit make sure the algorithm has ALL the features required, as explained in the class material.

d) Write in file p2.py a function *find_max_dup*(*s*) that implements the algorithm from part c).
In file p2.py, write after the function definition code that reads a string from the terminal in variable *s* and then calls *find_max_dup*(s) and prints the result.

Problem 3. Function Visualization

Write a program **p3.py** that reads in a string *fun_str* from the terminal a mathematical function definition that uses symbol *x* as a parameter, a domain (float $[x_{min},x_{max}]$ interval), and the number of samples *ns* (an integer), and then:

1. computes a list *xs* with *fs* sample points evenly dividing the $[x_{min},x_{max}]$ interval

2. computes a list *ys* with the results of applying the function definition in string *fun_str*, with *x* traversing the elements from list *xs*
3. displays a nice table with the *xs* and *ys* values, as shown in the figure below, using the ***format*** string method
4. displays the figure with the chart of the function defined in *fun_str* using domain samples in list *xs*, the corresponding range values in list *ys*, and the functions from the pylab module.

IMPORTANT: This problem is very similar to a part of problem 2 from Homework 1. This time, however, instead of using a hardcoded function *y(x)=a*x**2+b*x+c*, we will use a function whose expression is given by the interactive user. The user can rely on the functions in the math package. For instance, the user can enter the following expression for the *fun_str* variable:

*2 * math.sin(2*math.pi * x)*

The program reads string fun_str:

```
fun_str = input("Enter function with variable x: ")
```

The program calls the Python *eval*() function to evaluate the function expression for the current value of x:

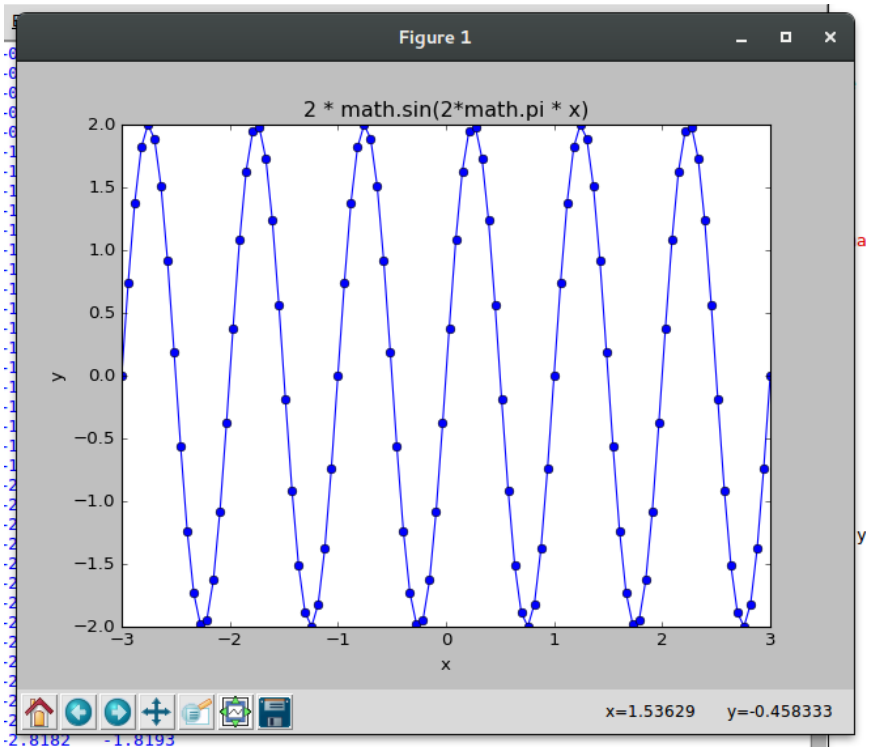
```
for x in xs:
      y = eval(fun_str)
      ys.append(y)
```

The *eval*() function takes a string containing valid Python code and interprets and evaluates within the environment of the current program, right where *eval* was called. Check out more about eval at http://stackoverflow.com/questions/9383740/what-does-pythons-eval-do

Here is a sample user session, with input highlighted in yellow:

```
Enter function with variable x: 2 * math.sin(2*math.pi * x)↵
Enter number of samples: 100↵
Enter xmin: -3↵
Enter xmax: +3↵
         x          y
--------------------
   -3.0000   +0.0000
   -2.9394   +0.7433
   -2.8788   +1.3802
   -2.8182   +1.8193
   -2.7576   +1.9977
   -2.6970   +1.8900
   -2.6364   +1.5115
   -2.5758   +0.9165
   -2.5152   +0.1901
   -2.4545   -0.5635
   -2.3939   -1.2363
   -2.3333   -1.7321
…… (rest of the table not shown)
```

The resulting function chart is shown next:



Use the pylab.xlabel(), pylab.ylabel(), and pylab.title() functions to set the x label, y label, and the figure title, respectively.

Instructions.

1. Write a heading "**Problem 3**" in file **h2.doc**.

2. After that, write the **p3.py** program. Test it a few times.

4. Copy-paste source file p3.c to the **h2.doc** file, with syntax highlighting if possible.

5. Take one or two screenshots of the *p3* program **after it completed**, showing the interactive session in IDLE or Spyder with the number table AND the function figure.
Insert/paste the screenshot(s) into the **h2.doc** file.

Submission Instructions

Convert the **h2.doc** file to PDF format (file **h2.pdf**) and upload the following files on Canvas by clicking on the Homework 1 link:

1. h2.pdf
2. p1.py
3. p2.py
4. p3.py

Grading:

Problem 1: 35%

- algorithm and code correctness: 30%
- programming style: 5%

Problem 2: 35%

- algorithm and code correctness: 30%
- programming style: 5%

Problem 3: 30%

- code correctness: 20%
- programming style: 5%

- screenshot(s) showing the program running correctly: 5%

IMPORTANT NOTES:

- A submission that does not follow the instructions 100% (i.e. perfectly) will not get full credit.
- Only submissions uploaded before the deadline will be graded.
- You have unlimited attempts to upload this assignment, but only the last one will be graded.