CSE 101 - Introduction to Programming
Assignment 1
January 2022

GENERAL INSTRUCTIONS

1. In this assignment, you need to write the code for every sub-problem in different files.
2. You are only allowed to use the basic modules available in Python. You cannot use any external modules, libraries or APIs.
3. Please create a document citing all the online and offline resources that you used in doing this assignment.
4. Your code will be checked for plagiarism against the code of your classmates as well as some sample codes available online.
5. You can solve **any 6 problems out of 8**. Out of these 6 problems, **one of either Q6 or Q7 is mandatory to attempt.** Rest all the other problems are bonus and practice questions.
6. Bonus problems will be graded only if you score more than 80% in the regular 6 problems.
7. As it is your first intensive assignment, hence start early. Resolve all your doubts with the TAs, *2 days* before the deadline.

SUBMISSION INSTRUCTIONS

1. Your code files must be renamed as A1_2021xxx_{Problem_no}.py. For example, A1_2021001_1.py, would be the python file for problem 1 for roll no. 2021001. For Bonus Problems refer to them as problem no. B1 and B2. So, the naming convention would be A1_2021xxx_B1.py for bonus problem 1.
2. Your document with references to resources used must be in pdf format and named A1_Resources_2021xxx.pdf
3. All these files (code files and the reference pdf file) must be put in a zipped folder named A1_2021xxx.zip
4. The zip file containing everything relevant to this assignment, must be submitted before the deadline on the google classroom.

PROBLEMS

Q1.

Print following patterns *(exactly as depicted)*: You must write a generic code so that you can change n to get different size patterns, but the pattern should not change. A menu-driven program should be made for each of these patterns. The program should take the type of pattern as input *(the strings in bold below)* first, followed by the value of n.

1. **"Right-angled triangle"** with n = 5

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2. **"Isosceles triangle"** with n = 4, n is always even.

```
   1
  123
 12345
1234567
```

3. **"Kite"** with n = 4 , n is always even.

```
   1
  123
 12345
1234567
 12345
  123
   1
```

4. **"Half Kite"** with n = 5

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

5. **"X"** with n = 4 , n is always even

```
1234567
 12345
  123
   1
  123
 12345
1234567
```

Q2.

You are a Maths Instructor and you have a class of ‘n’ students. One by one, a student comes to you and provides a geometric shape: 2D or 3D shape. For either of those scenarios, the students also pick a shape or a figure: for example, if a student picks 2D shape, they can pick one out of the 2D shapes as mentioned below, similar for 3D shapes. For all these shapes, a specific numbered code is assigned, use this for user input later.

2D shapes: square, rectangle, rhombus, parallelogram, circle
3D shapes: cube, cuboid, right circular cone, hemisphere, sphere, solid cylinder, hollow cylinder

1. Square input: side s
2. Rectangle input: length l, breadth b
3. Rhombus input: side a, diagonals d1,d2
4. Parallelogram input: length l, breadth b, height h
5. Circle input: radius r
6. Cube input: side s
7. Cuboid input: length l, breadth b, height h
8. Right circular cone: slant height l, radius r, height h
9. Hemisphere input: radius r
10. Sphere input: radius r
11. Solid cylinder input: radius r, height h
12. Hollow cylinder input: Radii R1, R2, height h

Input ‘n’ as number of students. For every student, take Geometry Dimension as input: 2D or 3D. For either of these choices, print a Menu List showcasing all geometric figures of that dimension and take shape code as user input (i.e. from 1 to 12, *based on the serial number above, For eg., 1 -> square, 2-> rectangle. etc.*)

If the input shape is 2D, print:
1. Perimeter
2. Area

If the input shape is 3D, print:
1. CSA
2. TSA
3. Volume

Reference for formulas for 2D shapes:
https://www.thoughtco.com/perimeter-and-surface-area-formulas-604147

Reference for formulas for 3D shapes: https://www.toppr.com/guides/maths/surface-areas-and-volumes/area-and-volume-of-combination-of-solids/

Sample Menu for reference (You are free to make your own menu)

```
-------------MENU-------------
1. Square
2. Rectangle
3. Rhombus
4. Parallelogram
5. Circle input
6. Cube input
7. Cuboid input
8. Right circular cone
9. Hemisphere input
10. Sphere input
11. Solid cylinder
12. Hollow cylinder
13. Exit
1
You chose square
Enter the side s of the square:4
The Perimeter of the square is: 16
The Area of the square is: 16
-------------MENU-------------
1. Square
2. Rectangle
3. Rhombus
4. Parallelogram
5. Circle input
6. Cube input
7. Cuboid input
8. Right circular cone
9. Hemisphere input
10. Sphere input
11. Solid cylinder
12. Hollow cylinder
13. Exit
2
You chose rectangle
Enter the length of the rectangle:3
Enter the breadth of the rectangle:4
The Perimeter of the rectangle is: 14
The Area of the square is: 12
-------------MENU-------------
1. Square
2. Rectangle
3. Rhombus
4. Parallelogram
5. Circle input
6. Cube input
7. Cuboid input
8. Right circular cone
9. Hemisphere input
10. Sphere input
11. Solid cylinder
12. Hollow cylinder
13. Exit
13
```

Q3. Plot some functions

For this problem you have to create a **polynomial function**. This function [which has to be taken as input from the user], calculates and returns value depending on the mathematical function that you have defined inside them.
In the main program, you need to accept the user's input for

- Degree of the polynomial
- Coefficients (starting from the leading coefficient)
- Lower bound for X
- Upper bound for X
- Steps in which you would vary X

For example, if I want to plot F1: The degree is two, thus the number of coefficients will be three. Then I will create a specific function for it. And, the range of X in this example is from 0 to 6(0 has been taken as the input for Lower bound on X and 6 as the input for the Upper bound on X). Also the increment step, d in which x has to be varied is 1. Also whatever value is returned by the function implementing F1, it has to be typecasted to the nearest integer value(eg. If it is 5.6, then F1 should return 6, if it is 4.3 then F1 should return 4). For the value returned from F1, let's say that it is **v,** then I will plot **'*', v** many times.

Note: Degree of polynomial function will be at max 3 and it will contain only one variable i.e. x.

```
Provide Polynomial Function details:
Enter the degree of the polynomial: 2
coeff 1: 1
coeff 2: -6
coeff 3: 9
Provide lower bound for x: 0
Provide lower bound for x: 6
Provide the steps in which you wish to increment x: 1

 -------------------------
|*********
|****
|*
|
|*
|****
|*********
```

<u>Q4.</u> Satisfiability Problem

For a given boolean formula **Fn** consisting of **only** three boolean variables b1, b2, b3 (i.e. the formula will be over these boolean variables and use **bool** operations), determine if the formula is satisfiable or not.

A formula **Fn** is called *satisfiable* if the variables of the formula **Fn** (b1, b2, b3) can be replaced by the values TRUE/FALSE in such a way that the formula **Fn** becomes TRUE.

If it is satisfiable, give any set of values for the variables that satisfies **Fn**.

Note: Please use the variables names as: Fn, b1, b2, b3.
The boolean expression **Fn** will be hard-coded. Thus, you'll use:
Fn = (b1 and not b1)
Fn = (b1 or b2) and (b2 or not b3)
to check if the expression evaluates to True or False.
Examples:

Fn = (b1 and not b1)

b1	**not b1**	**Fn**
T	F	F
F	T	F

Thus, this expression is unsatisfiable. (Fn never becomes TRUE)
Output:
> Unsatisfiable

Fn = (b1 or b2) and (b2 or not b3)

b1	**b2**	**b3**	**not b3**	**b1 or b2**	**b2 or not b3**	**Fn**
T	T	T	F	T	T	T
T	T	F	T	T	T	T
T	F	T	F	T	F	F
T	F	F	T	T	T	T
F	T	T	F	T	T	T
F	T	F	T	T	T	T
F	F	T	F	F	F	F
F	F	F	T	F	T	F

Thus, Fn is satisfiable. (Fn becomes TRUE)
One such combination for the boolean variables is TRUE, TRUE, TRUE.
Output:
> Satisfiable
> True, True, True

Hint: Nested loops will do the trick!

Q5.
Take input a non negative number **'n'** and create functions for each of the following :

i) a function **getReverse(n)** which will reverse the digits of n and return it.

ii) a function **checkPalindrome(n)** which will return True if n is a palindrome, else False.

iii) a function **checkNarcissistic(n)** which will return True if n is a ***Narcissistic*** number otherwise, False. (Reference : https://mathworld.wolfram.com/NarcissisticNumber.html)

iv) a function **findDigitSum(n)** which will find the sum of digits of n. If the given number, S, is a one digit number return S. [n >=0]

v) a function **findSquareDigitSum(n)** which will find the sum of squares of digits of n. Let say S. If S is a one digit number then, return S^2 . [n >=0]

Create a menu driven program to execute these operations. The program should take the operation as input first, followed by the value of the integer n.

Menu-driven means you have to print the menu and then ask from user to select an operation out of the given menu. The menu for this question is like this :

Hello User, Welcome to the Application. Please select one of the following operations.
1. Find reverse of a number
2. Check whether a number is a palindrome or not.
3. Check whether a number is a Narcissistic number or not.
4. Find the sum of digits of a number
5. Find the sum of squares of digits of a number.
6. Select 6 to exit the application.

General flow. You can deviate slightly not much from it. ->
1) First display the menu and ask the user to select an operation.
2) then ask the user to input n, call the required function and then print the result you got from the function call.
3) Then again go to step 1 and repeat the process until the user chooses option 6.

A sample case is provided below :

1) getReverse(12345) must return 54321. n will not contain a trailing zero here.
2) a) checkPalindrome(12321) must return True as number '12321' is a palindrome
b). checkPalindrome(1231) must return False.
3) checkNarcissistic(1634) must return True as 1634 is a Narcissistic number.
4) a) findDigitSum(1071) must return 9 as the sum of digits are 1 + 0 + 7 + 1 = 9 which is a single digit number
b) findDigitSum(89) , the sum is 8 + 9 = 17 which is a 2 digit, so again call findDigitSum (17) which is 1 + 7 = 8. We stop here as 8 is a one digit number. So the total sum your function must return is 17 + 8 = 25.
5) a) findSquareDigitSum(12) . Sum of squares of digit = 1^2 + 2^2 = 1 + 4 = 5, we stop and return 5.
b) findSquareDigitSum(89). Sum of squares of digit = 8^2 + 9^2 = 64 + 81 = 145.
again, call findSquareDigitSum(145). Sum of squares of digit = 1 + 16 + 25 = 42.
again, call findSquareDigitSum(42). Sum of squares of digit = 16 + 4 = 20.
again, call findSquareDigitSum(20). Sum of squares of digit = 4 + 0 = 4. We stop as 4 is a one digit number.
So the final result you function must return is : 145 + 42 + 20 + 4 = 211

Q6.

Consider a polynomial function
The coefficients of all the variables is 1, but their exponents vary.
You need to create a function **find_roots(l)** that takes as parameter a list consisting of the exponents l: [a, b, c, ...]. This will be a finite list. The function **find_roots(l)** will return a root of the function f(x).

To find the roots, you have to use the Newton-Raphson method. You can have a look at it here - https://en.wikipedia.org/wiki/Newton%27s_method

Note: Here, a,b,c … R

You will notice that the Newton-Raphson method requires . But, we don't have it as an input so you can work step-by-step and first try to compute and then try to apply Newton-Raphson method. You can also check the differentiability of a function through this and not jump to the Newton-Raphson method as the Newton-Raphson can only be applied to differentiable functions. If a non-differentiable function is entered, you can simply exit the code.

Example 1:
c = [2, -1] # so this means
Using Newton-Raphson method starting with x_0 = 1 and having a threshold of 0.001
We get the root as -1.0

Example 2:
c = [2, 1, 1, 1, 0.8] # so this means

Using Newton-Raphson method starting with $x_0 = 1$ and having a threshold of 0.001
We get the root as 0

Q7.

Consider a polynomial function
The coefficients of all the variables is 1, but their exponents vary.
Create a function “calculate_area” that takes as parameter a list consisting of the exponents **“l”**: [p, q, r, ...] (a finite list) as well as 2 inputs **“a”** and **“b”** to define a range of values, and **“d”** as a common difference for sum terms. The function calculate_area**(l,a,b,d)** will return the area under the curve from r1 to r2 limits as described by the polynomial definition of f(x) using a list of exponents l.

Note: Here, a,b,c,d … R

Use **Simpson's 1/3 Algorithm** to find the area under a curve, using the **exact mathematical expression** as shown below:

$$\int_a^b f(x)dx = \int_a^{a+d} f(x)dx + \int_{a+d}^{a+2d} f(x)dx + ... \int_{b-d}^{b} f(x)$$

where each individual term on the RHS can be approximated as:

$$\int_a^{a+d} f(x)dx = \frac{((a+d)-a)}{6}\left[f(a) + 4f\left(\frac{(a+(a+d))}{2}\right) + f(a+d)\right]$$

Note: To have integral summation terms on the RHS, (b-a) needs to be divisible by d. Hence, exit the code if d is not divisible by (b-a)**,** else continue with the procedure.

Example 1:
c = [2, 1] # so this means
a=0
b=6
d=2
Using Simpson’s ⅓ Algo, RHS= 90

Explanation

Putting the values, of x = 0, 1, 2 in f(x) we get,

So,

Similarly, do the same step for the other 2 terms,

,

Finally,

Example 2:

c=[0.5, -1] # so this means

a=1

b=3

d=1

Using Simpson's ⅓ Algo, RHS= 3.897335456369697

Q8. Optimization problem

You purchase a car. Every year the car price depreciates, and its maintenance cost increases. However, you derive value/utility from using it. Let's say the limit of using the car is 15 years.

The goal is to find the best time to sell your car within the 15 year limit. i.e. the time when the cost of owning becomes less than the value you derive.
If throughout the 15 year period, the value of your car is the same or more than the cost of owning, then you sell it after 15 years.

Variables:
Initial cost: **c**
Depreciation rate: **r%** every year
Maintenance cost: **1%** of **c** per year for the first 5 years, then it increases by **50%** every year
Value/Utility per km is Rs **50 per km** (cost of taxi plus comfort and convenience) and this increases by **10%** every year.
We assume you drive a fixed distance every year - say **6000 km**

cost of owning = cost of car after applying depreciation rate in that particular year

-

maintenance cost

Output:
Print the year which is the best time to sell your car.

Sample Test case:

Initial cost: 1,000,000
Depreciation rate: 5% every year
[initial value of rest of the variables are given above]

Output: 6 (you sell the car after 6 years)

Explanation:

1st year:
maintenance: 1% of initial_cost = 10000
depreciation: 5% of initial_cost = 50000
cost = 1000000 - 50000 - 10000 = 940000

value per km (10% increase) = 55
value = 6000*55 = 330000

2nd year:
maintenance: old_maintenance + 1% of initial_cost = 20000
depreciation: 5% of initial_cost = 50000
cost = 940000 - 50000 - 20000 = 870000

value per km (10% increase) = 60.5
value = 6000*60.5 = 363000

3rd year:
maintenance: old_maintenance + 1% of initial_cost = 30000
depreciation: 5% of initial_cost = 50000
cost = 870000 - 50000 - 30000 = 790000

value per km (10% increase) = 66.55
value = 6000*66.55 = 399300

4th year:
maintenance: old_maintenance + 1% of initial_cost = 40000
depreciation: 5% of initial_cost = 50000
cost = 790000 - 50000 - 40000 = 700000

value per km (10% increase) = 73.205
value = 6000*73.205 = 439230

5th year:
maintenance: old_maintenance + 1% of initial_cost = 50000
depreciation: 5% of initial_cost = 50000
cost = 700000 - 50000 - 50000 = 600000

value per km (10% increase) = 80.5255
value = 6000*80.5255 = 483153

6th year:

maintenance: 50% increase of itself = 1.5*50000 = 75000
depreciation: 5% of initial_cost = 50000
cost = 600000 - 50000 - 75000 = 475000

value per km (10% increase) = 88.57805
value = 6000*88.57805 = 531468.3

Now,
Cost of owning became less than the utility I derive. (475000 < 531468.3)
Thus, I will sell my car in the 6th year.

BONUS PROBLEMS

B1.

Abhay has recently opened a gift store. The store, since it is new, consists of only ***3 items*** for now. Every item must have a **price per pack**(*taken as input from the user*). To popularize his store, which he has named as ***"Delhi Days"***, he offers ***saver combo*** packs of **two items**. Each of these combo packs has some associated *discount* with them(*taken as input from the user*). Ultimately, there is one ***'SuperSaver'*** pack consisting of all the three items with a **28%** discount on the combined price of those 3 items.

- You have to take the required inputs based upon the output shown below.
- The 1st Saver Combo called ComboPack1 consists of Item1 and Item2.
- 2+nd Saver Combo called ComboPack2 consists of Item1 and Item3
- 3rd Saver Combo called ComboPack3 consists of item2 and Item3

- SuperSaver pack consists of all the items
- All the discounts apply on the sum of total prices for each of the items present in that pack.

Create a catalog of items and the prices as shown in the image below. You can use your own ideas. You are not required to strictly follow the pattern. You can add your ideas and intricacies within the code.

User Inputs required:

You have to take prices of each item, discounts for each combo pack and contact number as input from the user. You can see the prices of respective combo packs have been calculated automatically.

Desired Output:

```
Price of Item1: 1000
Price of Item2: 2000
Price of Item3: 3000
<------------------------------------------------------------

------------------------------------------------------------>
Discount details

Provide discount(in percentage) for the 1st saver combo: 10
Provide discount(in percentage) for the 2nd saver combo: 15
Provide discount(in percentage) for the 3rd saver combo: 20
<------------------------------------------------------------

Provide your 10 digit contact number: 9876543210
The resulting catalogue is:
------------------------------------------------------------
............................................................
Delhi Days
R-Block, Model Town 3
Delhi: 110009
------------------------------------------------------------
............................................................
Item       Price(per Pack)
|          |
Item1      1000
Item2      2000
Item3      3000
ComboPack1 2700.0
ComboPack1 3400.0
ComboPack1 4000.0
SuperSaver 4320.0
------------------------------------------------------------
............................................................

For home delivery, contact here: 9876543210
```

B2. Ray-Sphere Intersection

You might have heard of the term ray tracing. If not, it is one of the most popular topics in computer graphics. It is what allows many games to have beautiful visuals. It is a method of graphics rendering that simulates the physical behaviour of light. Well, we are not going to code a ray tracer in this course. But, we can go and code the fundamentals of a ray tracer. Fundamentally, ray tracing works on calculating ray-object intersection, and locating the hit point and producing the correct colour for the corresponding pixel. The sphere is always the best geometrical shape to start with as it is one of the simplest shapes to describe mathematically.

In this question, we will try to find at how many points does a given ray of light intersect a given sphere.

The origin of light ray is described as **e**

The direction of light ray is described as **d**
So, the ray can be written as p(t) = e + td, where t is the step we take in the direction of the light
For instance, a point 2 step in the direction of light would be p(2) = e + 2d

The origin of the sphere is described as ($\mathbf{x_0}$, $\mathbf{y_0}$, $\mathbf{z_0}$) and the radius as **R**
So, the equation of sphere would be

The variables: e, d, x_0 , y_0 , z_0, R will be inputted.

d will be provided as an input and it would be in the form of a vector < d_x , d_y, d_z >, and thus, d_x , d_y, d_z will be provided as an input.

Also, we know that a point ($\mathbf{x_1}$, $\mathbf{y_1}$, $\mathbf{z_1}$) is said:

1. On the sphere, iff
2. Inside the sphere iff
3. Outside the sphere iff

Given this information you need to find out if the ray of light intersects the sphere or not. And if it does, print the point(s) of intersection. For point(s) of intersection you can print the point(s) where you found out that the ray intersects the sphere and no need to separately find the point(s) of intersection.

Constraint: t ≤ 1000

Sample Case
For,
e = <0,0,0>
d = <0, 0.8, 0.6>
center = <0.1, 4, 3>
R = 3

We find the first intersection at t=3,
And for the points of intersection, here you can print -
The ray with origin at <0,0,0> and direction <0, 0.8, 0.6> intersects the circle with radius 3 and center at <0.1, 4, 3> first time somewhere between: point <0, 1.6, 1.2> at t=2 and point <0, 2.4, 1.8> at t=3

We find the second intersection at t=8,
And for the points of intersection, here you can print -
The ray with origin at <0,0,0> and direction <0, 0.8, 0.6> intersects the circle with radius 3 and center at <0.1, 4, 3> second time somewhere between: point <0, 5.6, 4.2> at t=7 and point <0, 6.4, 4.8> at t=8

PRACTICE PROBLEMS

Q1:

For these sub-problems you can either write an individual program for each one of them or, you can use a menu driven approach so as to select a problem number and then solve it. You are free to apply your ideas. Multiple solutions possible.

1. Find if a number is a prime number or not.
2. Find the sum of all digits of a number. Eg. 49, sum of digits is 4+9 = 13. Eg. 2. 12345, sum of digits is 1+2+3+4+5 = 15.
3. Write a function to print fibonacci series upto n terms using loop.
4. *Implement collatz conjecture*

   A conjecture is a statement that is yet to be proved/disproved. Collaz said that given a number **X,**

   - if it is even, then **X** is replaced by **X/2.**
   - If it is odd, then **X** is replaced by **3*X+1**

   The above process is repeated till 1 is encountered.
   You have to find out, in how many steps the process will reach to 1.

Q2:

In IP lab 2, Sam wanted your help in playing computer games, now he wants your help in his math assignment. Since you are the master of programming in his school, we expect you to write a python code for the following problem given by Sam :

Given 3 integers(possibly negative) x, y, z, find whether they can be converted to AP after multiplying exactly one integer out of 3 by an integer k(possibly negative). Note that you can not change the order of given integers. After multiplying x, y and z must be integers. If yes, find which integers we should multiply and value of k.

Sample 1 : x=3,y=5,z=6, answer is NO.
Sample 2 : x = -6, y = 2, z = -2, answer is YES, y is multiplied by -2.
Now, x= -6,y= -4,z = -2 . x, y, z forms an AP.

Q3:

Write a function playStr(), which takes two strings **X** and **Y** as parameters. Also you have to take these two strings as an **input** and **call the playStr()** function for them and then **print** the value which is returned by the function.
Note: X and Y consist of **English alphabets** only.

Do the following task in the function ***playStr(X, Y)***:

1. **Print** whether X has **all the vowels** (a, e, i, o, u) or not **# True or False**
   (if string X has ‘a’ or ‘A’ or both then we say ‘a’ is in X)
2. **Print** whether Y is a substring of X. **# True or False**
3. **Return** the count of consonants in Y.
   (if Y = ‘Apple’ then you have to return 2 i.e. p, l)

Sample 1:

X = ‘Apple’, Y = ‘App’

Output:

False
True
1

Explanation:

False, as Apple contains only 1 vowel
True, as App is a substring of Apple
1 , as App contains only one consonant i.e. p

Q4:

Mohan and Suresh are two close friends. One day they decided to explore the truth behind the special theory of relativity. With Mohan being the stationary observer on earth, Suresh, with his state of the art and scientifically approved spacecraft, selects a point beyond the solar system which he has to reach. The speed of his spacecraft is around 98% of the speed of light. After Suresh has returned back to the earth, exactly 2 days (24 hours) have passed since. For the clarification he asks Mohan, how much time has elapsed, according to his clock. To Suresh’s surprise the times being shown by the clock are different.

So how much time (in hours) has elapsed according to Mohan’s clock? [Hint: Lorentz factor, , where is the speed of light in m/s, is the speed of Suresh’s spacecraft in m/s.

The time from the perspective of an stationary observer, is given as

where, is the time from the perspective of the observer in motion]

Constraints:

[speed of spacecraft cannot reach or exceed the speed of light]

Hints:

- Compute Mohan’s version of time = (time by suresh’s clock)/
- Return the value in hours
- Handle the constraint as well, so that the universal law is not violated.

For this problem, You are free to use any number of inputs and create different types of command line interaction with the user. It’s all about your own ingenuity and creativity.

You can try implementing the “BABYLONIAN METHOD” to compute the square root of a number and use it for this problem.