Recursion :-

Recusive function :-

A function that calls itself within its own definition to solve a problem or perform a task.

```
def recurse():
    ...
    recurse()
    ....
    recurse()
```

calling itself

Base case :- The base case is the condition that specifies when the function should stop calling itself and return a result directly. It acts as the stopping criterion for the recursion, preventing infinite recursion.

Recursive case :- The recursive case is the part of the function that calls itself with a modified version of the input, leading to smaller instances of the same problem.

Ex:- fibonacci, factorial numbers.

Lambda function :-

Is a small, one-line function that can have any number of arguments but can only have one expression.

Syntax :-

lambda arguments : expression.

Ex: Addition of Two numbers :

```
add = lambda x,y : x+y
result = add (3,5)
print (result)
```

o/p : 8

Squaring a Number :

```
Square = lambda x : x**2
result = square (4)
print (result)
```

o/p : 16

Problems

Finding the sum of Two Numbers.

I/P : num1 = 5
num2 = 10

o/P : Sum : 15

Topics : Input & output, variables, Arithmetic operators.

Code :

```
def add2Numbers (c,d):
    print (c+d)
a = int (input())
b = int (input())
add2Numbers (a,b)
```

o/p :

```
a = 5
b = 10
sum = 15
```

4) Swap the values of two variables without using a temporary variable :

I/P : a = 10
b = 20

O/P : After swapping
a = 20
b = 10

Code:
```
def swap (a,b):
    b = b+a
    a = b-a
Print (f"value of a is : {a}")
Print (f"value of b is : {b}")
Swap (5,10)
swap (10,5)
swap (20,30)
```

O/P :-
20
10

Lists :

Definition :

- A list is a versatile and mutable data structure used to store a collection of items.
- Defined using square brackets []
- contain elements of different data types.
- Allow duplicates.

Creating Lists :

you enclose the elements inside square brackets, separated by commas

```
empty_list = []
numbers = [1,2,3,4,5]
fruits = ["apple", "banana", "cherry"]
mixed_list = [1, "hello", 3.14, True]
```

Accessing Elements :

You can access individual elements of a list using their index. Indexing in Python starts from 0.

~~truth~~

```
fruits = ["apple", "banana", "cherry"]
print (fruits [0])
print (fruits [1])
```

o/p : apple
banana

Python also supports negative indexing, where -1 refers to the last element.

Slicing Lists : You can extract a portion of a list using slicing. slicing allows you to create a new list with a subset of elements.

```
number = [1, 2, 3, 4, 5]
print (numbers [1:4])
```

o/p : [2, 3, 4]

Modifying Elements : You can modify individual elements in a list by accessing them using their index and then assigning a new value.

```
fruits = ["apple", "banana", "cherry"]
fruits [0] = "orange"
print (fruits)
```

o/p : ["orange", "banana", "cherry"]

Methods :

append () : Adds an element to the end of the list

```
l = [1, 2, 3, 4]
l.append (5)
print (l)
```

o/p : [1, 2, 3, 4, 5]