ques 1.What is the difference between a function and a method in Python?
A function is a collection of lines of code that accomplishes a certain task. Functions have:
*Name
*Parameters
*Return statement
Return statement and parameters are optional. A function can either have them or not.
syntax of function-
```
          def function_name(parameters)
```

Functions inside a class are called methods. Methods are associated with a class/object.
syntax of method-
```
          class class_name:
          def method_name(parameters)
```

Functions in Python	Methods in Python
Functions are outside a class	Methods are created inside a class
Functions are not linked to anything	Methods are linked with the classes they are created in
Functions can be executed just by calling with its name	To execute methods we need to use either an object or class name, a dot operator.
Functions can have zero parameters.	Methods should have a default parameter
Functions can not access or modify class attributes	Methods can access and modify class attributes
Functions are independent of classes	Methods are dependent on classes

```
#example of method
class Addition:
 def add( num1, num2):
    return num1 + num2
print(Addition.add(4,5))
```

```
9
```

```
#example of function
def temp():
    a= int(input("enter a number"))
    b= int(input("enter a number"))
    return a+b
print(temp())
```

```
enter a number3
enter a number4
7
```

ques 2. Explain the concept of function arguments and parameters in Python.
A parameter is the variable defined within the parentheses during function definition. Simply they are written when we declare a function.
parameter are declared once in a code.
```
# Here a,b are the parameters
def sum(a,b):
  print(a+b)
sum(1,2)
```

an argument is a value that is passed to a function when it is called. It might be a variable, value or object passed to a function or method as input. They are written when we are calling the function.The values of arguements can be changed
```
def sum(a,b):
  print(a+b)
# Here the values 1,2 are arguments
sum(1,2)
```
different types of arguements are:
Positional arguments: These are passed to functions in the same order as the parameters are defined.
Keyword arguments: These are passed with the parameter name, allowing them to be in any order.
Default arguments: These have a default value if no argument is provided.
Variable-length arguments: These allow passing an arbitrary number of arguments, using *args for positional and **kwargs for keyword arguments.

MAJOR DIFFERENCE BETWEEN PARAMETER AND ARGUEMENT IS:
Parameters are the variables listed inside the parentheses in the function definition.
Arguments are the values passed to the function when it is called.

ques 3.What are the different ways to define and call a function in Python?
Functions are nothing but a block of code which performs a specific task and called again and again as per requirement
syntax for declaration and calling of function:-
```
def function_name(): #function name is your choice
    print("hello")
function_name()  #function calling
```

```
#example:-
def try_(l1,l2):
    if l1>l2:
        print(l1)
    else:
        print(l2)
try_(45,56)
```

```
56
```

ques 4.. What is the purpose of the `return` statement in a Python function?
*The return statement in Python is used to exit a function and return a value to the caller. It allows you to pass back a specific result or data from a function, enabling you to utilize the output for further computation or processing.
*The return statement can be used with or without a value.
*If a value is provided with the return statement, it represents the function’s output or result.
*Multiple return statements can exist within a function, depending on different conditions or paths of execution.

```
def is_(number):
    if number > 0:
        return True
    else:
        return False
print(is_(10))
print(is_(-5))
```

```
True
False
```

ques 5. What are iterators in Python and how do they differ from iterables?
iterator is an object that is used to iterate over iterable objects lists , tuples , dicts and sets.
The iterator object is initialized using the iter() method. It uses the next() method for iteration.it is an representing stream of data.
iterables:-any python object or sequential structure that is capable of returning its elements one at a time.
obejct that can return iterator.

```
#example of iterator
l=("apple","mango","grapes")
y=iter(l)
print(next(y))
print(next(y))
print(next(y))
```

```
apple
mango
grapes
```

ques 6. Explain the concept of generators in Python and how they are defined.
*generators functions are a way to create iterators in a more concise and memory efficient manner.
*these functions allow you to iterate over a potentially large sequence of data without loading the entire sequence into memory
*generator function uses yield keyword , it returns the value specified in the yield and pauses the execution state.the next time generator function is called it resumes from where it left.
*yield keyword produces a series of values over time one at a time rather than returning a single result.

```
#eg:-
def fib_():
    a,b=0,1
    while True:
        yield a
        a,b=b,a+b
    fibg = fib_()
    for i in range(10):
        print(next(fibg))
```

ques 7. What are the advantages of using generators over regular functions?
1.Execution and Control Flow:
Normal Functions: Execute their code sequentially, complete their operations, and return values before ending execution.
Generators: Can pause and resume execution using yield statements, allowing them to produce values on thefly and maintain their local state between calls.
2.Memory Usage:
Normal Functions: Can consume substantial memory, especially if they involve large data structures, as all variables and data are retained in memory until the function completes.
Generators: Are memory-efficient, yielding values one at a time and minimizing memory consumption
3.Usage Scenarios:
Normal Functions: Ideal for computations and tasks with straightforward execution paths that require immediate results.
Generators: Suited for streaming data, handling large datasets, and infinite sequences, as they allow lazy evaluation and efficient memory usage.

ques 8. What is a lambda function in Python and when is it typically used?
*lambda function is a concise way to create anonymous functions
*these are often used for short term operations and are defined in a single line.
syntax:
lambda arguements : Expression

```
from re import X
#usage of lambda function:
#addition of two nnumbers:
s=lambda x:x**2
s(2)
```

```
4
```

```
#factorial of number
temp=(lambda n:1 if n ==0 else n*temp(n-1))
temp(7)
```

```
5040
```

```
#generate fibonacci series
temp=lambda n:n if n<=1 else temp(n-1)+temp(n-2)
[temp(i) for i in range(10)]
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
#check even numbers
l=lambda x:X%2==0
l(6)
```

```
True
```

```
#sort list by length
m=["mansi","data","course"]
sorted(m,key=lambda x:len(x))
```

```
['data', 'mansi', 'course']
```

ques 9. Explain the purpose and usage of the `map()` function in Python.
map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)\
Syntax : map(fun, iter)
Parameters:
fun: It is a function to which map passes each element of given iterable.
iter: It is iterable which is to be mapped.

```
#example
l=[3,4,5,6]
def sq(x):
    return x**2
list(map(sq,l))
```

```
[9, 16, 25, 36]
```

```
# map with lambda function
l1=[10,30,40]
l2=[45,56,67]
list(map(lambda x,y:x+y,l1,l2))
```

ques 10. What is the difference between `map()`, `reduce()`, and `filter()` functions in Python?
*The map () function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple, etc.).
*The map () function can be used with or without lambda functions.
Syntax: map(fun, iter)
Parameters:
fun: It is a function to which map passes each element of given iterable.
iter: iterable object to be mapped.

*The reduce function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.
*This function is defined in “functools” module.
Syntax: from functools import reduce
reduce(func, iterable)
Parameters:
fun: It is a function to execuate on each element of the iterable objec
iter: It is iterable to be reduced

*The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.
Syntax: filter(function, sequence)
Parameters:
function: function that tests if each element of a sequence is true or not.
sequence: sequence which needs to be filtered, it can be sets, lists, tuples, or containers of any iterators.

```
#example of map
l=[3,4,5,6]
def sq(x):
    return x**2
list(map(sq,l))
```

```
[9, 16, 25, 36]
```

```
#example of reduce
from functools import reduce
l=[2,1,3,4,4,5,6]
reduce(lambda x,y :x+y,l)
```

```
25
```

```
#example of filter
l=[2,1,3,4,4,5,6]
list(filter(lambda x:x%2==0,l))
```

```
[2, 4, 4, 6]
```

```
#ques 11. Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list[47,11,42,13]
l=[47,11,42,13]
reduce(lambda x,y:x+y,l)
```

```
113
```