**ASSIGNMENT**

**1. Difference Between Function and Method**

* **Function**: A standalone block of code that can take arguments and return a result. Defined using the def keyword.
* **Method**: A function that is associated with an object. It is called on an object and can access the object's data. Defined within a class.

**2. Function Arguments and Parameters**

* **Parameters**: Variables listed inside the parentheses in the function definition. They act as placeholders for the values you will pass to the function.
* **Arguments**: The actual values you pass to the function when you call it.

**3. Ways to Define and Call a Function**

* **Define**: Use the def keyword.

def my\_function(param1, param2):

return param1 + param2

* **Call**: Invoke the function by its name and pass the required arguments.

result = my\_function(5, 10)

**4. Purpose of the return Statement**

The return statement is used to exit a function and send a value back to the caller. If no return is specified, the function returns None.

**5. Iterators vs. Iterables**

* **Iterable**: An object that can be iterated over (e.g., lists, tuples). It implements the \_\_iter\_\_() method.
* **Iterator**: An object that implements the iterator protocol, consisting of \_\_iter\_\_() and \_\_next\_\_() methods. It maintains the state during iteration.

**6. Concept of Generators**

Generators are a type of iterable that allow you to iterate over data without storing it all in memory at once. They are defined using a function that contains one or more yield statements.

def my\_generator():

yield 1

yield 2

**7. Advantages of Using Generators**

* **Memory Efficient**: They produce items one at a time and do not require all items to be stored in memory.
* **Lazy Evaluation**: Values are computed only when requested, which can lead to performance improvements.

**8. Lambda Function**

A lambda function is an anonymous function defined with the lambda keyword. It's typically used for small, throwaway functions or when passing functions as arguments.

add = lambda x, y: x + y

**9. Purpose of the map() Function**

map() applies a given function to all items in an iterable (like a list) and returns a map object (an iterator).

result = map(lambda x: x \* 2, [1, 2, 3]) # [2, 4, 6]

**10. Difference Between map(), reduce(), and filter()**

* **map()**: Applies a function to all items in an iterable.
* **filter()**: Filters items in an iterable based on a function that returns True or False.
* **reduce()**: Applies a function cumulatively to the items of an iterable, reducing it to a single value. (Requires from functools import reduce)

**11. Internal Mechanism for Sum Operation Using reduce**

To calculate the sum of the list [47, 11, 42, 13] using reduce, it can be illustrated as follows:

from functools import reduce

# Define the addition function

def add(x, y):

return x + y

# Using reduce to sum the list

result = reduce(add, [47, 11, 42, 13])

output will be processed like this :-

1. Start with the first two numbers: 47 + 11 = 58.
2. Then add the next number: 58 + 42 = 100.
3. Finally, add the last number: 100 + 13 = 113.
4. Thus, the final result is 113.