Theory Questions

Q 1) . What is the difference between a function and a method in Python?

Ans : In Python, both functions and methods are blocks of code that perform specific tasks, but have some difference.

Function: Function is block of code that performs specific task. Function are defined using ‘def’ keyword and used anywhere in the code after they are defind. Functions can called in different modules or class.

```
Ex :  def add(a, b):
            return a + b
      result = add(3, 4)
```

Method: Method is defined in the class.it is operate on the object of the class. Methods are called object or instance of a class. Usually the first parameter of the method is self. Method is bound to the class.

```
Ex: class Calculator:
        def add(self, a, b):
            return a + b
      calc = Calculator()
     result = calc.add(3, 4)
```

Q2) Explain the concept of function arguments and parameters in Python.

Ans: 1) Parameters: Parameters are the variables inside the parenthesis in the function definition. They act as placeholder for the values that passed to function when it is called.

Parameters allow to pass data into the function. So that it can operate on the data

```
Ex:     def greet(name):     # 'name' is a parameter
            print(f"Hello, {name}!")
```

2) Argument: Arguments are the actual data or the values that you passed to function when it is called.

```
Ex:   greet("Ajay")    # 'Ajay' is an argument
```

Q3) What are the different ways to define and call a function in Python?

Ans : Function can be defined and called in various ways.

1) Standard function definition.
   Most common way to define function is using def keyword.

Ex:
```
def greet(name):
    return f"Hello, {name}!"
# Function call
print(greet("Ajay"))  # Output: Hello, Ajay!
```

2) Function with Default Parameter.
   Define functions with default parameter values, which are used if no argument is provided during the function call.

Ex:
```
def greet(name="Guest"):
    return f"Hello, {name}!"
# Function calls
print(greet())       # Output: Hello, Guest!
print(greet("Ajay")) # Output: Hello, Ajay!
```

3) Function with Variable Length Arguments:

Ex:
```
def add(*args):
    return sum(args)
# Function call with multiple arguments
print(add(1, 2, 3, 4))  # Output: 10
```

Q4) What is the purpose of the `return` statement in a Python function?

Ans: Return statement provide the result or the output of the function. When return statement is executed the function terminates and specific values is sent back to the caller.

Ex:
```
def add(a, b):
    return a + b
result = add(3, 5)
print(result)  # Output: 8
```

Q5) What are iterators in Python and how do they differ from iterables?

Ans: 1) Iterable : An Iterable is any python object that can return its element one at a time. Example of iterable is list, tuples, strings, dictionaries.

```
Ex:  my_list = [1, 2, 3]
     for item in my_list:      # 'my_list' is an iterable
         print(item)
```

2) Iterator: Iterator is an object that represent the stream of data. It is an object that keeps the state and produce next value when you call the 'next()'.

```
Ex:  my_list = [1, 2, 3]
     my_iterator = iter(my_list)  # 'my_iterator' is an iterator
     print(next(my_iterator))  # Output: 1
     print(next(my_iterator))  # Output: 2
     print(next(my_iterator))  # Output: 3
     # print(next(my_iterator))  # Raises StopIteration
```

Q6) . Explain the concept of generators in Python and how they are defined.

Ans : Generator is special type of function that uses 'yeild' keyword instead of 'return' to produce series of values.

```
Ex:  def even_numbers(n):
         count = 0
         while count < n:
             yield count * 2
             count += 1
     # Create a generator object
     evens = even_numbers(5)
     # Iterate over the generator and print each value
     for num in evens:
         print(num)
```

Q 7) What are the advantages of using generators over regular functions?

Ans : Memory Efficiency**:** Generators avoid loading entire datasets into memory.

Performance**:** They allow for immediate data processing without waiting for all data.
Simplified Code: Generators retain state and reduce complexity.
Resource Management: They help with managing and cleaning up resources effectively.
Handling Infinite Sequences**:** Generators are ideal for iterating over potentially infinite sequences.

Q8) What is a lambda function in Python and when is it typically used?

Ans: It is defined using 'lambda' keyword. Lambda functions are commonly used in scenarios where small, throwaway functions are needed for a short period of time.

Syntax- lambda arguments: expression

Ex: add = lambda x, y: x + y

print(add(3, 5)) # Output: 8

Q9) Explain the purpose and usage of the `map()` function in Python.

Ans: The `map()` function in Python is used to apply a function to every item in an iterable (like a list or tuple) and return a new iterable (a `map` object) with the results.

The main purpose of `map()` is to transform data. If you have a collection of items and you want to apply the same operation to each item, `map()` helps you do that in a clean and efficient way.

Syntax - map(function, iterable)

Ex: numbers = [1, 2, 3, 4, 5]

squares = map(lambda x: x * x, numbers)

Convert the map object to a list to see the results

print(list(squares))

#output [1, 4, 9, 16, 25]

Q10) What is the difference between `map()`, `reduce()`, and `filter()` functions in Python?

Ans: 1) map() function:

Applies a function to each item in an iterable and returns a new iterable with the transformed items. Use `map()` when you want to transform all elements in a sequence using a specific function.

Ex: numbers = [1, 2, 3, 4, 5]

squares = map(lambda x: x * x, numbers)

print(list(squares)) # Output: [1, 4, 9, 16, 25]

2) filter() function:

Use `filter()` when you want to create a subset of items from an iterable based on a condition.

Ex : numbers = [1, 2, 3, 4, 5, 6]

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers)) # Output: [2, 4, 6]

3) Reduce() function:
Use `reduce()` when you need to combine all items in an iterable into a single value (e.g., summing numbers, multiplying items).
Ex: from functools import reduce
numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product) # Output: 120

Q 11) Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:[47,11,42,13]