FUNCTION

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

- Function: A function is a block of reusable code that is defined using the def keyword and can be called anywhere in the code. Functions are not tied to any specific object and can be standalone.
- Method: A method is a function that is associated with an object and is called on that object. Methods are defined inside a class and typically operate on the object's data.
- Key Difference: Function: Independent and can be called directly. Method: Requires an instance of a class to be called and often works with object attributes.

```
#Function
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))  # Output: Hello, Alice!

#Method
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        return f"Hello, {self.name}!"

person = Person("Alice")
print(person.greet())  # Output: Hello, Alice!
```

Q. Explain the concept of function arguments and parameters in Python.

- In Python, parameters and arguments are related to functions:
  - Parameters: Variables listed inside the parentheses in the function definition. Act as placeholders for the values the function will receive.
  - Arguments: Actual values or data passed to the function when it is called. These values are assigned to the parameters.

Q. What are the different ways to define and call a function in Python?

Ways to Define and Call a Function in Python

1. **Standard Function**: Define using `def` and call by name.
2. **Default Arguments**: Assign default values to parameters and call with or without arguments.

3. **Lambda Function**: Use `lambda` for single-expression functions, called directly or via a variable.
4. **Nested Function**: Define a function inside another function, called through the outer function.
5. **Recursive Function**: A function that calls itself, with a base case to stop recursion.
6. **Variable-Length Arguments**: Use `*args` for positional and `**kwargs` for keyword arguments, called with varying numbers of arguments.
7. **Class Method**: Define functions inside a class and call using an instance of the class.
8. **Higher-Order Functions**: Functions that take other functions as arguments or return them.

Q. What is the purpose of the `return` statement in a Python function?

- The return statement in Python is used to:
  - Send a result back to the caller.
  - Terminate the function's execution.
  - Enable reusability by allowing the returned value to be used elsewhere.
  - If omitted, the function returns None by default.

Q. What are iterators in Python and how do they differ from iterables?

Iterables vs. Iterators

1. **Iterable**:
   - Objects like lists, tuples, or strings that can be looped over.
   - Has `__iter__()` method, which returns an iterator.
2. **Iterator**:
   - An object used to fetch items one at a time.
   - Created by applying `iter()` on an iterable.
   - Has `__iter__()` and `__next__()` methods.

Key Difference:

- **Iterable**: Can be iterated over.
- **Iterator**: Produces items from an iterable and maintains iteration state.

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

Generators in Python

- **Definition**: Generators are iterators that produce values one at a time using the `yield` keyword, instead of storing all values in memory.
- **Memory Efficient**: They generate values on-the-fly.
- **State Preservation**: The function resumes from where it left off after each `yield`.

Key Points:

- Values are produced lazily and can be accessed using `next()` or a `for` loop.
- They raise `StopIteration` when exhausted.

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

Advantages of Generators over Regular Functions

1. **Memory Efficiency**:
    - Generators yield items one at a time, using less memory compared to storing all values in a list.
2. **Lazy Evaluation**:
    - Values are produced only when needed, making generators suitable for large datasets or streams.
3. **State Retention**:
    - Generators remember their state between iterations, eliminating the need to re-compute results.
4. **Improved Performance**:
    - Since they generate values on-demand, they can be faster for iterating over large collections or infinite sequences.

Q. What is a lambda function in Python and when is it typically used?

Lambda Function in Python

- **Definition**: A **lambda function** is an anonymous, small function defined using the `lambda` keyword. It can have any number of arguments but only one expression.
- **When to Use**:
    - For short, throwaway functions where defining a full function using `def` is unnecessary.
    - Commonly used in situations where a function is required temporarily, like in sorting, mapping, or filtering operations.

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

Purpose and Usage of `map()` in Python

- **Purpose**: The `map()` function applies a given function to all items in an iterable (like a list, tuple, etc.) and returns a map object (an iterator) that produces the results.
- **Syntax**:

```
map(function, iterable)
```

- **Usage**:
  - It is used to transform or modify data in an iterable without using explicit loops.

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

Difference between `map()`, `reduce()`, and `filter()`

1. `map()`:
   - Applies a function to every item in an iterable and returns an iterator with the results.
   - **Purpose**: Transformation of data.
   - **Example**: Squaring each number in a list.
2. `reduce()`:
   - Applies a function cumulatively to the items of an iterable, reducing them to a single value.
   - **Purpose**: Aggregation or accumulation of data.
   - **Example**: Calculating the product of all numbers in a list.
3. `filter()`:
   - Filters elements from an iterable based on a function that returns `True` or `False`.
   - **Purpose**: Selective data extraction.
   - **Example**: Filtering even numbers from a list.

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

Sum Operation using `reduce()` on `[47, 11, 42, 13]`

1. Start with the first two elements: `47 + 11 = 58`
2. Apply to the next element: `58 + 42 = 100`

3. Apply to the last element: `100 + 13 = 113`

Final Result: `113`