1. Difference Between a Function and a Method in Python

- **Function**: A function is a block of code that performs a specific task and can be called independently.
- **Method**: A method is a function that is associated with an object and is called on that object.

Example:

```
def greet(name):
return f"Hello, {name}!"
class Person:
    def __init__(self, name):
        self.name = name

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

# Calling function
print(greet("Alice"))

# Calling method
p = Person("Bob")
print(p.greet())
```

Output:

- `Hello, Alice!`
- `Hello, Bob!`

---

2. Function Arguments and Parameters in Python

- **Parameter**: A variable in the function definition.
- **Argument**: The value passed to the function when calling it.

Example:

```
def add(a, b):  # a and b are parameters
return a + b
result = add(3, 5)  # 3 and 5 are arguments
print(result)
```

Output:

- 8

---

3. Ways to Define and Call a Function in Python

- **Standard Function**:

  ```
  def multiply(x, y):
     return x * y
  print(multiply(2, 3))
  ```

- **Lambda Function** (Anonymous function):
  ```
  square = lambda x: x * x
  print(square(4))
  ```

Output:

- 6
- 16

---

4. Purpose of the `return` Statement in a Python Function

The `return` statement terminates the function and optionally passes a value back to the caller.

Example:

```
def subtract(x, y):
return x - y
result = subtract(10, 4)
print(result)
```

Output:

- 6

---

5. Iterators vs Iterables in Python

- **Iterable**: An object capable of returning its members one at a time, permitting it to be iterated over in a for-loop.
- **Iterator**: An object representing a stream of data; it returns the next item in the sequence when `next()` is called.

Example:

```
# Iterable

lst = [1, 2, 3]
for item in lst:
 print(item)

# Iterator

it = iter(lst)
print(next(it))
print(next(it))
```

Output:

- 1
- 2
- 3
- 1
- 2

---

6. Generators in Python

Generators are functions that yield items one at a time using the `yield` keyword, allowing iteration over large datasets efficiently.

Example:

```
def count_up_to(n):
  count = 1
 while count <= n:
    yield count
 count += 1
for number in count_up_to(5):
    print(number)
```

Output:

- 1
- 2
- 3
- 4
- 5

---

7. Advantages of Using Generators Over Regular Functions

- **Memory Efficiency**: Generators yield items one at a time, consuming less memory.
- **Lazy Evaluation**: Values are produced only when needed.
- **Convenience**: Useful for reading large files or streams.

---

8. Lambda Function in Python

A lambda function is a small anonymous function defined with the `lambda` keyword.

Example:

```
double = lambda x: x * 2
print(double(5))
```

Output:

- `10`

---

9. Purpose and Usage of the `map()` Function in Python

The `map()` function applies a given function to all items in an input list (or any iterable).

Example:

```
numbers = [1, 2, 3, 4]
squared = map(lambda x: x ** 2, numbers)
print(list(squared))
```

Output: `[1, 4, 9, 16]`

---

10. Difference Between `map()`, `reduce()`, and `filter()` Functions in Python

- **`map()`**: Applies a function to all items in an iterable.
- **`filter()`**: Filters items in an iterable based on a function that returns a boolean.
- **`reduce()`**: Applies a rolling computation to sequential pairs of values in an iterable.

Examples:

```
from functools import reduce
```

```
# map()
numbers = [1, 2, 3, 4]
squared = map(lambda x: x ** 2, numbers)
print(list(squared))

# filter()
even = filter(lambda x: x % 2 == 0, numbers)
print(list(even))

# reduce()
product = reduce(lambda x, y: x * y, numbers)
print(product)
```

Output:

- `[1, 4, 9, 16]`
- `[2, 4]`
- `24`

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

NOTES

11.)

⇒ from functools import reduce

```
numbers = [47, 11, 32, 48]
result = reduce (lambda x, y: x+y, numbers)
print (result)
```

Output :- 138