1. Difference between a Function and a Method in Python

- **Function**: A block of code that performs a specific task, defined using `def`, and can exist independently.
- **Method**: A function that belongs to an object (invoked using dot `.` operator).

✅ **Example**:

```
# Function
def greet(name):
    return "Hello " + name

print(greet("Ishan"))

# Method (string has built-in methods)
text = "python"
print(text.upper())   # upper() is a method of string objects
```

---

2. Function Arguments and Parameters

- **Parameters**: Variables listed inside the function definition.
- **Arguments**: Actual values passed when calling the function.

✅ **Example**:

```
def add(a, b):   # a, b are parameters
    return a + b

print(add(5, 3))  # 5, 3 are arguments
```

---

3. Different Ways to Define and Call a Function

1. **Normal function**

```
def square(x):
    return x*x
```

```
print(square(4))
```

2. **Lambda function**

```
square = lambda x: x*x
print(square(4))
```

3. **Using `def` with default argument**

```
def greet(name="Guest"):
    return "Hello " + name
print(greet())         # default used
print(greet("Ishan"))  # overridden
```

---

4. Purpose of the `return` Statement

- Exits a function and sends a value back to the caller.

✅ **Example**:

```
def multiply(x, y):
    return x * y

result = multiply(3, 4)
print(result)  # 12
```

---

5. Iterators vs Iterables

- **Iterable**: An object that can return an iterator (`list`, `tuple`, `str`).
- **Iterator**: An object with `__iter__()` and `__next__()` methods.

✅ **Example**:

```
numbers = [1, 2, 3]     # iterable
it = iter(numbers)      # iterator
```

```
print(next(it))  # 1
print(next(it))  # 2
```

---

6. Generators in Python

- Special kind of iterator created using `yield`.
- Produces values lazily (one at a time).

✅ **Example**:

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

for val in countdown(3):
    print(val)
# Output: 3, 2, 1
```

---

7. Advantages of Generators

- **Memory efficient** (don't store all values at once).
- **Lazy evaluation** (generate only when needed).
- Useful for large data streams.

✅ **Example**:

```
def squares(n):
    for i in range(n):
        yield i*i

gen = squares(5)
print(next(gen))  # 0
print(next(gen))  # 1
```

---

8. Lambda Function

- Small anonymous function using `lambda` keyword.
- Often used for short tasks.

✅ **Example**:

```
square = lambda x: x*x
print(square(5))   # 25
```

---

9. Purpose of `map()` Function

- Applies a function to all elements of an iterable.

✅ **Example**:

```
nums = [1, 2, 3, 4]
squares = list(map(lambda x: x*x, nums))
print(squares)  # [1, 4, 9, 16]
```

---

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

- **map(func, iterable)** → applies function to each element.
- **filter(func, iterable)** → keeps only elements where function returns `True`.
- **reduce(func, iterable)** → reduces iterable to a single value (must `import functools`).

✅ **Example**:

```
from functools import reduce

nums = [1, 2, 3, 4]

print(list(map(lambda x: x*2, nums)))   # [2, 4, 6, 8]
print(list(filter(lambda x: x%2==0, nums)))  # [2, 4]
print(reduce(lambda a,b: a+b, nums))   # 10
```

11. Internal Mechanism of `reduce` for Sum ?