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

Function: Function is an independent block of code that performs a task and can be called by its name. Functions are typically independent.
Example:

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

print(greet(Omkar))
```

Method: Method is a functioon associated with an object and it operates on that object's data. So methods are Object-specific.
Example:

```
greet = "hello"
print(greet.upper())
```

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

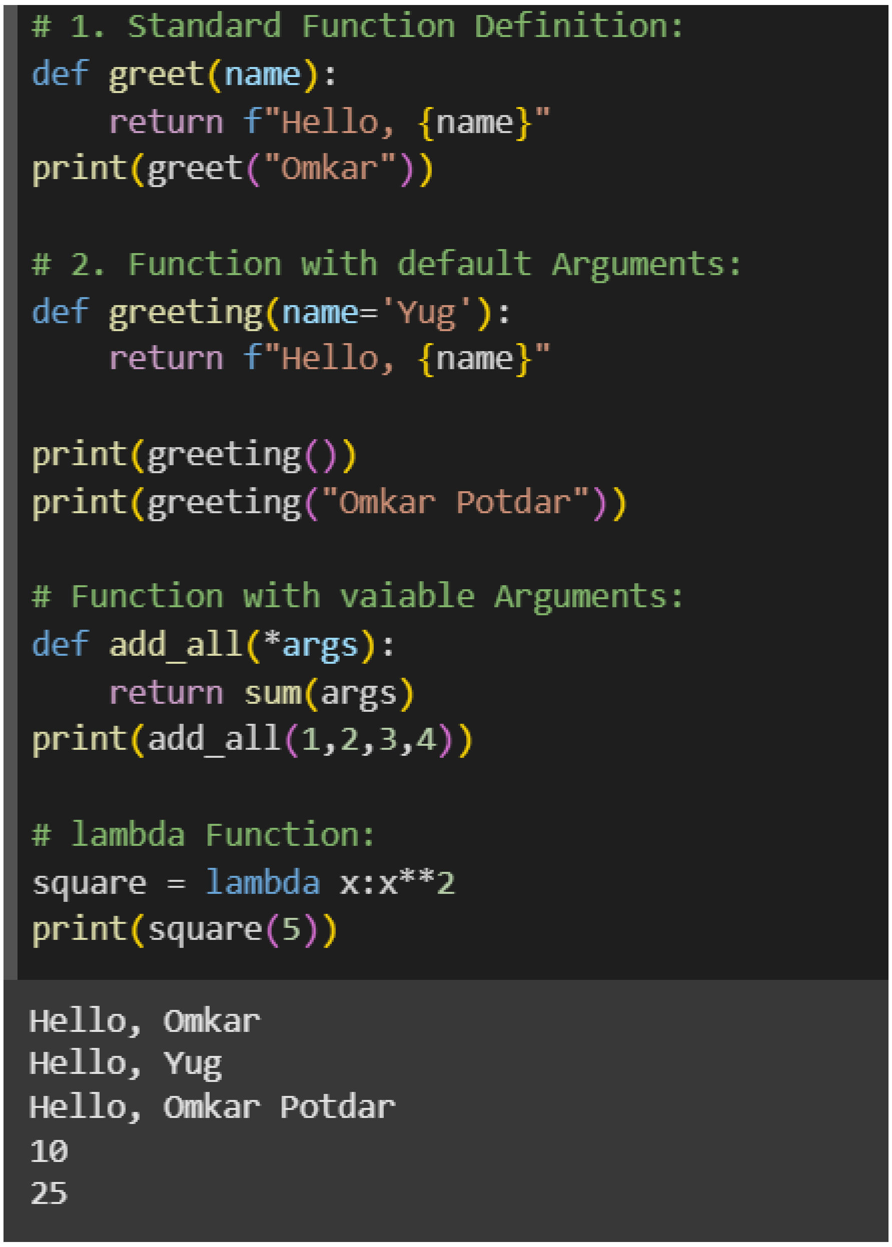
Parameters are the variable names listed in a function's definition.

```
def greet(name):
    print("Hello", name)
```

Arguments are the actual values passed to the function when it is called.

```
greet("Omkar")
```

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



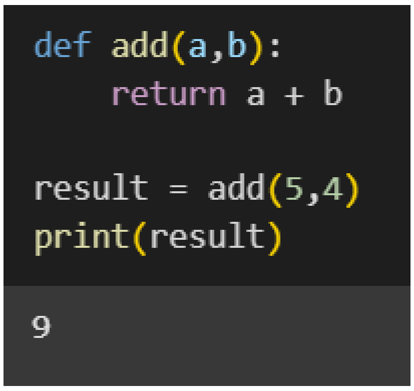
```python
# 1. Standard Function Definition:
def greet(name):
    return f"Hello, {name}"
print(greet("Omkar"))

# 2. Function with default Arguments:
def greeting(name='Yug'):
    return f"Hello, {name}"

print(greeting())
print(greeting("Omkar Potdar"))

# Function with vaiable Arguments:
def add_all(*args):
    return sum(args)
print(add_all(1,2,3,4))

# lambda Function:
square = lambda x:x**2
print(square(5))
```

```
Hello, Omkar
Hello, Yug
Hello, Omkar Potdar
10
25
```

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

The 'return' statement in Python is used to deliver a result. It ends the function's execution.

Example:



```python
def add(a,b):
    return a + b

result = add(5,4)
print(result)
```

```
9
```

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

- **Iterable:** Any python object that can loop over (like - list, tuple, dict, set, str). It implements the '**iter**()' method.

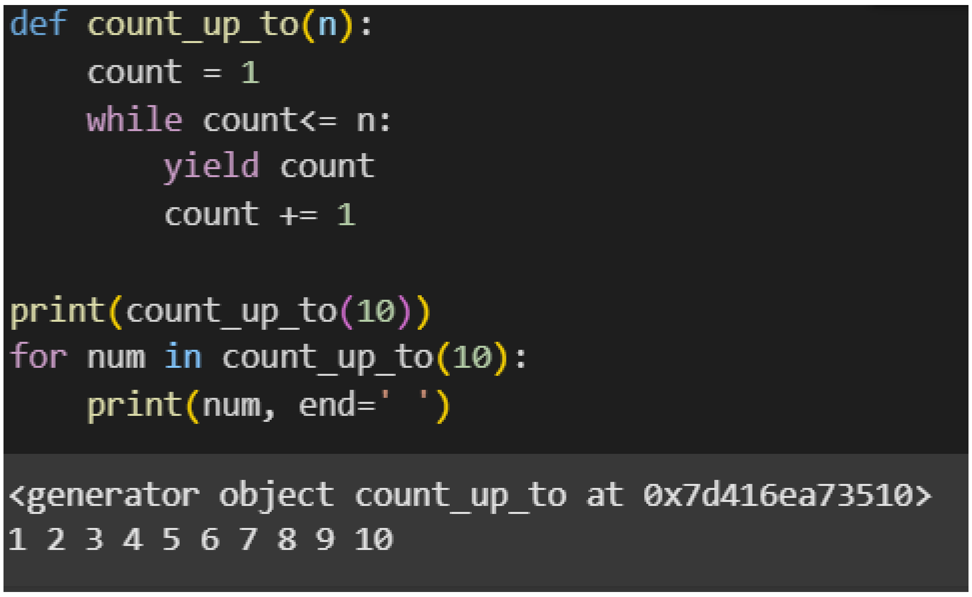
  my list = [1,2,3]

- **Iterator:** An object that produces values **one at a time** using '**next**()' and remembers where it left off. It's created from an iterable using 'iter()'.

  my iter = iter(my list)
  print(next(my iter))
  print(next(my iter))

- ## 6. Explain the concept of generators in Python and how they are defined.
- **Generator:** a special type of iterator in Python that yields values one at a time using the 'yield' keyword instead of 'return'. It's **memory-efficient** way to produce large sequences without storing them all in memory.

  Example:



```python
def count_up_to(n):
    count = 1
    while count<= n:
        yield count
        count += 1

print(count_up_to(10))
for num in count_up_to(10):
    print(num, end=' ')
```

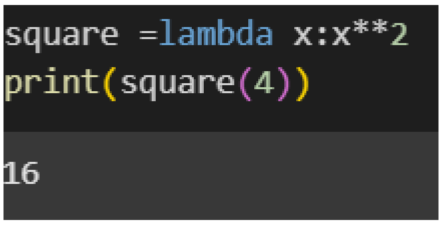
```
<generator object count_up_to at 0x7d416ea73510>
1 2 3 4 5 6 7 8 9 10
```

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

1. **Memory Efficiency :** Generators yield one items at a time instead of storing the entire result in memory.
2. **Improved Performance :** No need to compute all results upfront, which saves time and processing power.
3. **Lazy Evaluation :** Values are generated only when needed, which improves performance.

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

- **lambda Function** is an **anonnymous, one-line** function using 'lambda' keyword.
- It is typically used for **short, simple operations** that don't require a full 'def' block.



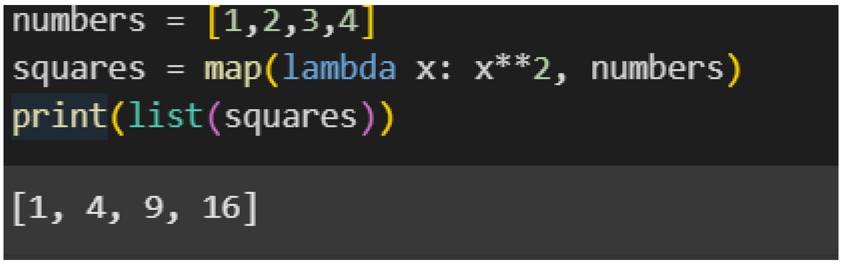
```python
square =lambda x:x**2
print(square(4))
```

```
16
```

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

The 'map()' function is used to **apply a function to every item** in an iterable(like a list or tuple) and return a **new map object** (an iterator) with the transformed values.

map(function, iterable)



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

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

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

1. **map :** Returns a new iterator by **applying the given function to each item** in the input iterable.

   Purpose : Transformation
   Example : map(lambda x:x**2, [1,2,3]) → [1,4,6]

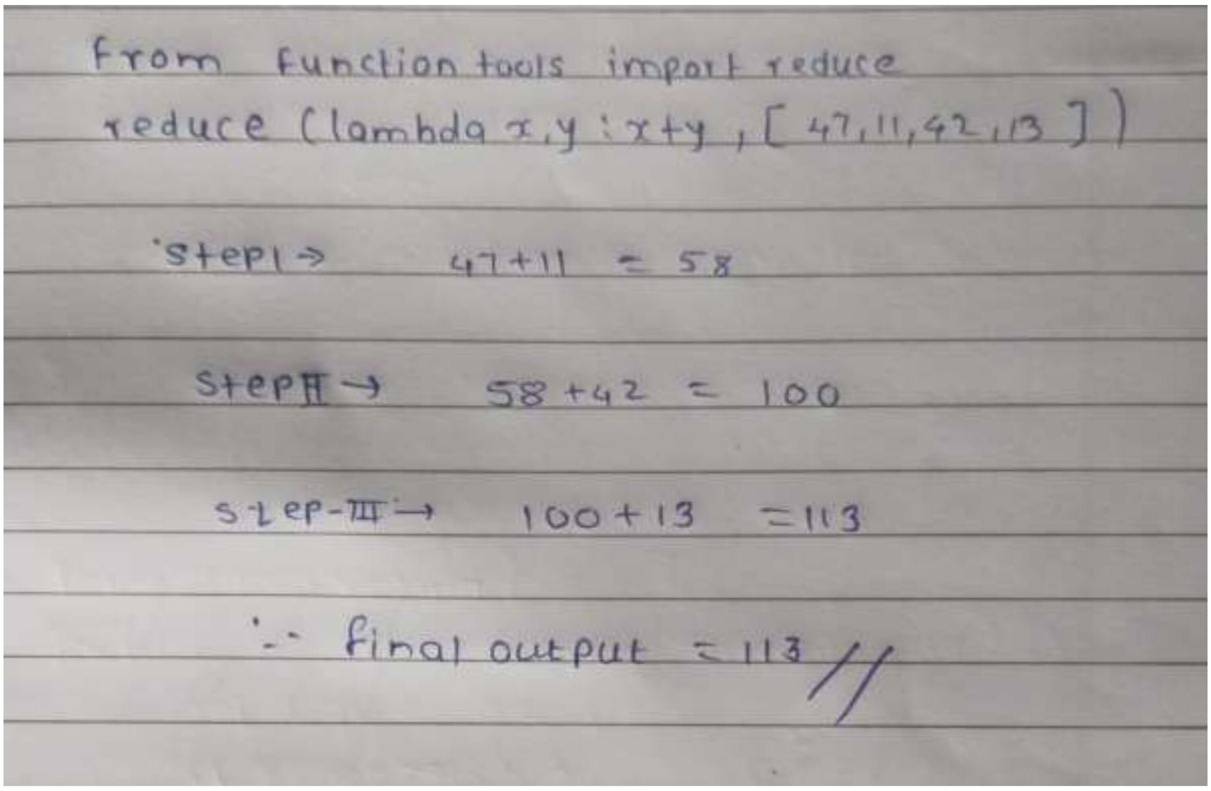
2. **filter :** Returns a new iterator containing **only those items** from the iterable for which the **function returns 'True'**.

   Purpose : Selection
   Example : filter(lambda x: x>2, [1,2,3] → [3])

3. **reduce :** Applies the function **cumulatively** to the items in the iterable, so as to **reduce it to a single value.**

   Purpose : Aggregattion
   Example : reduce(lambda x,y: x+y, [1,2,3] → 6)

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



```
from function tools import reduce
reduce (lambda x,y : x+y , [ 47,11,42,13 ] )
```

step1 → 47+11 = 58

stepII → 58+42 = 100

step-III → 100+13 = 113

∴ final output = 113 //