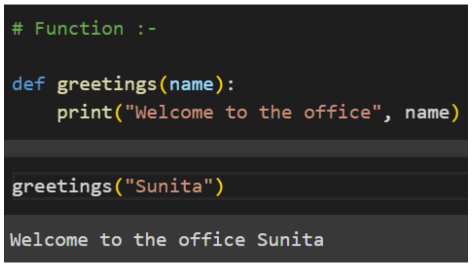
Theoretical Answers

Answer 1 :-

Function in Python :-

A function is a structured, reusable code block utilized for carrying out a specific task. Functions facilitate the creation of modular code, simplifying maintenance and reusability.

- Functions are defined at the global level or within another function or class.
- They are independent of any specific object or instance.
- Functions can be called directly by their name, followed by parentheses `()`, passing necessary arguments.



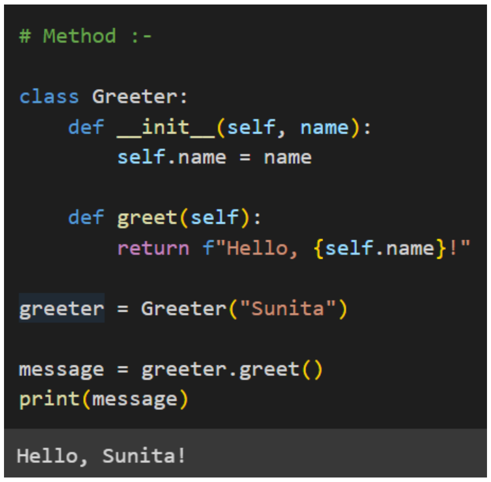
```
# Function :-

def greetings(name):
    print("Welcome to the office", name)

greetings("Sunita")
```

```
Welcome to the office Sunita
```

Method in Python :-

A method in Python is a function that is associated with an object (instance of a class). Methods are defined inside the class definition and are accessed via instances of the class or the class itself.

- Methods are defined within a class using the def keyword.
- They operate on data that belongs to a specific instance (object) of the class.
- They operate on data that belongs to a specific instance (object) of the class.



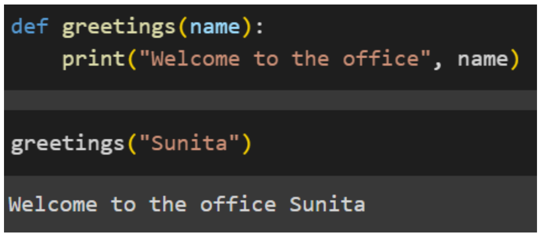
```
# Method :-

class Greeter:
    def __init__(self, name):
        self.name = name

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

greeter = Greeter("Sunita")

message = greeter.greet()
print(message)
```

```
Hello, Sunita!
```

Answer 2 :-

Parameter :- Parameters are variables listed in the function definition. They act as placeholders for values that will be passed to the function when it is called.

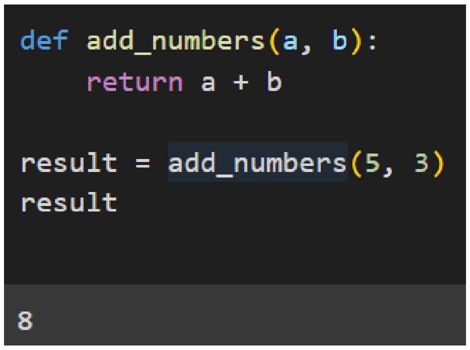
- Parameters are defined in the function header and specify the input the function expects.
- They act as local variables within the function scope.



```
def greetings(name):
    print("Welcome to the office", name)

greetings("Sunita")

Welcome to the office Sunita
```

Argument :- Arguments are the actual values passed to a function when it is called. They match the parameters of the function based on their order or by explicitly naming the parameter.

- Arguments are the actual values or expressions passed to the function during its invocation.
- They can be positional (matched based on order) or keyword-based (explicitly matched by parameter name).
- An argument is the value that is sent to the function when it is called.



```
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 3)
result

8
```

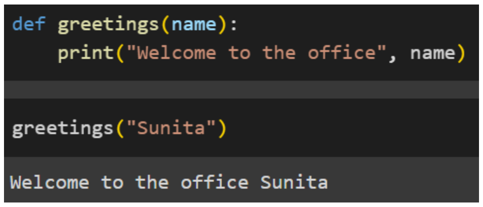
Answer 3 :-

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

A function is a structured, reusable code block utilized for carrying out a specific task. Functions facilitate the creation of modular code, simplifying maintenance and reusability.

There are multiple ways to call a function in Python :-

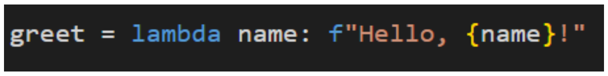
1) **Standard Function Definition**: This is the most common way to define a function using the `def` keyword.



```
def greetings(name):
    print("Welcome to the office", name)

greetings("Sunita")
```

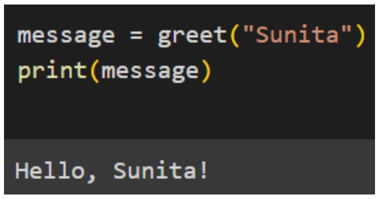
```
Welcome to the office Sunita
```

2) **Lambda Functions**: Lambda functions are anonymous functions defined using the `lambda` keyword, primarily used for short and simple operations.



```
greet = lambda name: f"Hello, {name}!"
```

3) **Direct Call**: Functions are typically called by their name followed by parentheses `()` containing the arguments (if any).

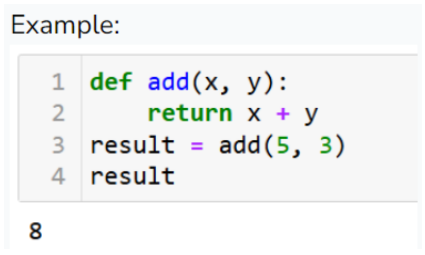


```
message = greet("Sunita")
print(message)
```

```
Hello, Sunita!
```

Answer 4 :-

Return Statement: Determines the value that the function will return. If not included, the function defaults to returning None.



Example:

```
def add(x, y):
    return x + y
result = add(5, 3)
result
```

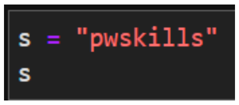
```
8
```

Answer 5 :-

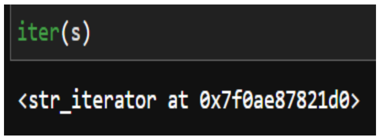
What are iterators in Python and how do they differ from iterables

Iterators :- Iterator is an object representing a string of data and this return the element one by one. It implements the iterator protocol, which requires two methods: __iter__() and __next__().

An iterator is an object that allows traversal through a countable number of values. It is used to iterate over iterable objects like lists, tuples, sets, etc

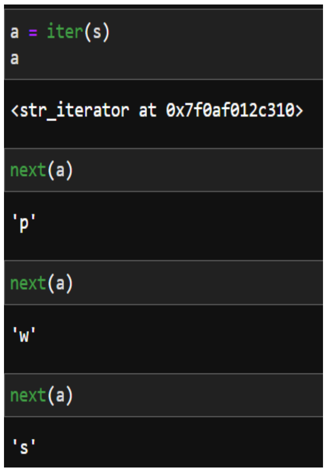


```
s = "pwskills"
s
```



```
iter(s)
```

```
<str_iterator at 0x7f0ae87821d0>
```



```
a = iter(s)
a
```

```
<str_iterator at 0x7f0af012c310>
```

```
next(a)
```

```
'p'
```

```
next(a)
```

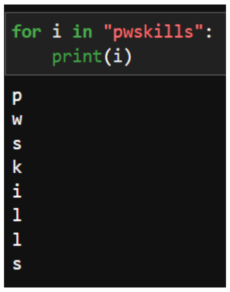
```
'w'
```

```
next(a)
```

```
's'
```

Iterables :- Iterables is any python object/sequential structure/ data structure that is capable of returning it's member one at a time/ one by one.

Iterables which can produce the elements one by one. Permitting it to be iterated over in a for loop/while loop. Example :- List,string



```
for i in "pwskills":
    print(i)
```

```
p
w
s
k
i
l
l
s
```

Answer 6 :-

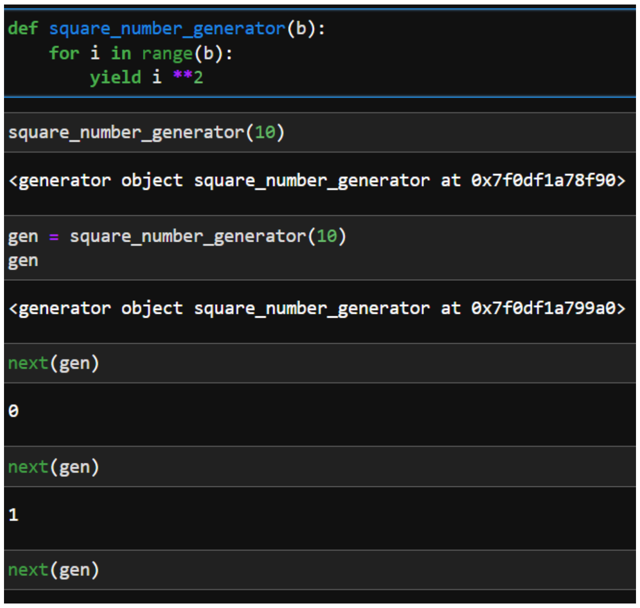
Explain the concept of generators in Python and how they are defined.

Generators in Python :- Generators are a simple way to create iterators using functions and the yield keyword instead of returning values.

General function use return statement but a generator function use ‘yield statement’.

yield is a python keyword.

Yield vs. Return: Unlike the return statement, which exits the function, yield returns a value and pauses the function’s state, allowing it to resume from where it left off.



```
def square_number_generator(b):
    for i in range(b):
        yield i **2
```

```
square_number_generator(10)
```

```
<generator object square_number_generator at 0x7f0df1a78f90>
```

```
gen = square_number_generator(10)
gen
```

```
<generator object square_number_generator at 0x7f0df1a799a0>
```

```
next(gen)
```

```
0
```

```
next(gen)
```

```
1
```

```
next(gen)
```

Answer 7 :-

The advantages of using generators over regular functions :-

Lazy Evaluation:

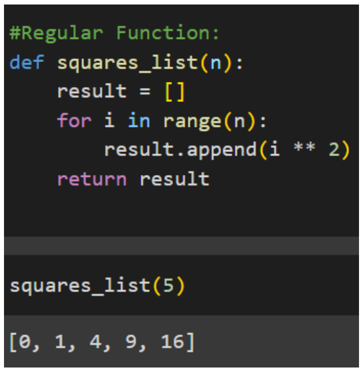
On-Demand Data: Generators produce values one at a time and only when required, avoiding the need to generate or store all values upfront.

Memory Efficiency: Since values are generated as needed and not all at once, generators use minimal memory.

Characteristics: Iterables: Generators return data in an iterable format, allowing the sequence to be iterated over, much like a list or tuple.

Advantages: Efficient Memory Usage: Generators do not store the entire sequence in memory, making them suitable for large datasets or infinite sequences.

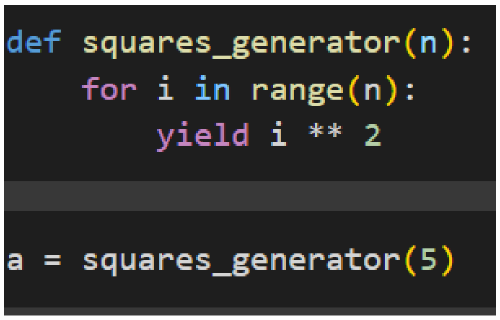
Simpler Syntax: Using yield simplifies the creation of iterators, making the code more concise and readable.



```
#Regular Function:
def squares_list(n):
    result = []
    for i in range(n):
        result.append(i ** 2)
    return result
```

```
squares_list(5)
```

```
[0, 1, 4, 9, 16]
```



```
def squares_generator(n):
    for i in range(n):
        yield i ** 2
```

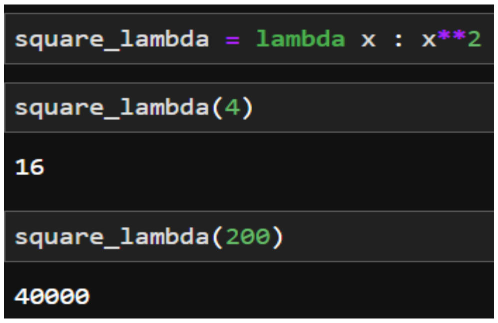
```
a = squares_generator(5)
```

Answer 8 :-

Lambda function :- Lambda functions are small, anonymous functions defined using the lambda keyword. They are used for creating small, throwaway functions without the need to formally define a function using def.

- Syntax >>>>>> lambda argument: expression
- lambda doesn't have any name, that's why it called anonymous function.
- This is also called Anonymous function or short hand function.
- It helps to short the code in one line, that's why it called short hand.
- You don't need to write return statement and def keyword function while you are using lambda function.

This is typically used when you want some value in specific



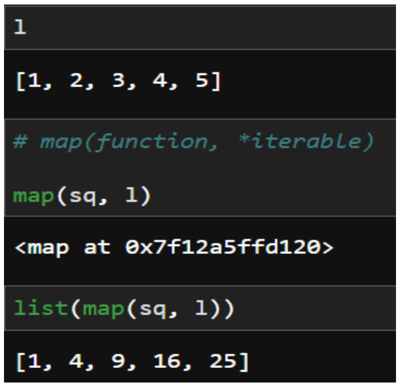
```
square_lambda = lambda x : x**2
square_lambda(4)
16
square_lambda(200)
40000
```

Answer 9 :-

map() function :- The map function applies a given function to all items in an input iterable (like a list) and returns an iterator with the results.

Map in Python is a function that works as an iterator to return a result after applying a function to every item of an iterable (tuple, list, etc.). It is used when you want to apply a single transformation function to all the iterable elements. That iterable and function are passed as arguments to the map in Python.

- Syntax >>>>> map(write Custom function, write iterables)
- Used to transform each item in an iterable by applying the specified function.

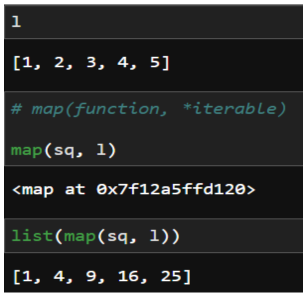


```
l
[1, 2, 3, 4, 5]
# map(function, *iterable)
map(sq, l)
<map at 0x7f12a5ffd120>
list(map(sq, l))
[1, 4, 9, 16, 25]
```

Answer 10 :-

map() function :- The map function applies a given function to all items in an input iterable (like a list) and returns an iterator with the results.

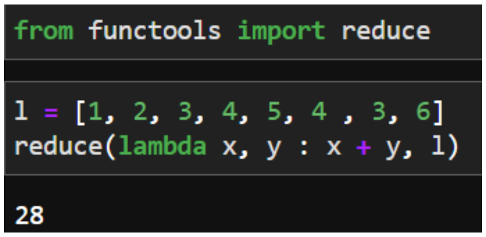
- Syntax >>>>> map(write Custom function, write iterables)
- Used to transform each item in an iterable by applying the specified function.



```
l
[1, 2, 3, 4, 5]
# map(function, *iterable)
map(sq, l)
<map at 0x7f12a5ffd120>
list(map(sq, l))
[1, 4, 9, 16, 25]
```

reduce() function :- The reduce function from the functools module applies a given function cumulatively to the items of a sequence, from left to right, to reduce the sequence to a single value.

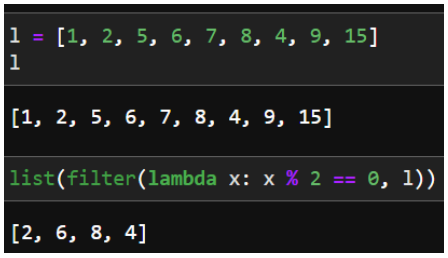
- Syntax >>>>> reduce(function, iterable)
- Reduced always and only take 2 arguments functions.
- It reduces an iterable to a single cumulative value by applying the function cumulatively.



```
from functools import reduce

l = [1, 2, 3, 4, 5, 4 , 3, 6]
reduce(lambda x, y : x + y, l)
28
```

filter() function :- The filter function constructs an iterator from elements of an iterable for which a specified function returns True.

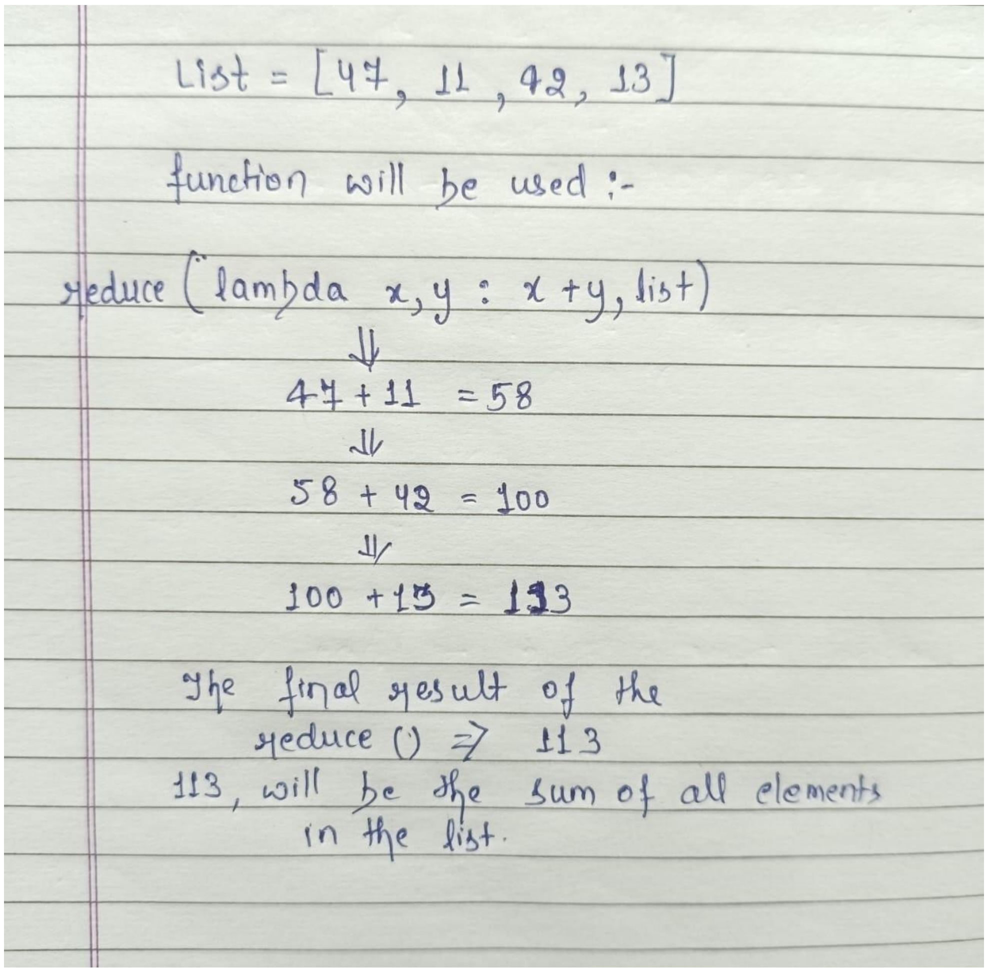
- Syntax >>>>>> filter(function, iterable)
- Used for filtering elements from an iterable based on a condition defined by the function.
- It is used to filter elements from an iterable based on some condition.



```
l = [1, 2, 5, 6, 7, 8, 4, 9, 15]
l
[1, 2, 5, 6, 7, 8, 4, 9, 15]
list(filter(lambda x: x % 2 == 0, l))
[2, 6, 8, 4]
```

Answer 11 :-

The internal mechanism for sum operation using reduce function on this given list : [47,11,42,13] is :-



List = [47, 11, 42, 13]

function will be used :-

reduce (lambda x, y : x + y, list)

$\Downarrow$

47 + 11 = 58

$\Downarrow$

58 + 42 = 100

$\Downarrow$

100 + 13 = 113

The final result of the reduce () $\Rightarrow$ 113

113, will be the sum of all elements in the list.