Programming Refresher

Python Functions

Learning Objectives

By the end of this lesson, you will be able to:

- Create and use functions in Python to improve code organization and reusability
- Identify the advantages of using functions in programming to improve modularity, simplify debugging, and enhance collaboration
- Construct a Python function with different types of arguments to enhance code modularity and flexibility
- Analyze the scope of variables within functions to manage variable accessibility and duration effectively
- Utilize Python generators for efficient data handling to optimize memory usage and performance

Business Scenario

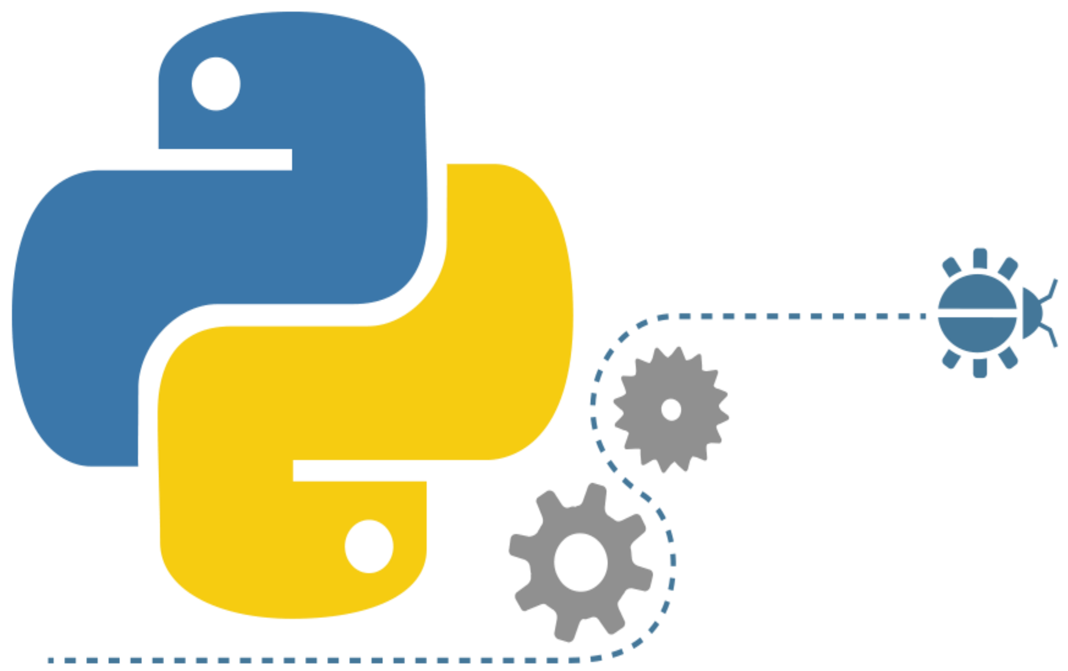
ABC is a fintech company that onboards merchants for its payment gateway. Each new merchant requires a customized backend class, leading to separate codes for each module. This process is inefficient and time-consuming.

By combining repetitive tasks into functions, the same code can be reused across different merchant modules, improving efficiency.

To achieve this, the organization will explore functions, variables, and their scope.

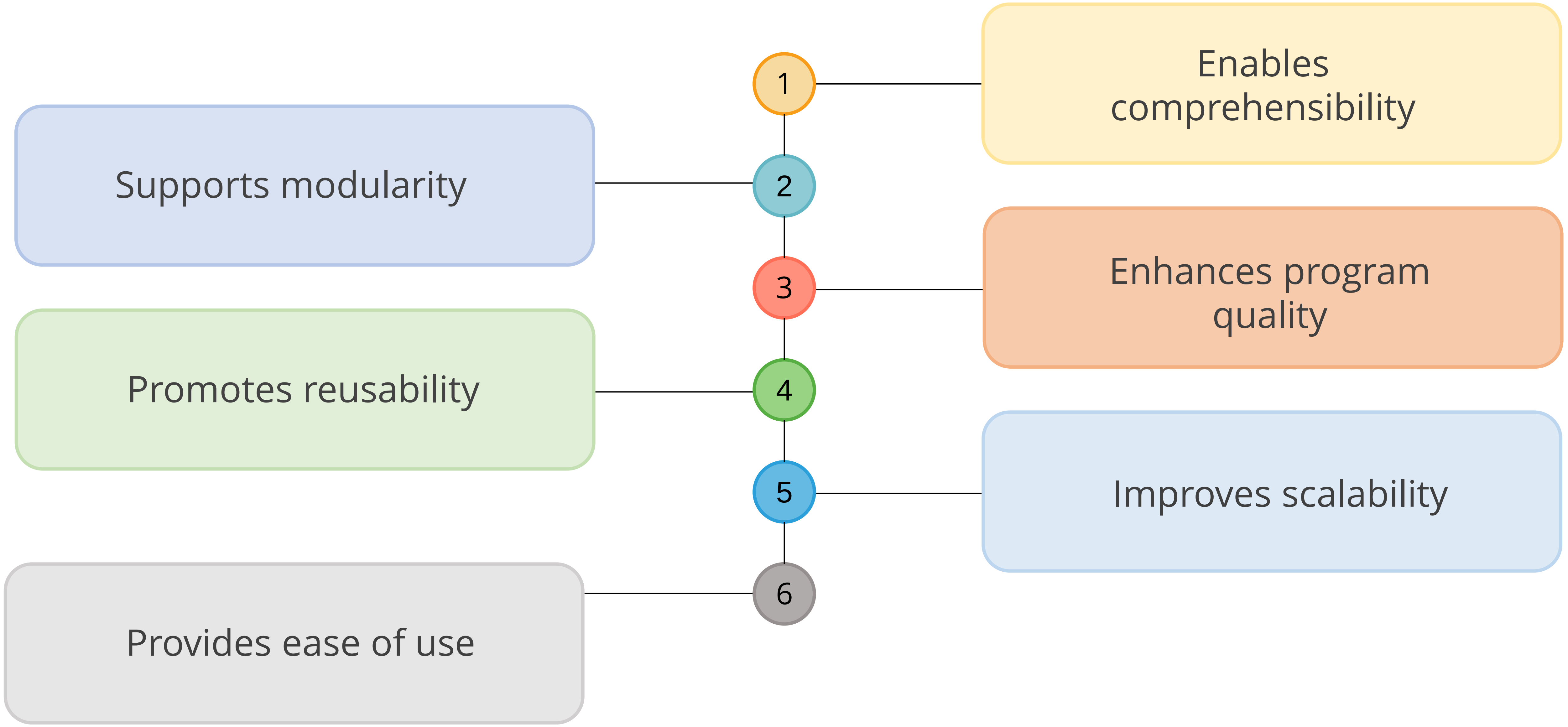
Python Functions and Their Advantages

Introduction to Python Functions



- A function is a collection of related statements that accomplish a specific task.
- It is an organized block of reusable code.
- It executes only when it is called.
- Parameters are data passed into a function.

Functions: Advantages



Functions: Syntax

The basic syntax of a function in Python consists of two parts:

01

Function definition

- A function is defined using the keyword *def,* followed by a function name and parenthesis.
- The argument names passed to the function are mentioned inside this parenthesis.
- The body of the function is an indented block of code.

Syntax

```
def myFunction(arg1, arg2):
    body
```

Functions: Syntax

The basic syntax of a function in Python consists of two parts:

02

Function call

- A function can be called from anywhere in the program.
- When a function is called, program control is transferred to the called function to perform the defined task.
- All the necessary parameters and arguments should be passed during the function call.

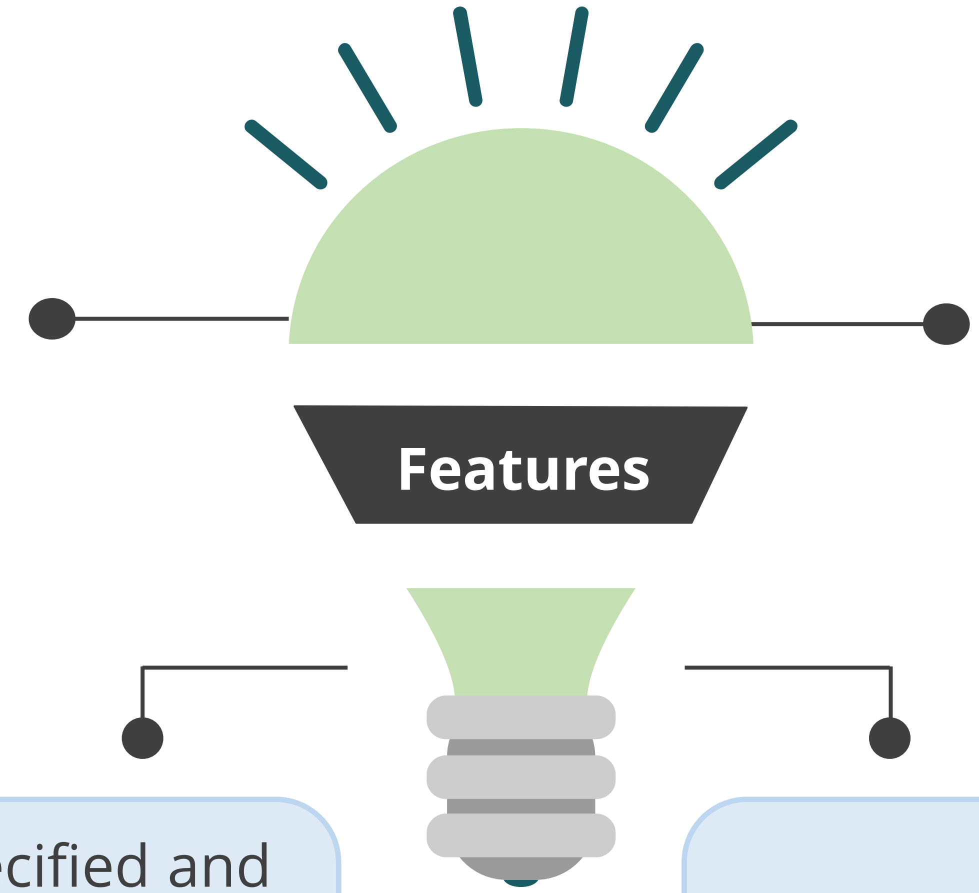
Syntax

```
myFunction(arg1, arg2)
```

Function Arguments

Functions Arguments

Function arguments are values passed to a function as inputs for performing operations. The features of the arguments are as follows:



Information can be passed to a function in the form of arguments.

A function may have zero to multiple arguments separated by commas.

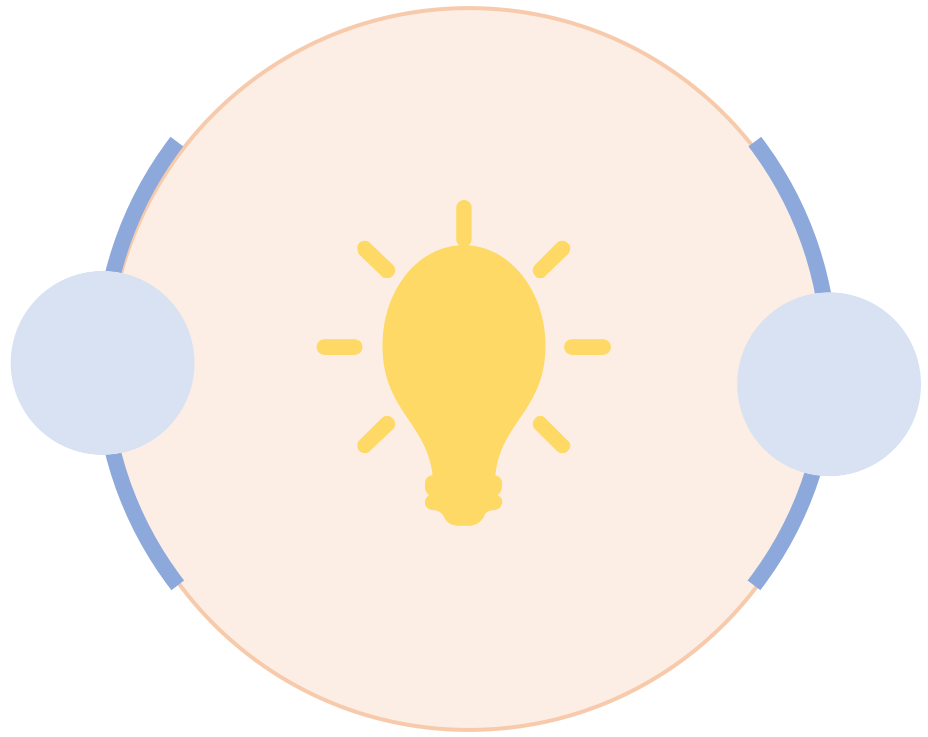
The arguments are specified and enclosed in parentheses in the function definition immediately after the function name.

The arguments can be data values, objects, or functions.

Functions: Argument Matching

Arguments in Python can be matched by position or name.

Positional arguments must be in the same order as in the function definition.



Keyword arguments are referenced by name and may be in any order.

Positional and keyword arguments can be mixed in a function call.

Note

Arguments in a function may also be defined with a default value.

Assisted Practice: Function and Argument Matching

Duration: 5 mins

Objective: In this demonstration, you will learn to create a function with arguments.

Steps to perform:

Step 1: Create a function

Step 2: Pass arguments to the function

Step 3: Perform argument matching

Step 4: Call the function and display the results

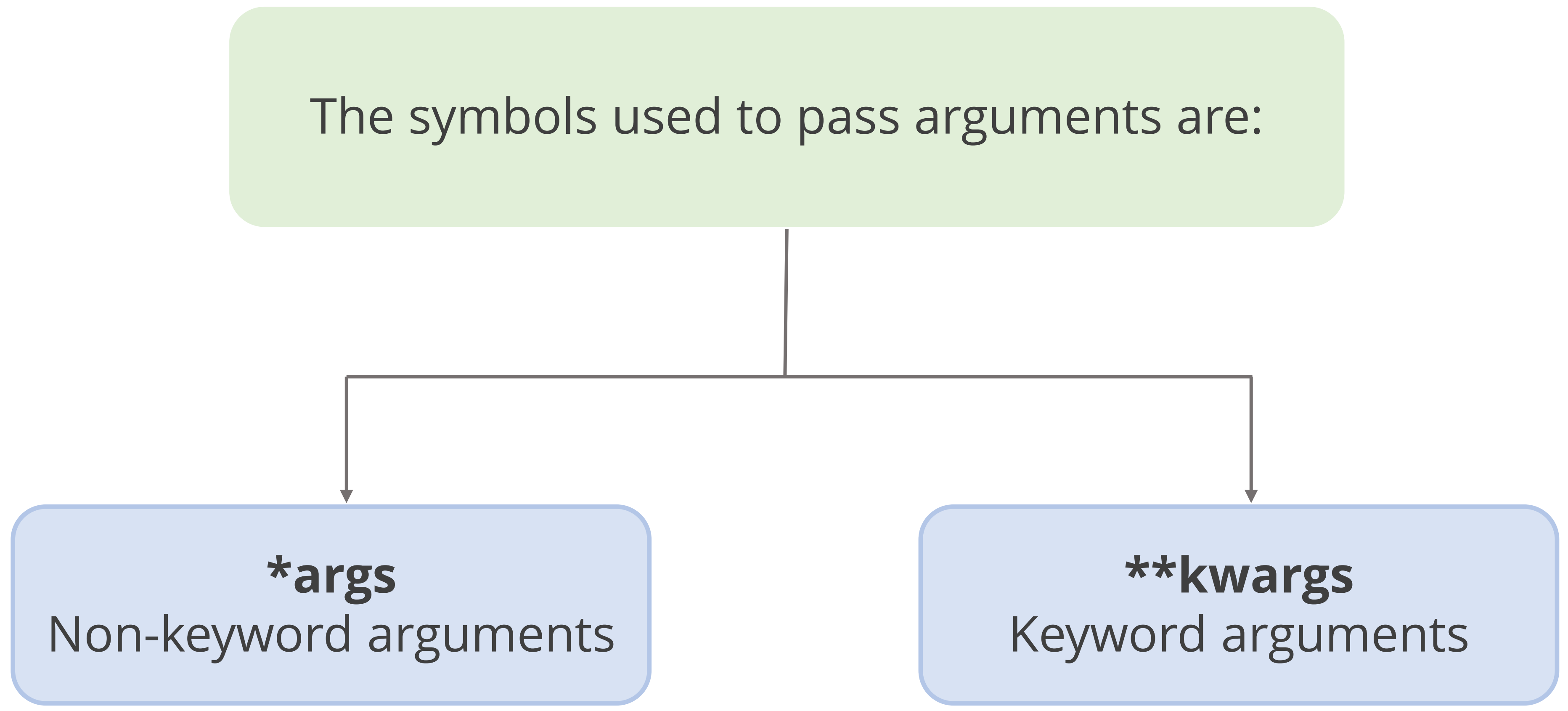
Note

Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

ASSISTED PRACTICE

Functions: Arbitrary Arguments

Python allows passing variables with multiple arguments using special symbols.



The symbols used to pass arguments are:

***args**
Non-keyword arguments

****kwargs**
Keyword arguments

Note

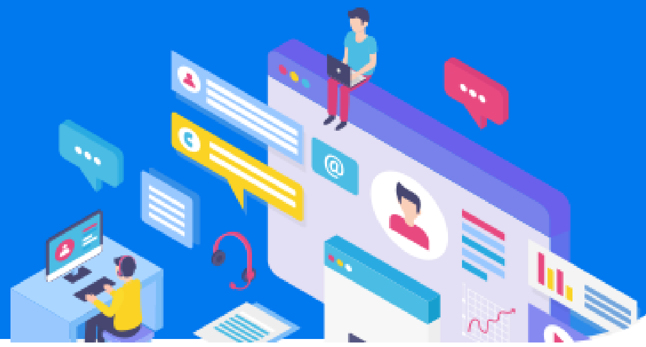
These notations are used when the number of arguments to be passed to the function is uncertain.

Functions: Arbitrary Arguments

The types of arbitrary arguments are explained below:

Non-keyword arguments	**Keyword arguments**
The symbol * captures a variable number of arguments.	Keyword arguments use the ** symbol. Conventionally, **kwargs** defines keyword arguments.
Conventionally, **args** defines non-keyword arguments.	A keyword argument allows you to name the variable before passing it to the function.
he function unpacks these arguments as a list.	Keyword arguments function like a dictionary, mapping each value to its keyword.

Assisted Practice: Arbitrary Arguments



Duration: 5 mins

Objective: In this demonstration, you will learn to use arbitrary arguments.

Steps to perform:

Step 1: Define a function that takes arbitrary arguments

Step 2: Pass arguments to the function

Step 3: Call the function and display the results

Note

Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

ASSISTED PRACTICE

return Statement

Functions: *return* Statement

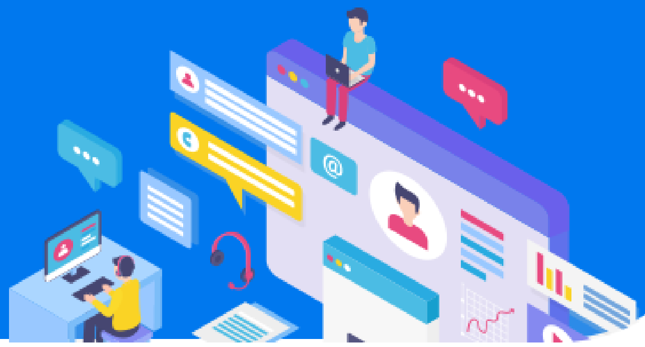
The *return* keyword passes values back to the function call.

If the *return* statement is not used, Python returns *None* by default.

After the *return* statement is executed, no further statements of the function are executed.

A function can return none, multiple values, or data objects.

Assisted Practice: *return* Statement



Duration: 5 mins

Objective: In this demonstration, you will learn to use the *return* statement.

Steps to perform:

Step 1: Define a function with arguments

Step 2: Use the *return* statement to return values when the function is called

Step 3: Call the function and display the results

Note

Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

Scope of a Variable

Function: Scope

Scope in programming defines the environment where the variables can be accessed and referenced.

Features

It defines the functional range of any variable or function.

If a value of a variable is to be retrieved in a function, it searches for the global environment of each package on the search list.

Function: Scope

There are two types of scope:

Global scope	Local scope
Variables and functions defined outside of all classes and functions have global scope.	Variables or functions defined inside a function block, including argument names, are local variables.
It allows variable values to be used or functions to be called from anywhere in the file in the program.	It can be accessed only inside the function block.

Assisted Practice: Scope of Variables

Duration: 5 mins

Objective: In this demonstration, you will learn about the scope of variables.

Steps to perform:

Step 1: Define and demonstrate a global variable

Step 2: Define and demonstrate local variables inside a function

Step 3: Modify global variables inside a function using the global keyword

Note

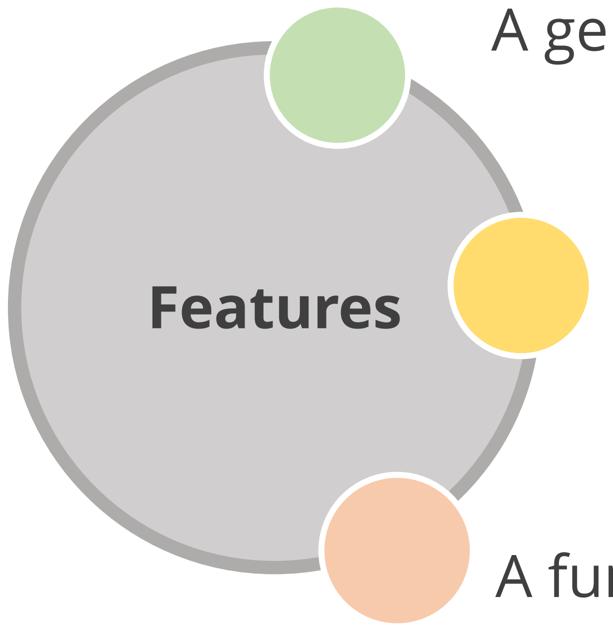
Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

ASSISTED PRACTICE

Generators Function

Function: Generators

Generators are a special function supported in Python that returns an iterator object with a sequence of values.



A generator uses a *yield* statement instead of a return statement.

A generator object can be iterated only once.

A function with a *yield* statement is termed a generator.

return vs. *yield* Statement

The differences between the *return* and *yield* statements are given below:

return statement

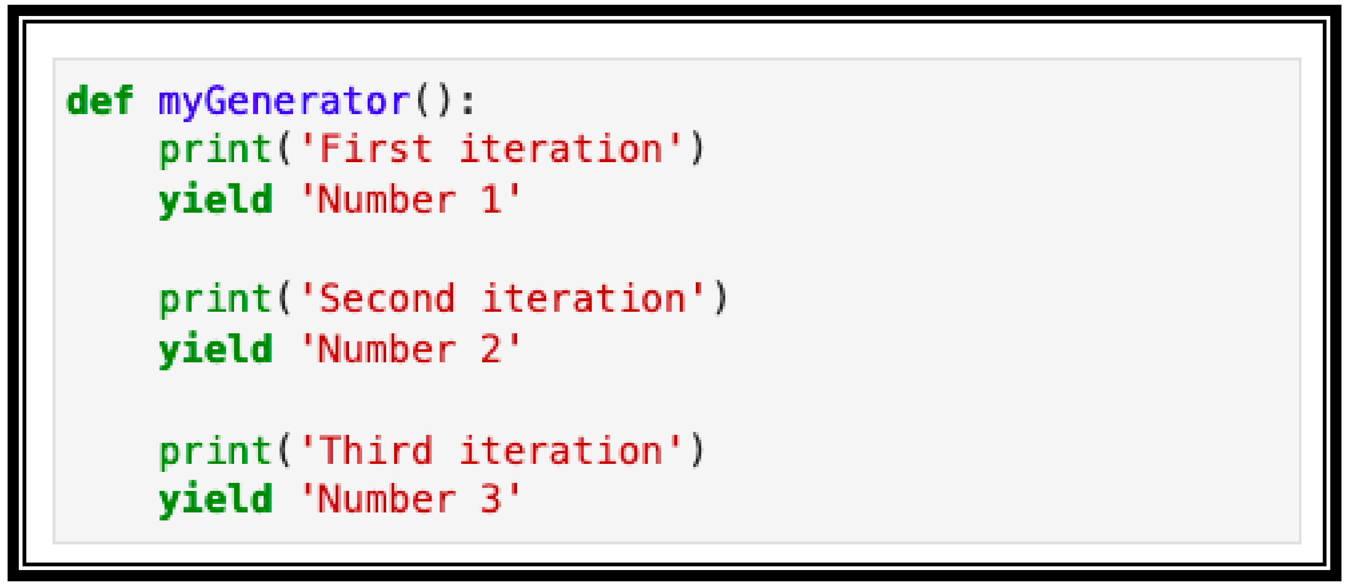
- The *return* statement implies that the function is returning control of execution to the point from where the function is called.
- The process destroys the local variables and values generated.

yield statement

- The *yield* statement implies that the transfer of control is temporary.
- It does not destroy the states of the function's local variables.

Generator: Example

Here a generator function is defined that yields three values:



```
def myGenerator():
    print('First iteration')
    yield 'Number 1'

    print('Second iteration')
    yield 'Number 2'

    print('Third iteration')
    yield 'Number 3'
```

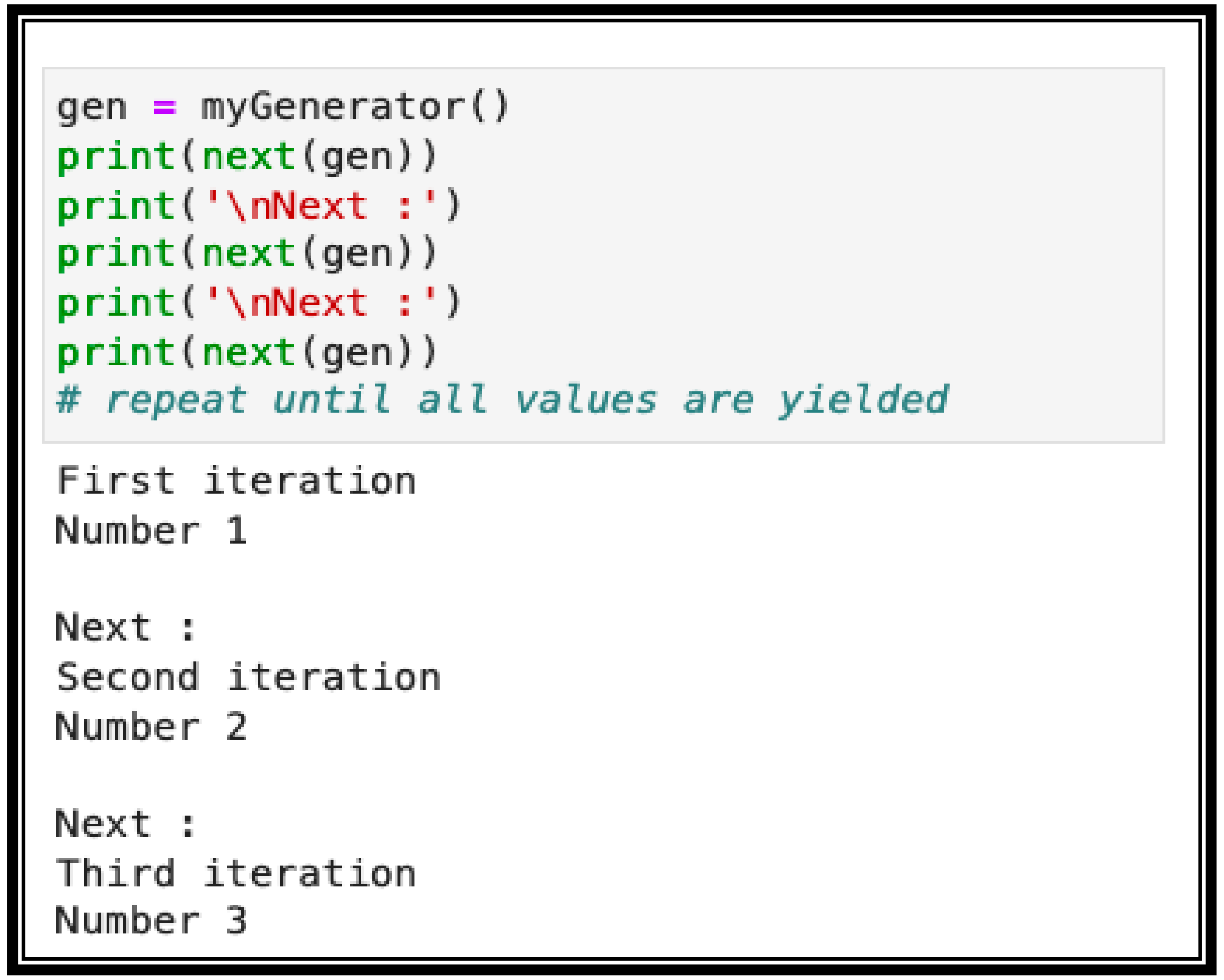
Note

In place of multiple *yield* statements, a *loop* can also be used inside the function to generate the values.

Ways to Use a Generator

The generator function can be used in two ways:

It can be used with the built-in *next* function.



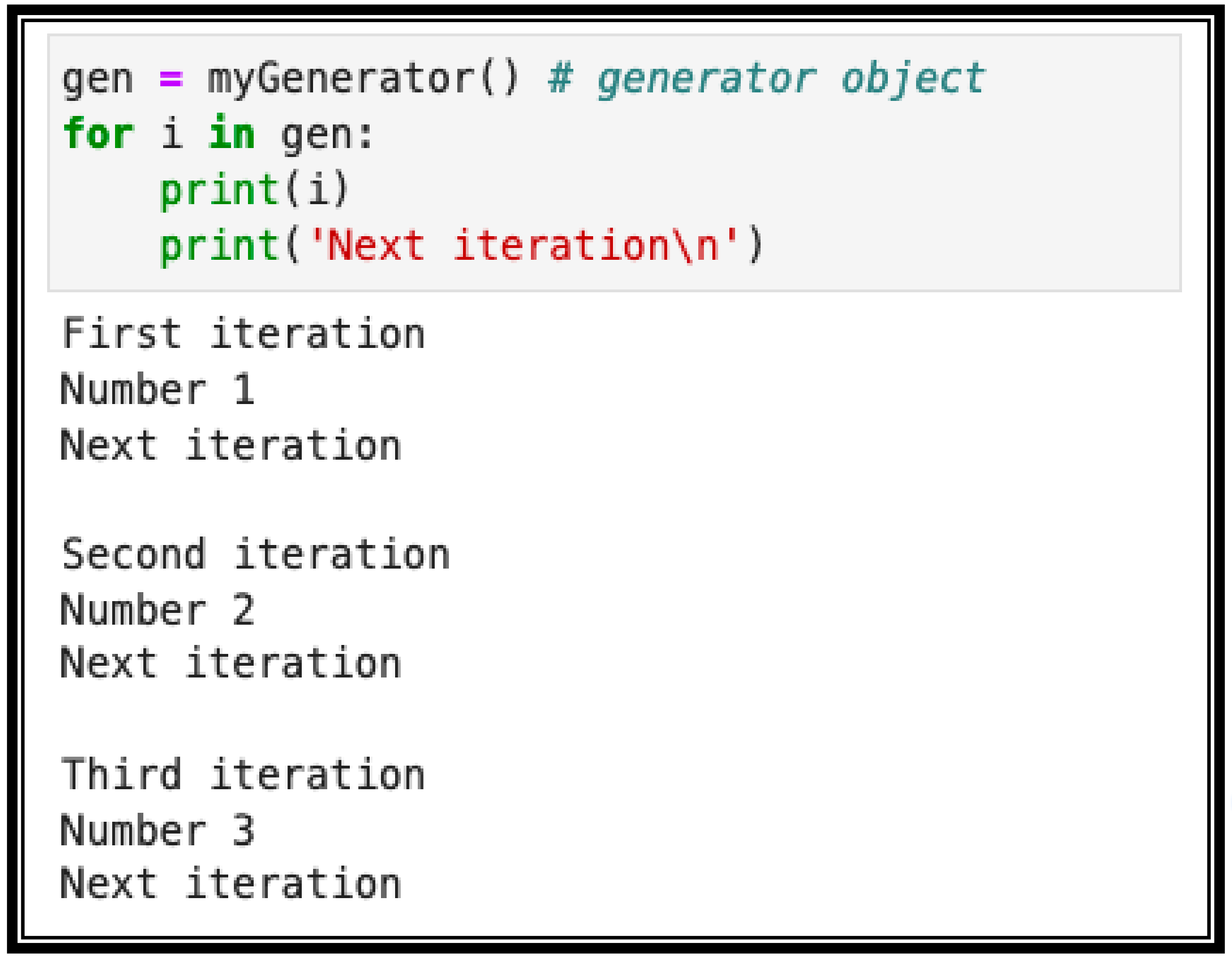
```
gen = myGenerator()
print(next(gen))
print('\nNext :')
print(next(gen))
print('\nNext :')
print(next(gen))
# repeat until all values are yielded
```

```
First iteration
Number 1

Next :
Second iteration
Number 2

Next :
Third iteration
Number 3
```

It can be used with a *for* loop as an iteration object.



```
gen = myGenerator() # generator object
for i in gen:
    print(i)
    print('Next iteration\n')
```

```
First iteration
Number 1
Next iteration

Second iteration
Number 2
Next iteration

Third iteration
Number 3
Next iteration
```

Lambda Functions

Lambda Functions

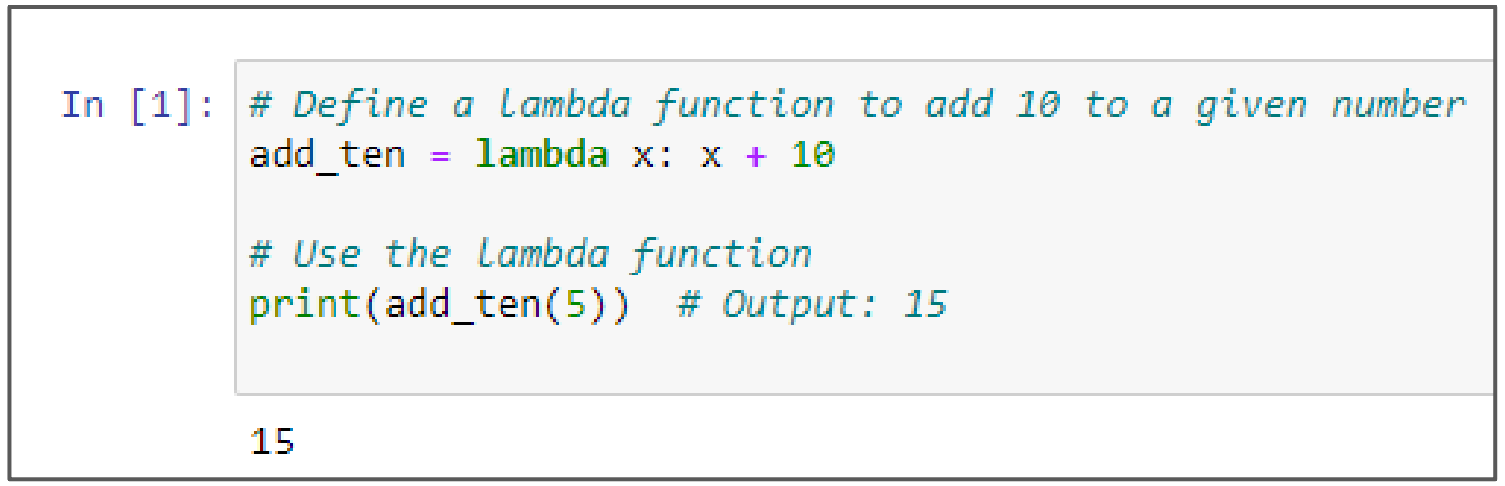
Python lambda functions are anonymous functions defined using the keyword *lambda.*

Features:

- A lambda function may have any number of arguments but contains only one expression.
- The lambda functions are syntactically restricted to only a single expression, which is evaluated and returned.
- A lambda function can be defined without assigning a variable or explicitly calling it.
- **Syntax:** lambda arguments: expression

Lambda Function: Example

This example demonstrates how to use a lambda function in Python to add 10 to a given number:



```
In [1]: # Define a lambda function to add 10 to a given number
        add_ten = lambda x: x + 10

        # Use the lambda function
        print(add_ten(5))  # Output: 15

        15
```

Function Types

Types of Functions

Functions can be of two types:

Built-in functions	User-defined functions
Built-in functions are predefined functions in the programming framework.	Python supports the creation of customized, user-defined functions to perform specific tasks.
Python has basic built-in functions, such as len(), sum(), type(), slice(), next(), help(), and format().	A user can give any function a name with any number of arguments.

Functional Programming Implementation

Functional Programming Methods

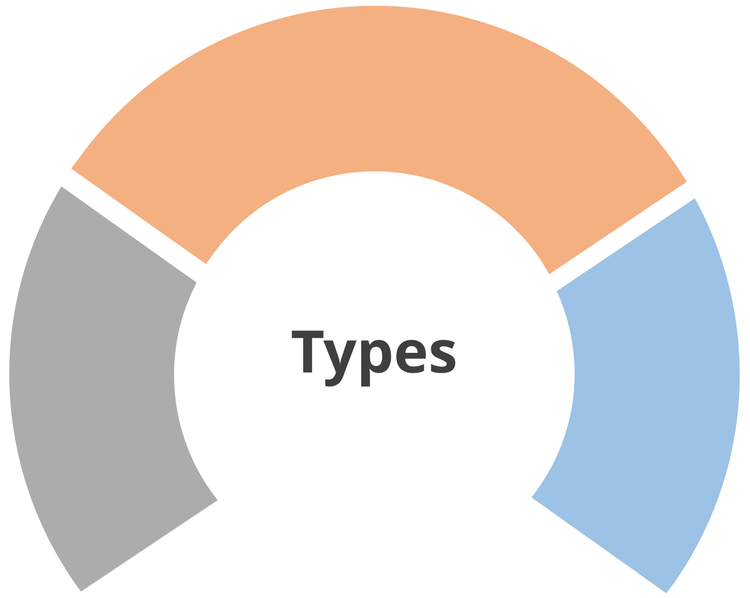
These built-in functions facilitate functional programming in Python, which uses methods to define computations.

Filter function

Syntax: filter(func, iterable)

Map function

Syntax: map(func, iterable)



Reduce function

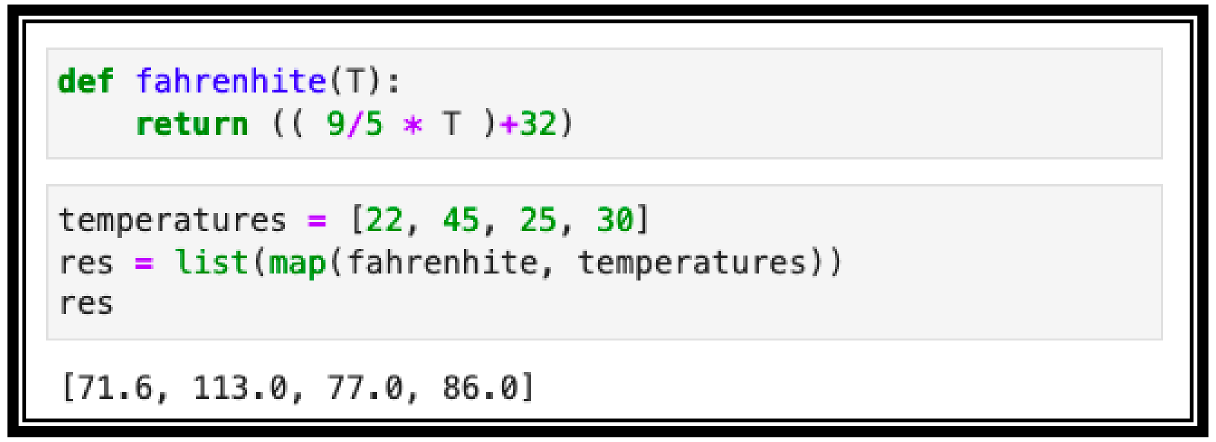
Syntax: reduce(func, iterable)

The *map* and *filter* functions are basic built-in functions, whereas the *reduce* function is part of a module named **_functools**.

map()

A *map* function can apply a function to each element of an iteration object.

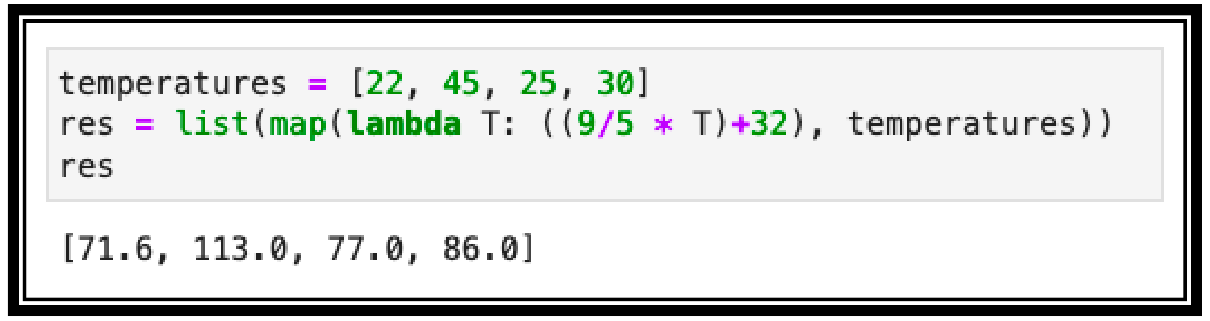
Example



```
def fahrenhite(T):
    return (( 9/5 * T )+32)
```

```
temperatures = [22, 45, 25, 30]
res = list(map(fahrenhite, temperatures))
res
```

```
[71.6, 113.0, 77.0, 86.0]
```



```
temperatures = [22, 45, 25, 30]
res = list(map(lambda T: ((9/5 * T)+32), temperatures))
res
```

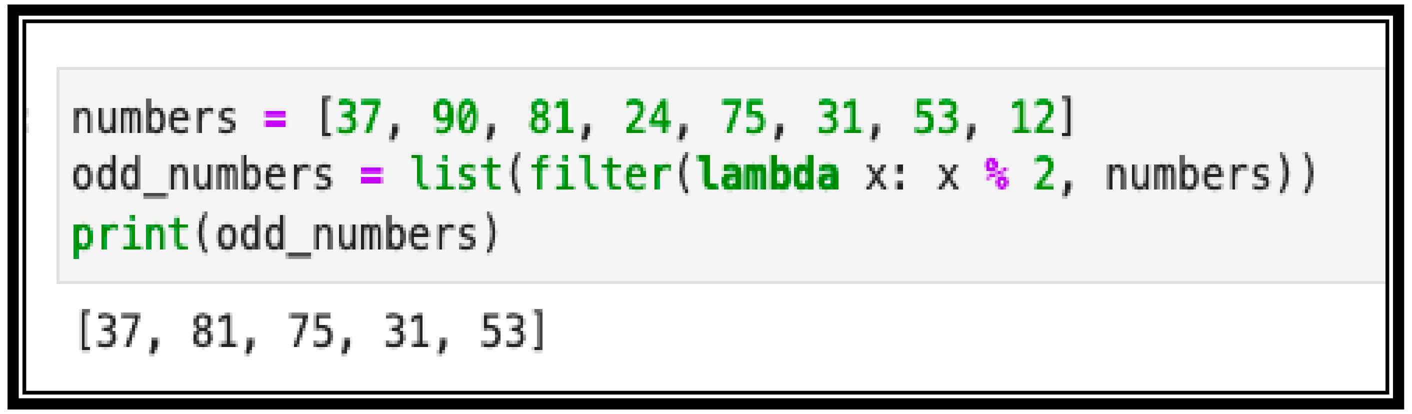
```
[71.6, 113.0, 77.0, 86.0]
```

- The *map* function takes two arguments: a function and an iterable object.
- It applies the given function to all elements of the given iteration object.
- It generates an iterator, which can be converted to a list.
- It can also use the *lambda* function for implementation.

filter()

A *filter* function filters data based on a condition defined in the function.

Example



```
numbers = [37, 90, 81, 24, 75, 31, 53, 12]
odd_numbers = list(filter(lambda x: x % 2, numbers))
print(odd_numbers)
```

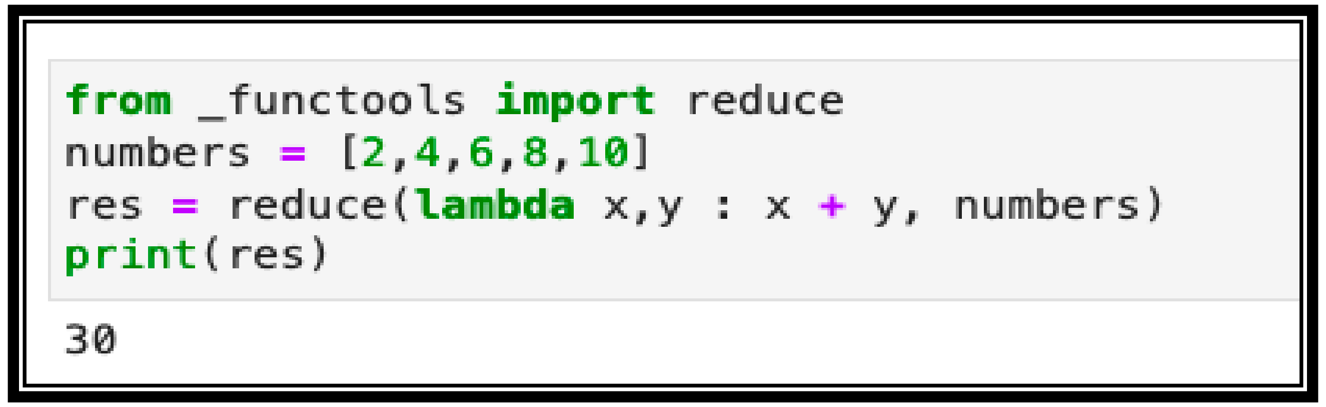
```
[37, 81, 75, 31, 53]
```

- The *filter* function also takes a function and an iterable object and applies the function to each element of the iteration object.
- The given function should return a Boolean value.

reduce()

A *reduce* function implements the mathematical technique of folding on an iteration object.

Example

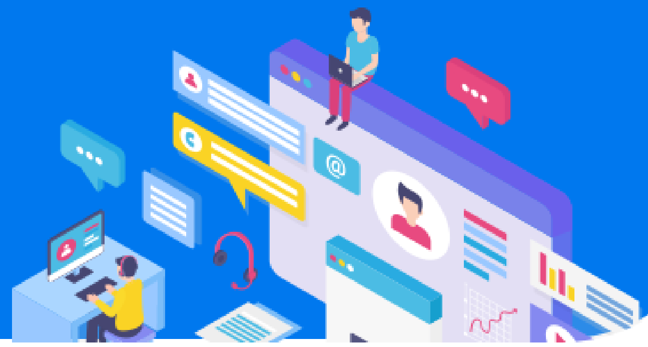


```
from _functools import reduce
numbers = [2,4,6,8,10]
res = reduce(lambda x,y : x + y, numbers)
print(res)
```

```
30
```

- The reduce function uses a given function to process an iterable object.
- The reduce function applies the function continually to each element of the iterable object and produces a single cumulative value.

Assisted Practice: Generators, Lambda, and Built-in Functions



Duration: 10 mins

Objective: In this demonstration, you will learn to use the generators, lambda, and built-in functions

Steps to perform:

Step 1: Define a generator function that yields values one by one

Step 2: Introduce the lambda function and use it with built-in functions

Step 3: Apply built-in functions like map(), filter(), or reduce() to process lists efficiently

Note

Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.

ASSISTED PRACTICE

Key Takeaways

- Functions are an important structure of any programming language to create a modular and reusable application.
- Python functions use the *def* keyword to define functions, can take 0 or more arguments, and can return a value or None.
- Variables declared in the function have a local scope, and variables declared outside a function have a global scope.
- Iterators are a way to traverse over a string, list, or tuple, and they exhibit the concept of generators.
- The Lambda keyword defines an anonymous function in Python.

Knowledge Check

Knowledge Check

1

The advantages of functions are:

A. Reusability

B. Ease of use

C. Scalability

D. All of the above

Knowledge Check

1

The advantages of functions are:

A. Reusability

B. Ease of use

C. Scalability

D. All of the above

The correct answer is **D**

Functions are reused, easy to access, and scalable.

Knowledge Check 2

Information can be passed to a function in the form of ______.

A. Arguments

B. Names

C. Tasks

D. All of the above

Knowledge Check

2

Information can be passed to a function in the form of ______.

A. Arguments

B. Names

C. Tasks

D. All of the above

The correct answer is **A**

Information can be passed to a function in the form of arguments or parameters.

Knowledge Check

3

A function is defined using the keyword _____.

A. func

B. function

C. def

D. All of the above

Knowledge Check

3

A function is defined using the keyword _____.

A. func

B. function

C. def

D. All of the above

The correct answer is **C**

A function is defined using the keyword def, followed by a function name and parenthesis.

Thank You!