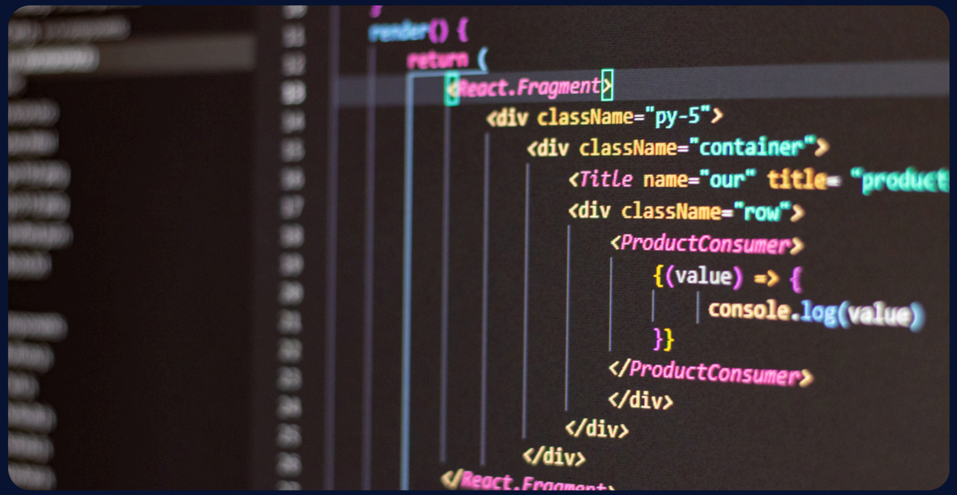


```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title= "product
          <div className="row">
            <ProductConsumer>
              {(value) => {
                console.log(value)
              }}
            </ProductConsumer>
          </div>
        </div>
      </div>
    </React.Fragment>
```

Python Function Presentation

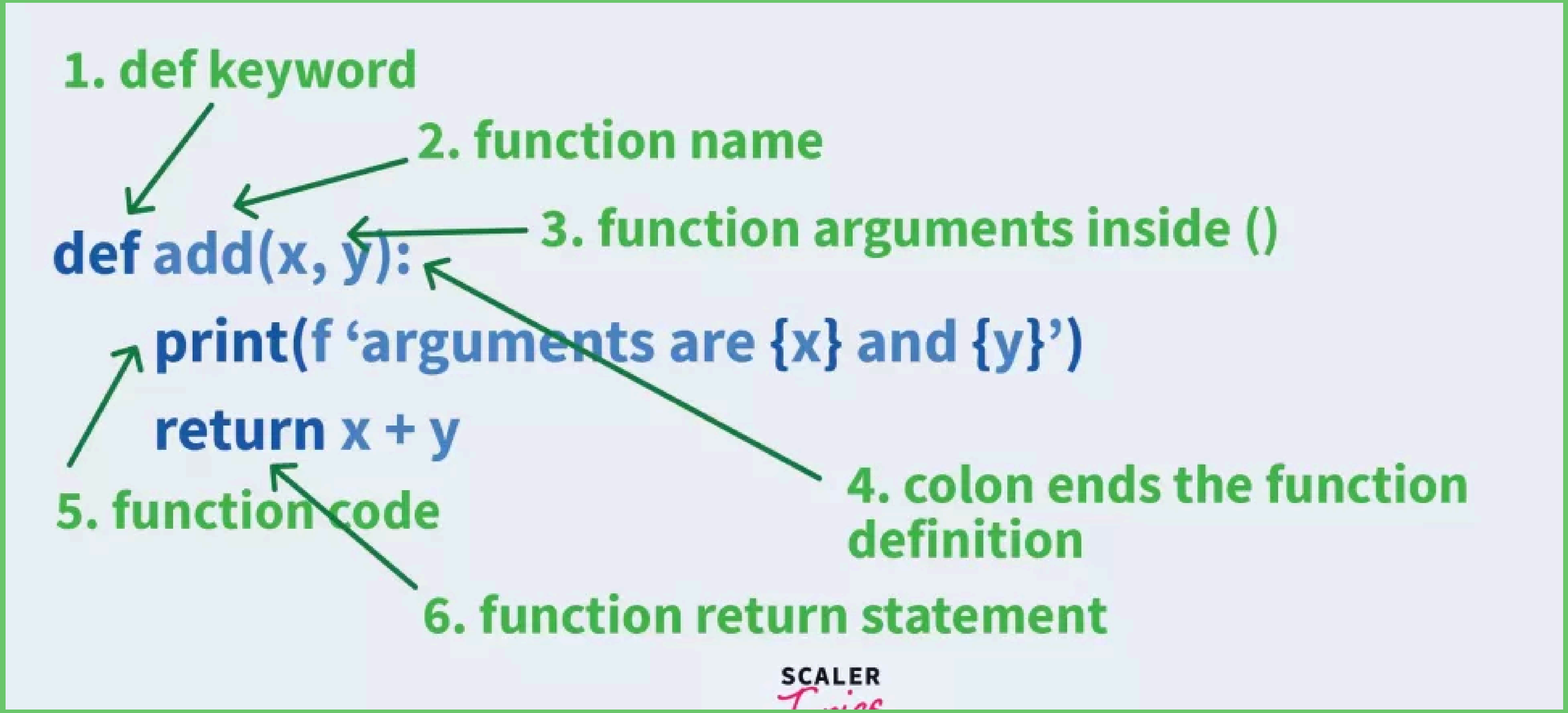
Presented By
Rahul Yadav

What is a Function ?

- A function in Python is a reusable block of code designed to perform a specific task. Functions take input, process it, and return a result.
- Functions are defined using the def keyword, followed by the function name and parentheses.

Real-World Examples:

- E-commerce
- Banking
- Healthcare



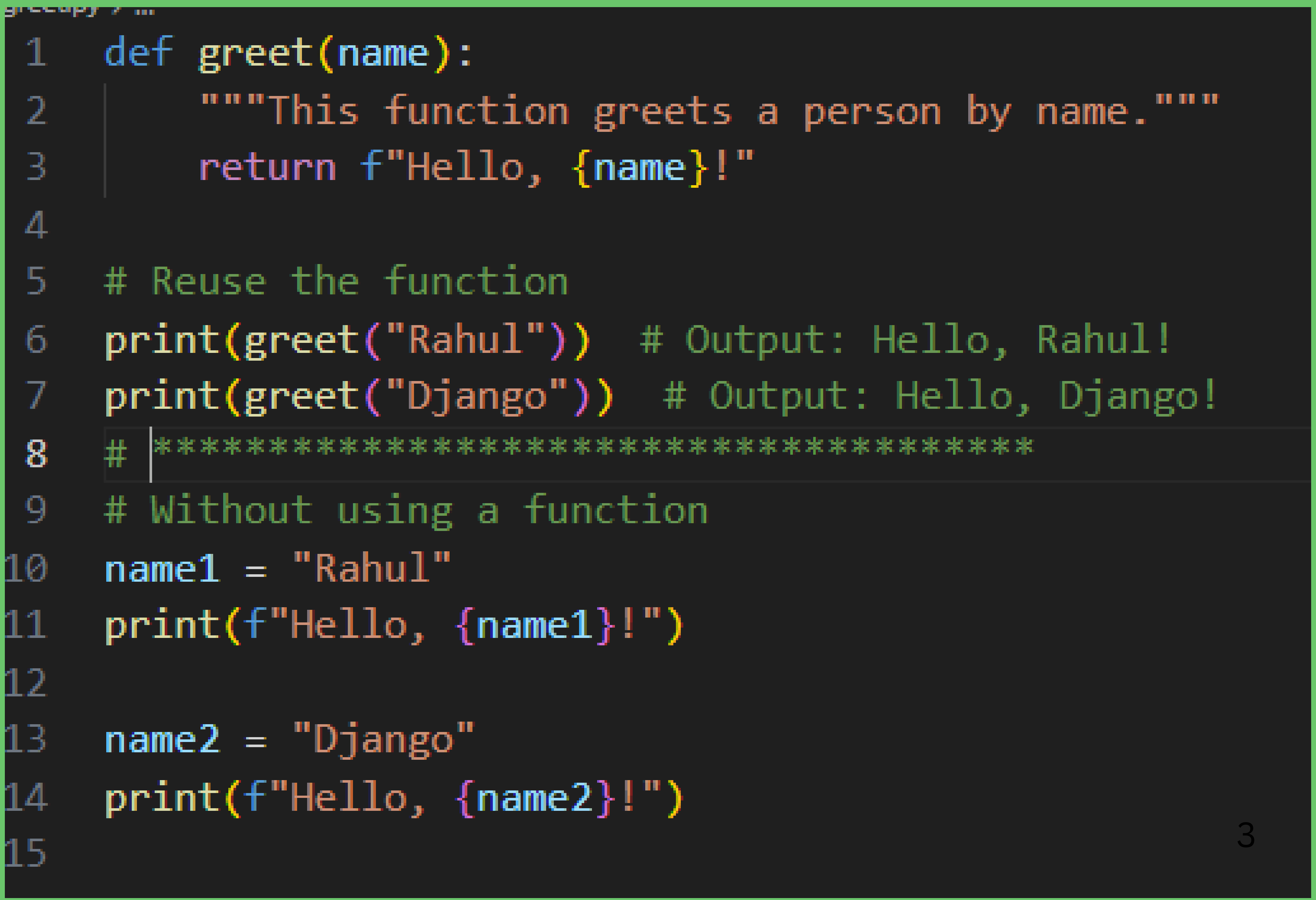
```
def add(x, y):
    print(f 'arguments are {x} and {y}')
    return x + y
```

1. def keyword
2. function name
3. function arguments inside ()
4. colon ends the function definition
5. function code
6. function return statement
SCALER Topics

Why to use function ?

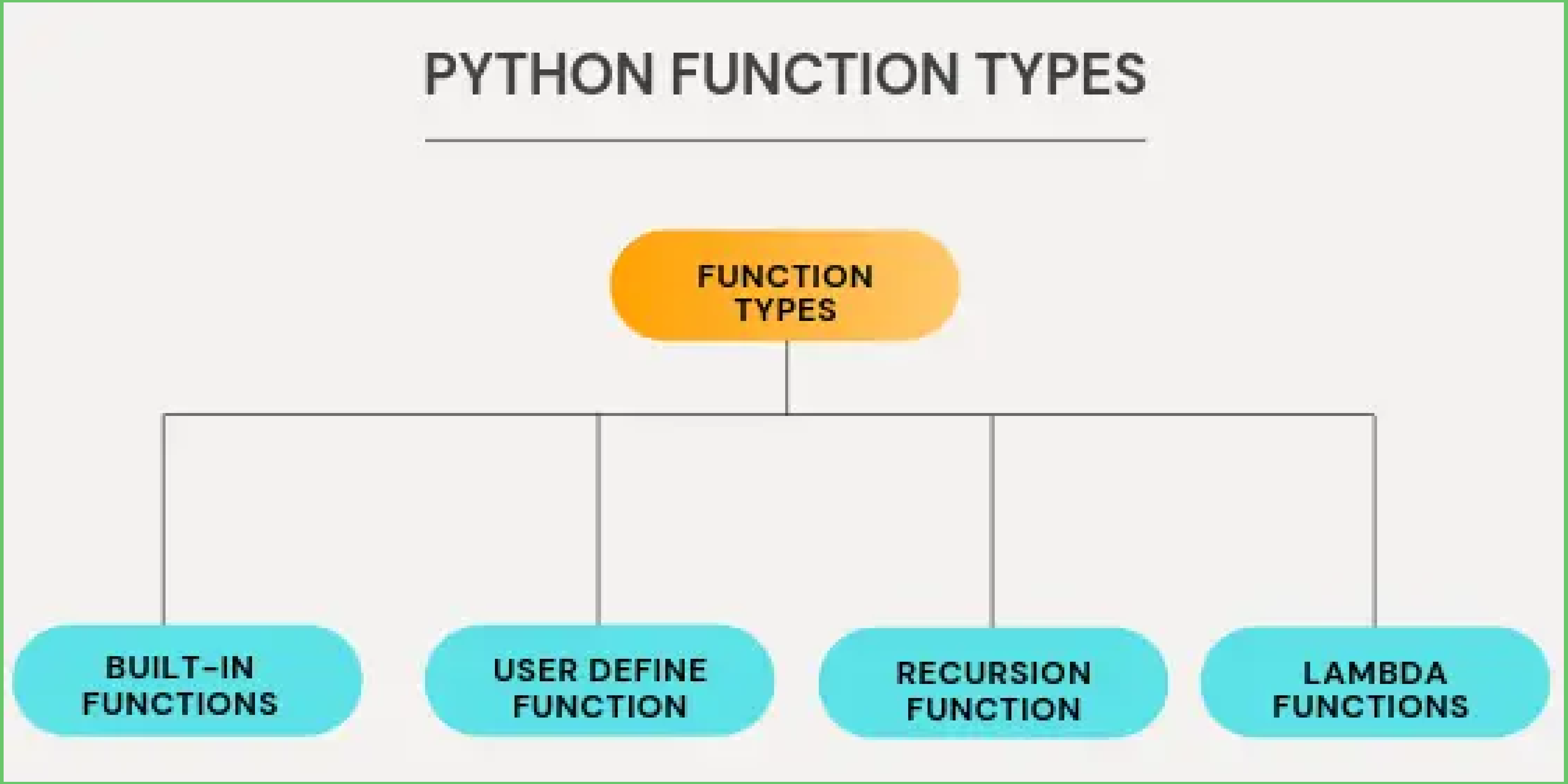
Using Python functions is essential for efficient, clean, and reusable code. Here are some key reasons to use Python functions:

1. Code Reusability
2. Improved Readability
3. Avoiding Redundancy
4. Scalability
5. Modularity
6. Easy Debugging



```python
def greet(name):
    """This function greets a person by name."""
    return f"Hello, {name}!"

# Reuse the function
print(greet("Rahul"))  # Output: Hello, Rahul!
print(greet("Django"))  # Output: Hello, Django!
# ***************************************
# Without using a function
name1 = "Rahul"
print(f"Hello, {name1}!")

name2 = "Django"
print(f"Hello, {name2}!")

```

Types of Function :

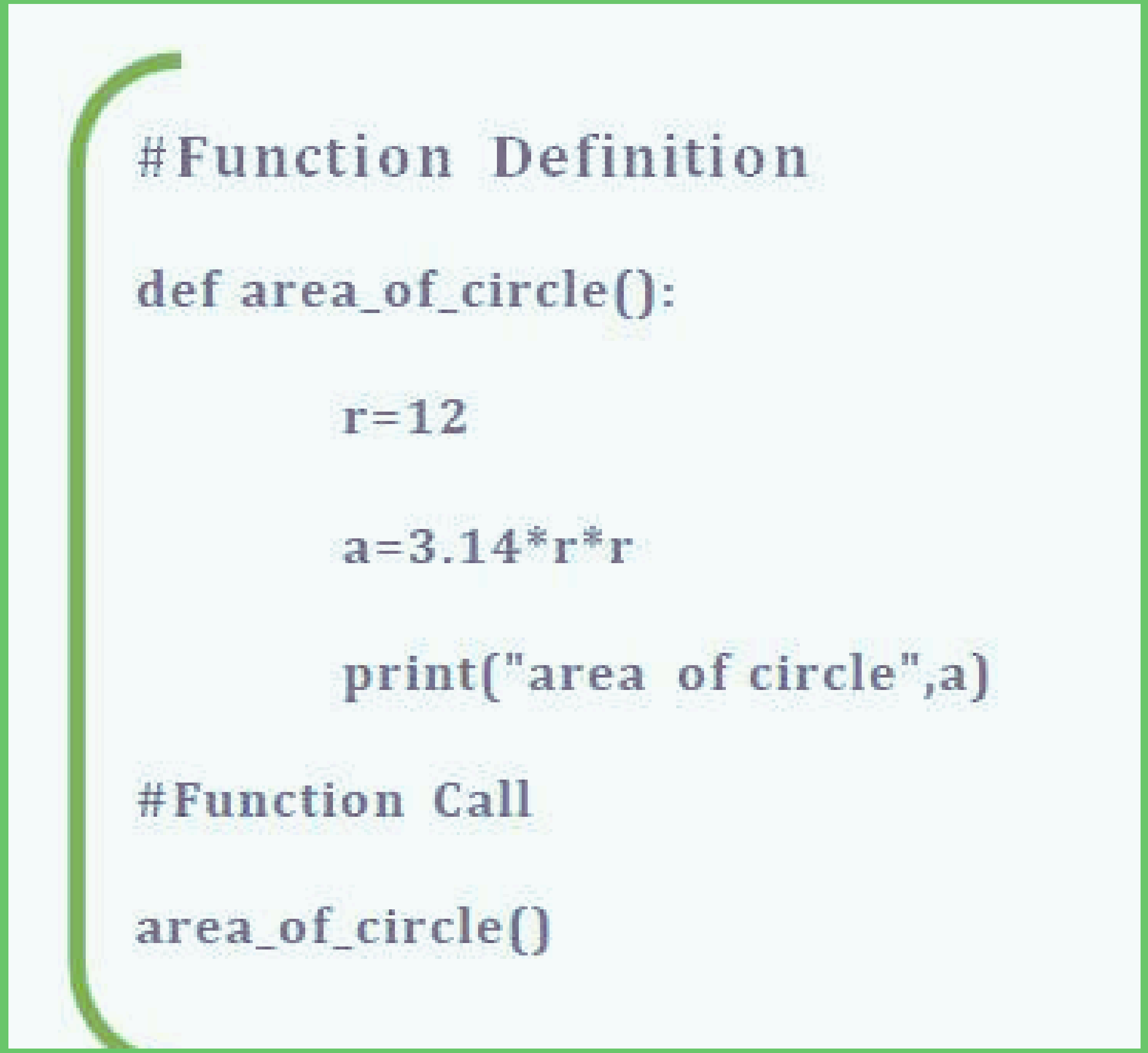


User defined Function:

User-defined functions in programming are custom functions created by developers to perform specific tasks.

1. Functions Without Parameters and Without Return Value

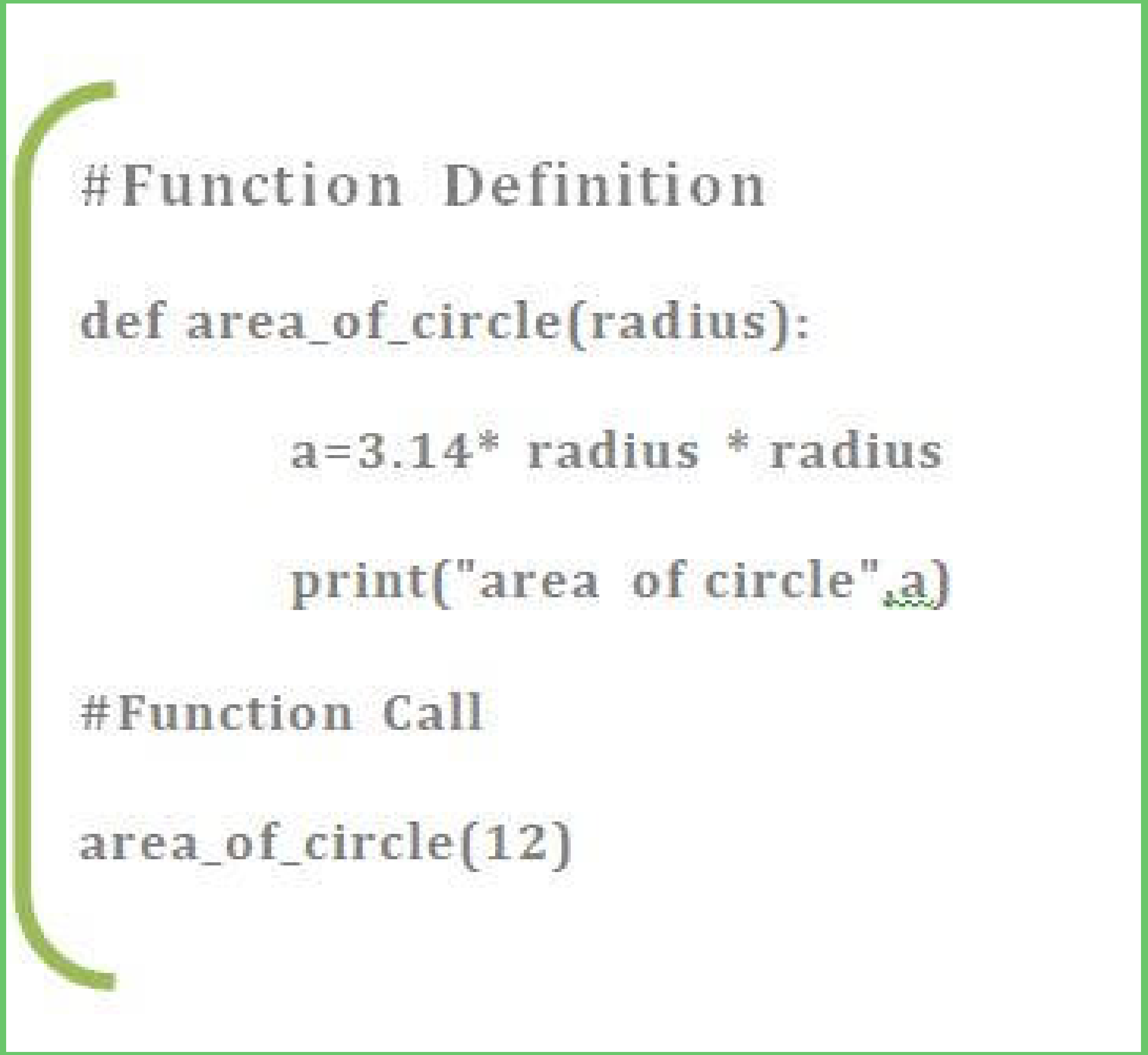
- These functions do not accept any parameters and do not return any value.
- They simply execute a set of instructions.



```
#Function Definition
def area_of_circle():
        r=12
        a=3.14*r*r
        print("area of circle",a)
#Function Call
area_of_circle()
```

2. Function with Arguments and No Return Value

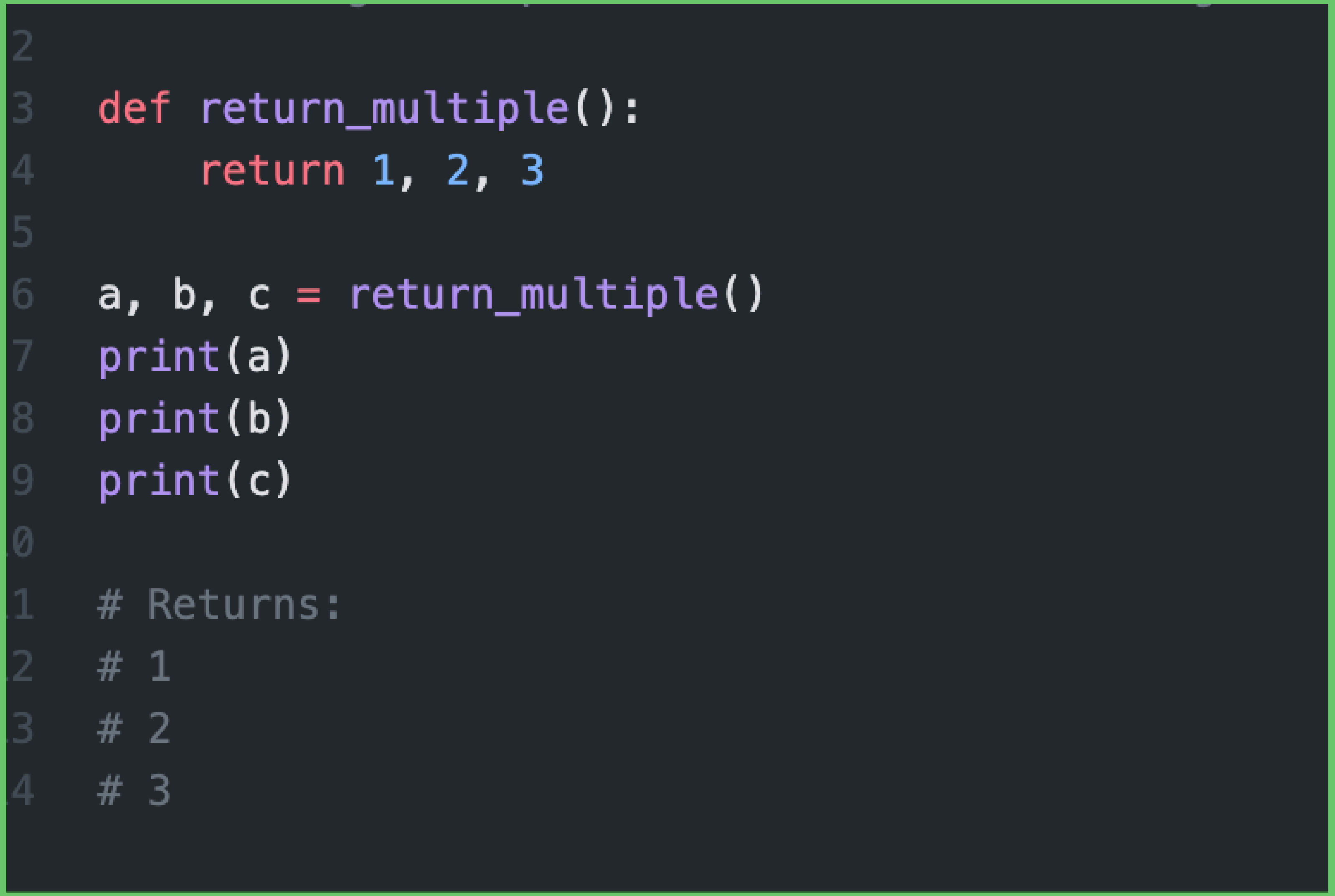
- These functions accept input arguments but do not return any value.
- Typically used for operations where inputs modify some external state.



```
#Function Definition
def area_of_circle(radius):
        a=3.14* radius * radius
        print("area of circle",a)
#Function Call
area_of_circle(12)
```

3. Function with No Arguments and a Return Value

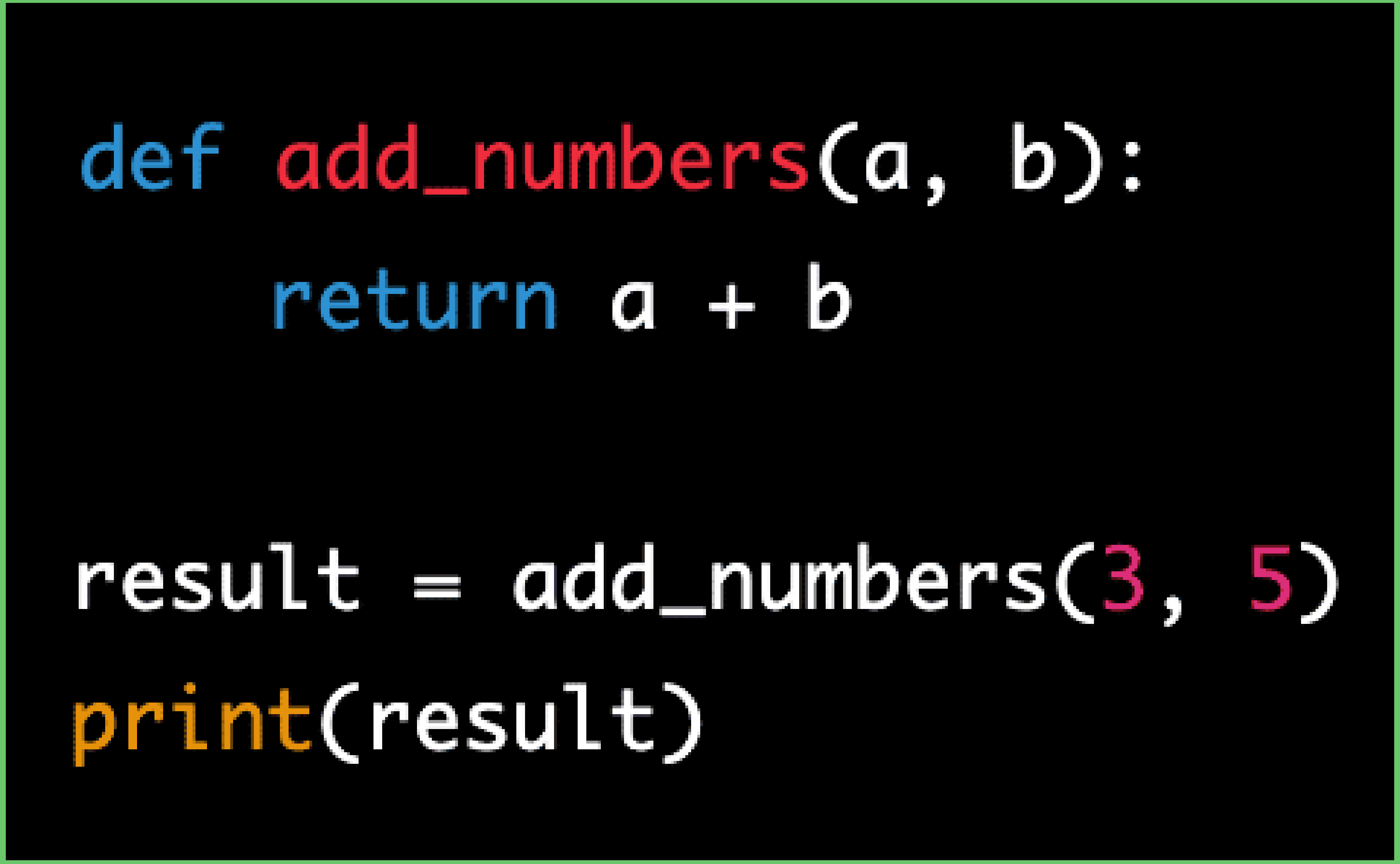
- These functions do not take any input but return a value.
- Useful for generating data without needing external input.



```python
def return_multiple():
    return 1, 2, 3

a, b, c = return_multiple()
print(a)
print(b)
print(c)

# Returns:
# 1
# 2
# 3
```

4. Function with Arguments and a Return Value

- These functions take inputs and return a result.
- Commonly used for performing computations or transformations.



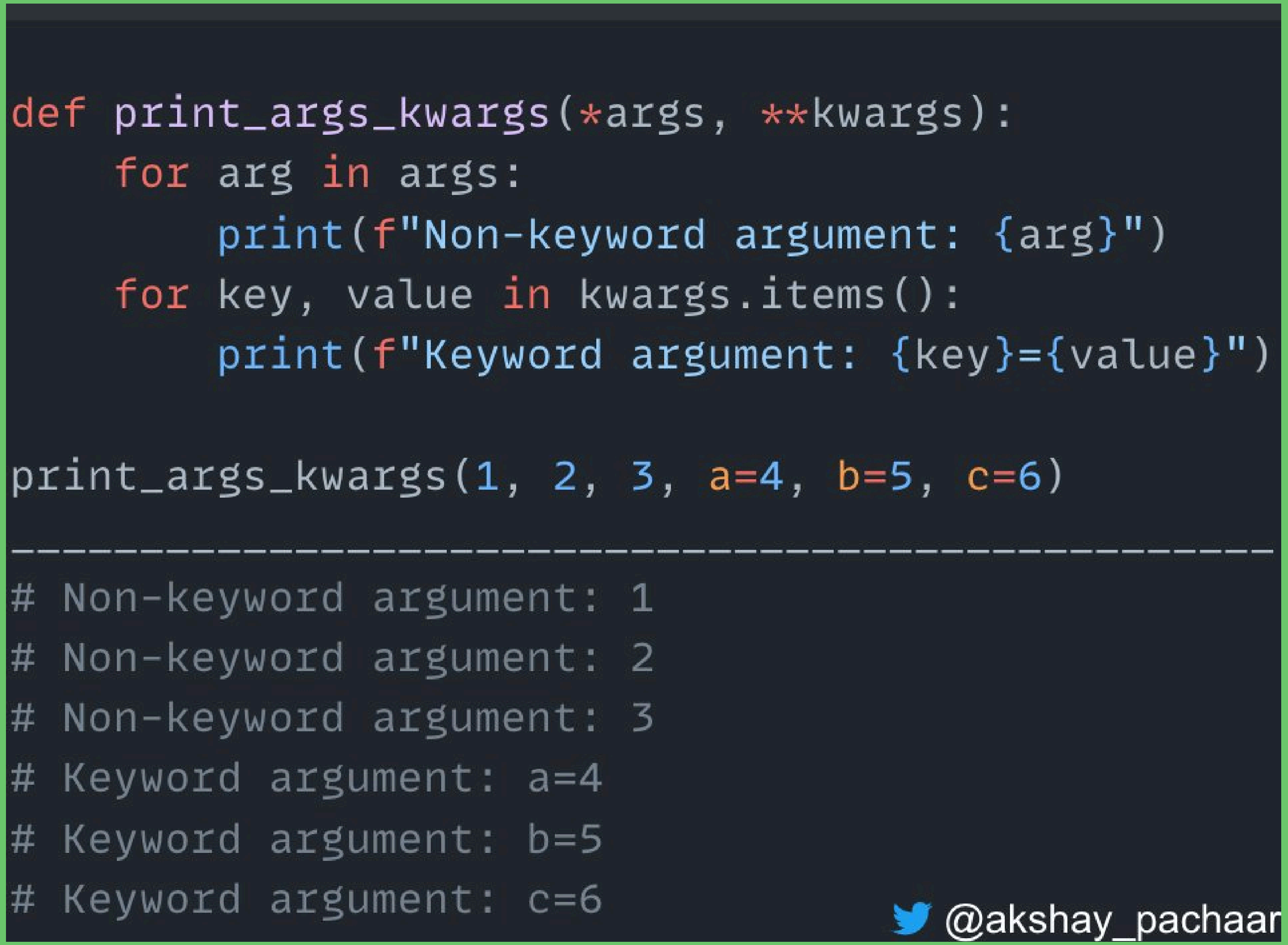
```python
def add_numbers(a, b):
    return a + b

result = add_numbers(3, 5)
print(result)
```

1. *args (Non-Keyword Variable Arguments)

- Used to pass a variable number of non-keyworded arguments to a function.
- Inside the function, *args is treated as a tuple of arguments.

2. **kwargs (Keyword Variable Arguments)

- Used to pass a variable number of keyworded arguments to a function.
- Inside the function, **kwargs is treated as a dictionary.

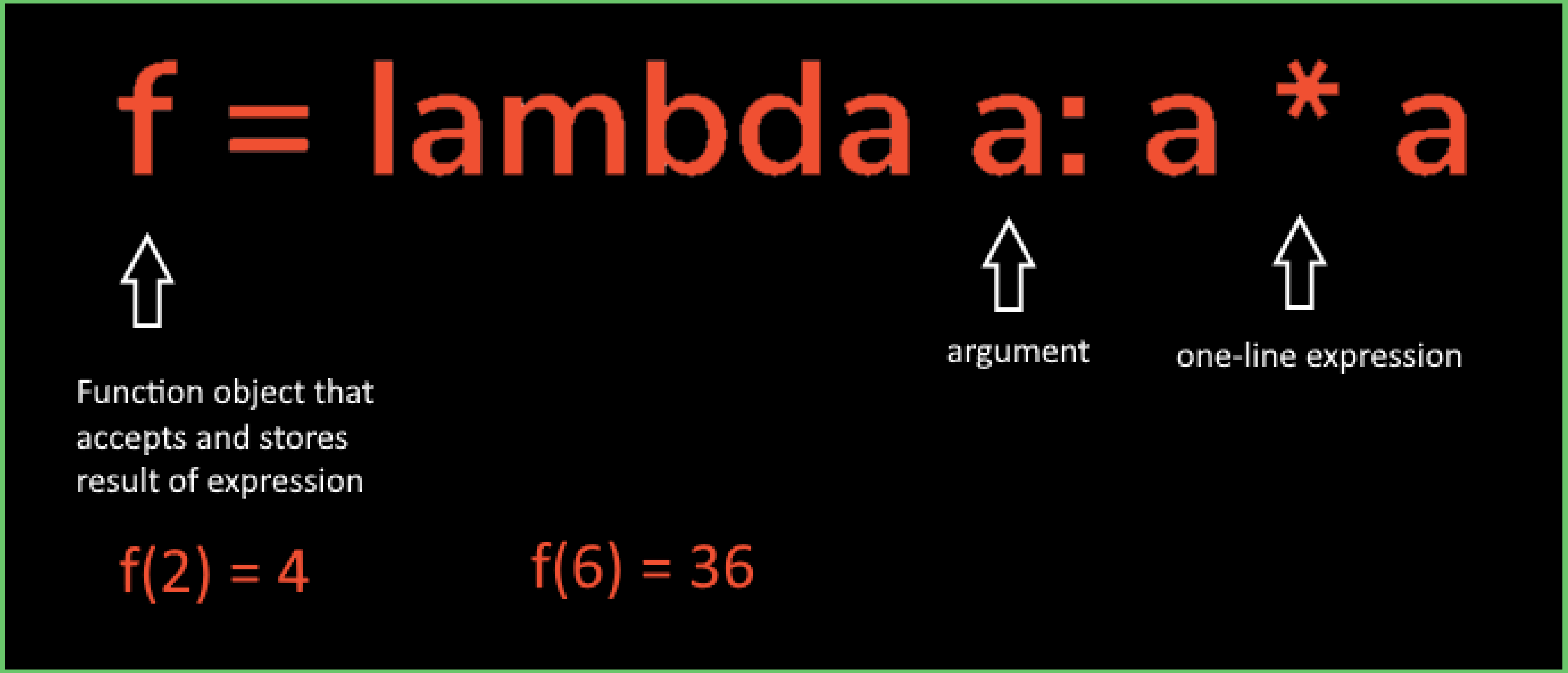


```python
def print_args_kwargs(*args, **kwargs):
    for arg in args:
        print(f"Non-keyword argument: {arg}")
    for key, value in kwargs.items():
        print(f"Keyword argument: {key}={value}")

print_args_kwargs(1, 2, 3, a=4, b=5, c=6)
```

```
------------------------------------------------
# Non-keyword argument: 1
# Non-keyword argument: 2
# Non-keyword argument: 3
# Keyword argument: a=4
# Keyword argument: b=5
# Keyword argument: c=6
```

Lambda function :

- A lambda function in Python is a small, anonymous function that is defined using the lambda keyword.
- It can have any number of arguments, but it can only contain a single expression.
- The result of the expression is returned when the function is called.



Built-in functions :

- Built-in functions in Python are pre-defined functions that are always available for use without the need for importing any libraries.

1. Type Conversion Functions

- int(): Converts to an integer.
- float(): Converts to a floating-point number.
- str(): Converts to a string.
- bool(): Converts to a boolean (True or False).

- list(), tuple(), set(), dict(): Convert to respective data structures.
- ord(): Converts a character to its Unicode code.
- chr(): Converts a Unicode code to its character.
- hex(), oct(), bin(): Converts an integer to hexadecimal, octal, or binary.

2. Input/Output Functions

- print(): Outputs data to the console.
- input(): Reads input from the user as a string.
- open(): Opens a file for reading or writing.

3. Iterables and Sequence Functions

- len(): Returns the length of a sequence or collection.
- max(), min(): Return the maximum or minimum value in an iterable.
- sum(): Returns the sum of elements in an iterable.
- sorted(): Returns a sorted list from an iterable.
- reversed(): Returns a reversed iterator.
- enumerate(): Returns an enumerator with index and value pairs.
- zip(): Combines multiple iterables element-wise into tuples.

4. Logical Functions

- all(): Returns True if all elements in an iterable are true.
- any(): Returns True if any element in an iterable is true.

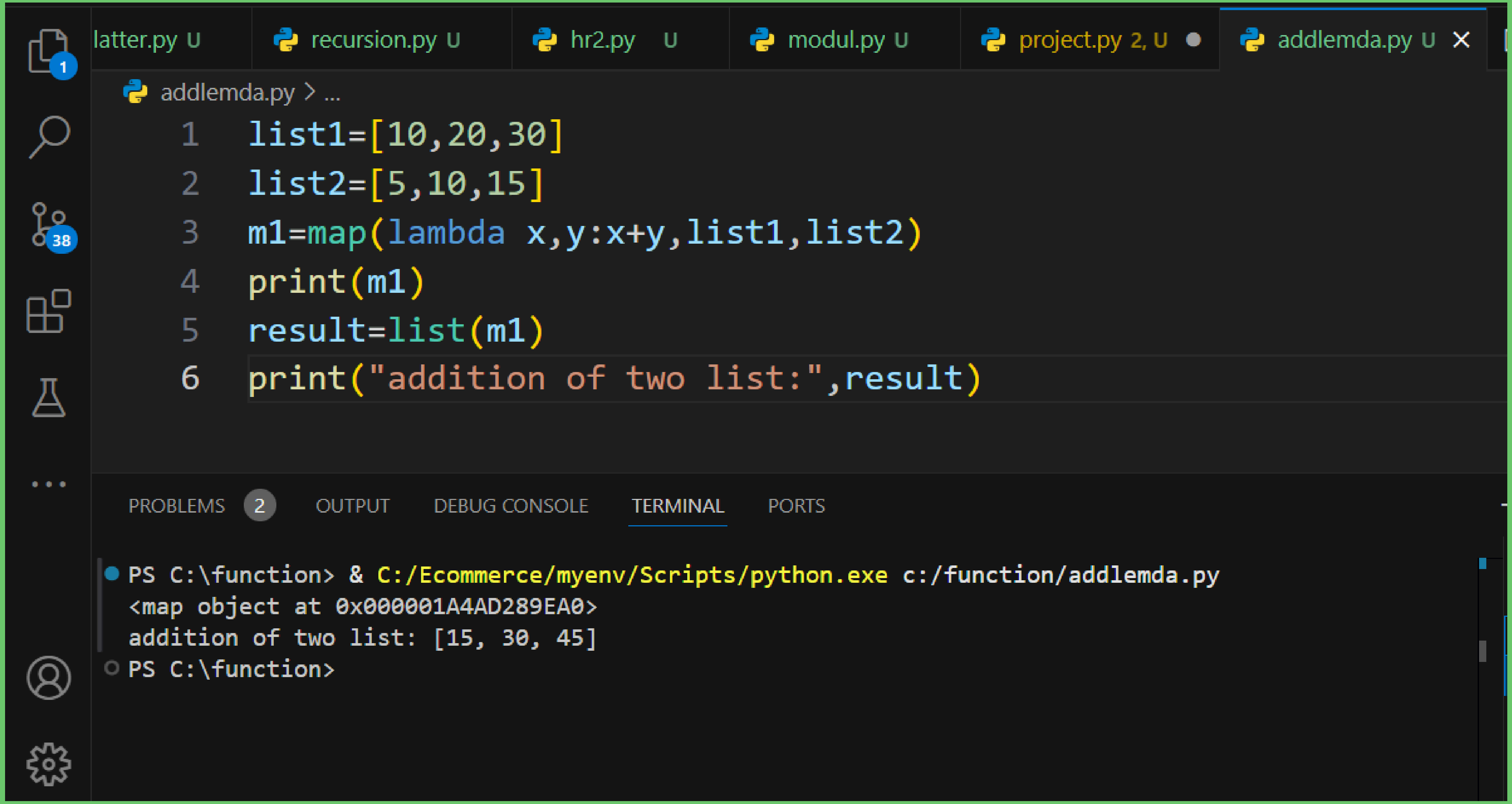
5. Object and Attribute Functions

- type(): Returns the type of an object.
- id(): Returns the unique ID of an object.
- isinstance(): Checks if an object is an instance of a class.
- dir(): Returns a list of valid attributes for an object.
- getattr(), setattr(), hasattr(): Work with attributes of an object.

6.map() function

- The map() function applies a function to every item in an iterable and returns a map object (an iterator).

Syntax: map(function, iterable)



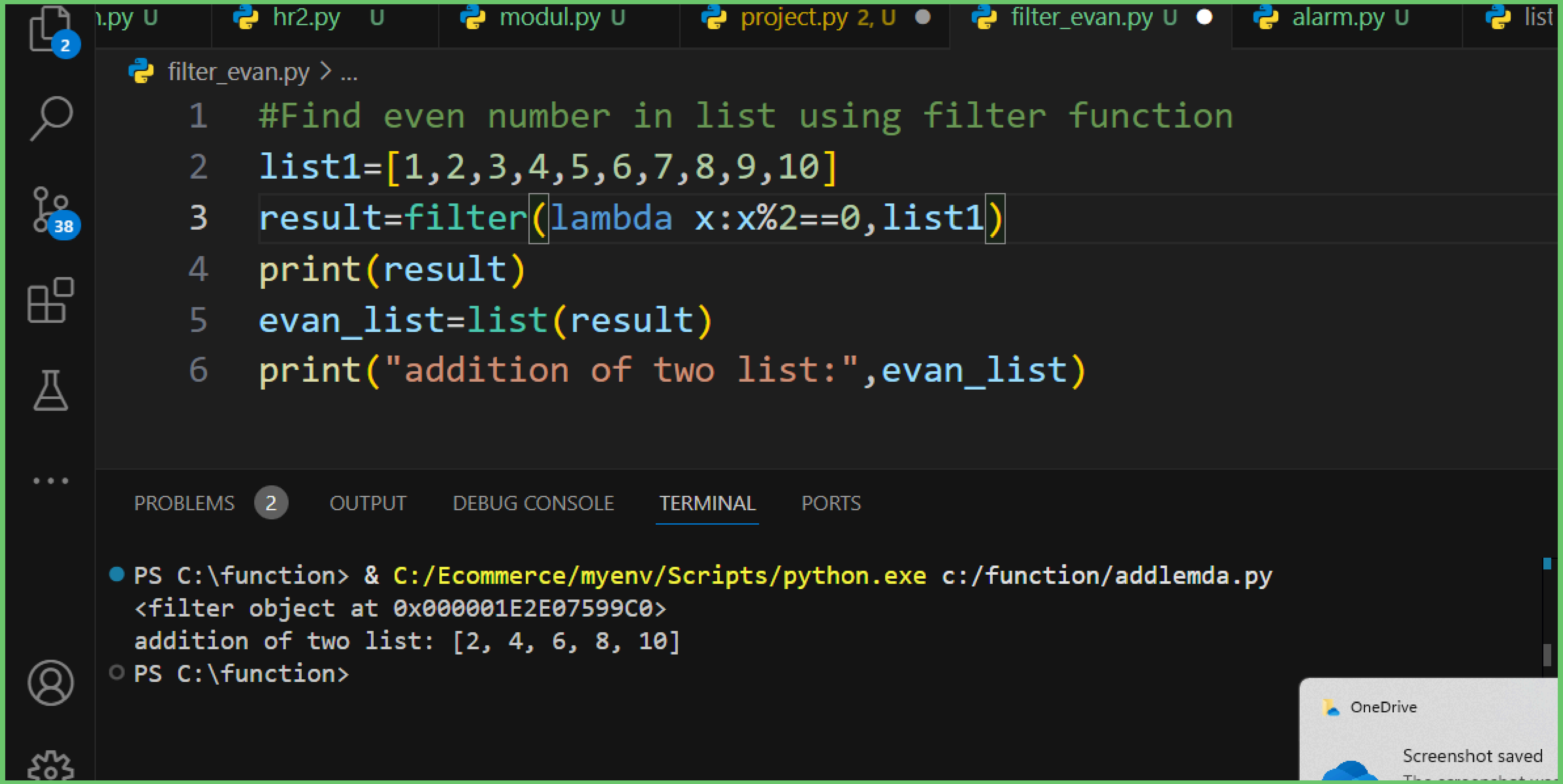
```python
list1=[10,20,30]
list2=[5,10,15]
m1=map(lambda x,y:x+y,list1,list2)
print(m1)
result=list(m1)
print("addition of two list:",result)
```

```
PS C:\function> & C:/Ecommerce/myenv/Scripts/python.exe c:/function/addlemda.py
<map object at 0x000001A4AD289EA0>
addition of two list: [15, 30, 45]
PS C:\function>
```

7. filter() function

- The filter() function filters items in an iterable based on a condition and returns an iterator.

Syntax: filter(function, iterable)



```python
#Find even number in list using filter function
list1=[1,2,3,4,5,6,7,8,9,10]
result=filter(lambda x:x%2==0,list1)
print(result)
evan_list=list(result)
print("addition of two list:",evan_list)
```

```
PS C:\function> & C:/Ecommerce/myenv/Scripts/python.exe c:/function/addlemda.py
<filter object at 0x000001E2E07599C0>
addition of two list: [2, 4, 6, 8, 10]
PS C:\function>
```

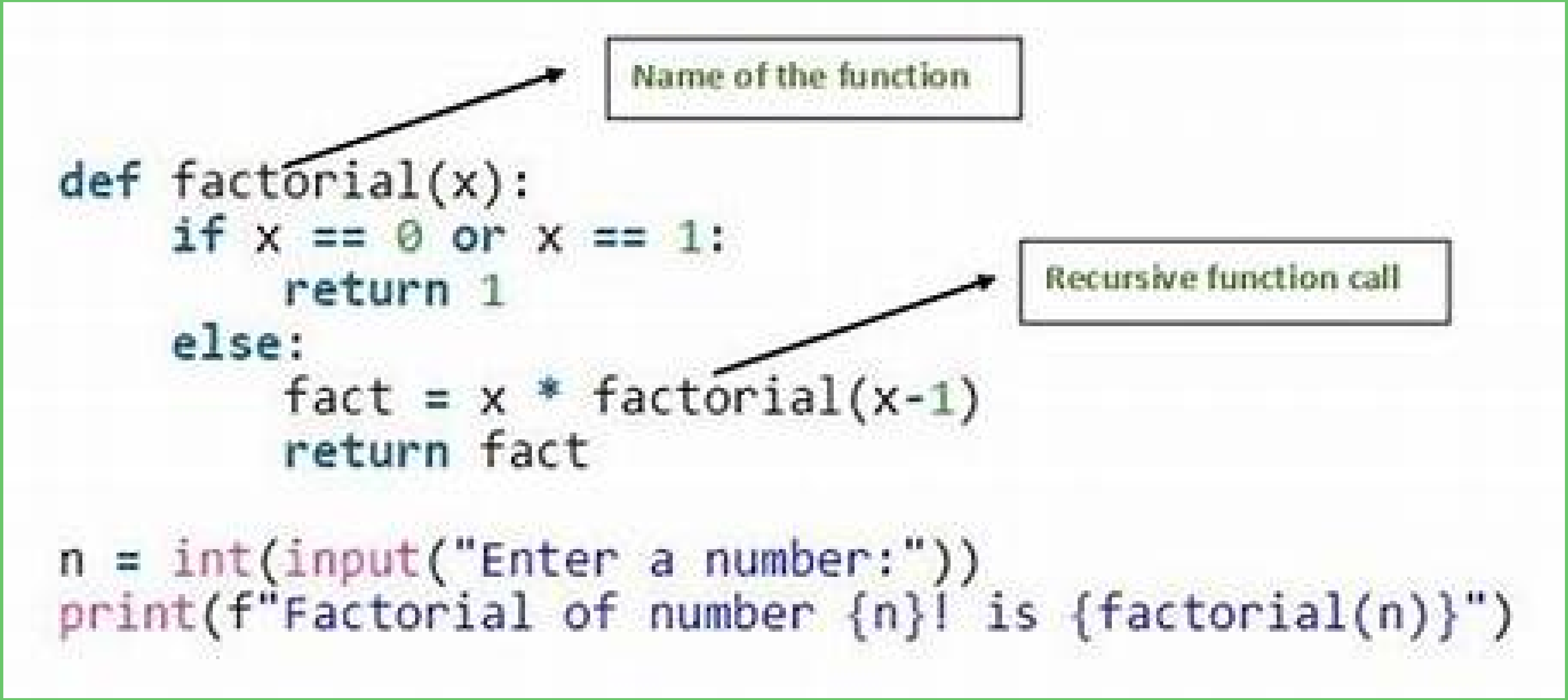
Recursion function:

- A recursion function in Python is a function that calls itself in order to solve a problem.
- Recursion is a technique where the solution to a larger problem depends on solutions to smaller instances of the same problem.

Key Characteristics of Recursive Functions

- Base Case: A condition to stop the recursion and avoid infinite loops.
- Recursive Case: The part of the function where the function calls itself.

Example:



```
def factorial(x):
    if x == 0 or x == 1:
        return 1
    else:
        fact = x * factorial(x-1)
        return fact

n = int(input("Enter a number:"))
print(f"Factorial of number {n}! is {factorial(n)}")
```

```
n=5
Factorial of number 5 ! is 120
```

What is a Decorator?

In Python, a decorator is a function that modifies or extends the behavior of another function or method without permanently modifying it.

Decorators are often used for:

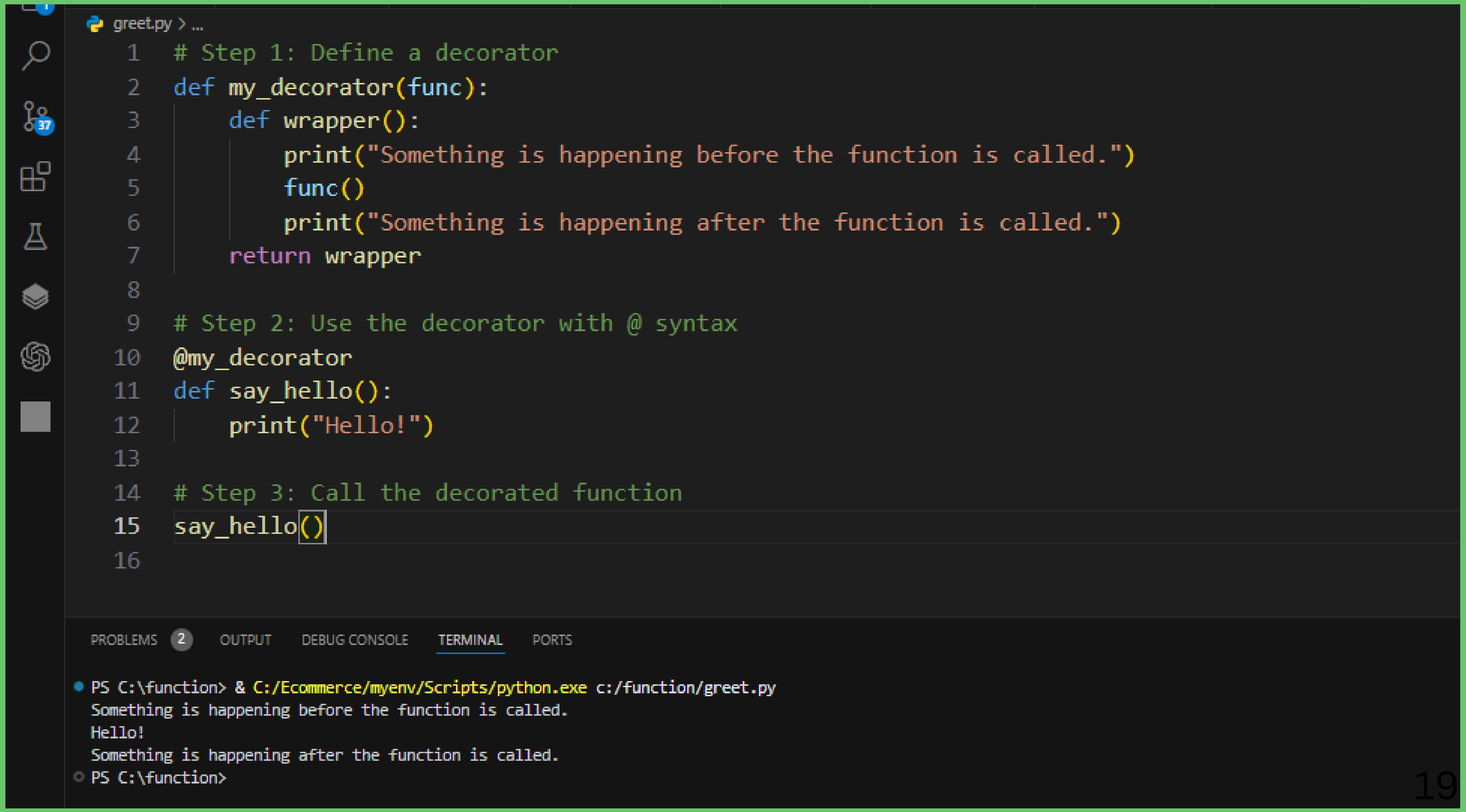
- Logging
- Access control (authentication/authorization)
- Memoization (caching)
- Input validation
- Timing execution

How Decorators Work (Step-by-Step)

- A decorator is a callable (usually a function) that takes a function as an argument.
- It wraps the original function with additional behavior and returns a new function (or the original function).

Built-in Decorators:

- @staticmethod
- @classmethod
- @property

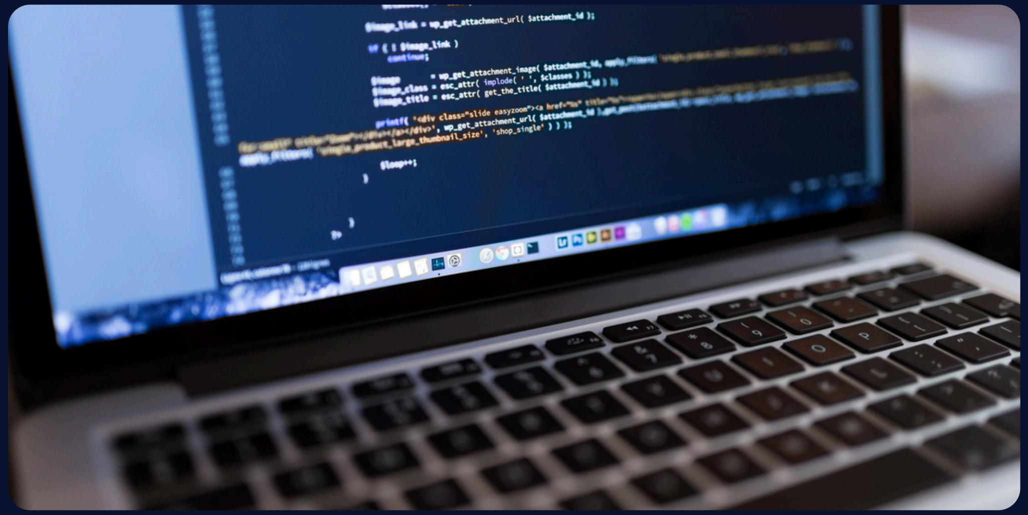


```python
# Step 1: Define a decorator
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

# Step 2: Use the decorator with @ syntax
@my_decorator
def say_hello():
    print("Hello!")

# Step 3: Call the decorated function
say_hello()
```

```
PS C:\function> & C:/Ecommerce/myenv/Scripts/python.exe c:/function/greet.py
Something is happening before the function is called.
Hello!
Something is happening after the function is called.
PS C:\function>
```

Conclusion :

- Purpose of Functions:
  - Simplify code by breaking it into reusable blocks.
  - Enhance readability, scalability, and debugging.
- Features of Python Functions:
  - Support dynamic inputs through parameters (*args, **kwargs).
  - Allow default arguments for flexibility.
  - Facilitate higher-order operations and decorators.
- Best Practices:
  - Use meaningful names and add docstrings for clarity.
  - Keep functions focused on a single task.
  - Handle errors gracefully with exception handling.



Thank You