TG ACADEMIC RESOURCE SERIES

CLASS 12 COMPUTER SCIENCE

Python User-Defined Functions
Parameters, Arguments, Return Values & Worked Dry Run

SUBJECT	Computer Science / Python Programming
LEVEL	CBSE Class 12 / Senior Secondary
RESOURCE TYPE	Sample learning note
PUBLICATION ID	CS12-PY-001 | Version 1.0

Prepared by Tanmoy Gupta
Computer Science Educator & Industry Professional

This sample introduces modular programming in Python. It shows how a task can be placed inside a function, supplied with input values, and made to return a result to the calling statement.

A focused learning preview—not a complete Class 12 Python programming workbook.

1. Learning Objectives

After studying this note, you should be able to:
• Explain why user-defined functions are used.
• Write a function definition and a function call.
• Distinguish parameters from arguments.
• Explain the role of the return statement.
• Trace the flow of values between the calling code and a function.

2. What Is a User-Defined Function?

A **function** is a named, reusable block of statements designed to perform a specific task. A user-defined function is created by the programmer using the **def** keyword.

General syntax

```
def function_name(parameter_list):
    statement_1
    statement_2
    return value
```

The return statement is optional. A function that reaches the end without an explicit return statement returns the special value **None**.

3. Definition and Function Call

```
def greet():
    print("Welcome to Computer Science")

greet()
```

Output

```
Welcome to Computer Science
```

The function is defined first. The statement greet() calls it. The indented body runs when the call is executed; merely defining the function does not execute its body.

4. Parameters and Arguments

A **parameter** is a name listed in the function definition. An **argument** is the actual value or expression supplied when the function is called.

```
def square(number):   # number is a parameter
    print(number * number)

square(6)              # 6 is an argument
```

Output: 36

5. Returning a Value

A function can send a result back to the caller using **return**. The caller can store that result in a variable or use it directly in an expression.

```
def calculate_area(length, breadth):
    area = length * breadth
    return area

result = calculate_area(8, 5)
print("Area =", result)
```

Output

```
Area = 40
```

print() displays a value. return sends a value back from the function. These statements perform different roles.

6. Common Ways of Supplying Arguments

Argument style	Example call	Explanation
Positional	power(2, 3)	Values are matched to parameters by position.
Keyword	power(base=2, exponent=3)	Values are matched using parameter names.
Default	power(2)	A predefined value is used for an omitted argument if the parameter has a default.

Example: default and keyword arguments

```
def power(base, exponent=2):
    return base ** exponent

print(power(5))
print(power(base=2, exponent=3))
```

Output

```
25
8
```

A parameter with a default value must appear after parameters without default values in the function definition. In a function call, positional arguments generally come before keyword arguments.

7. Worked Program: Total and Average Marks

The function below accepts three subject marks and returns two results: the total and the average.

```
def calculate_result(m1, m2, m3):
    total = m1 + m2 + m3
    average = total / 3
    return total, average

total_marks, average_marks = calculate_result(78, 84, 90)

print("Total =", total_marks)
print("Average =", average_marks)
```

Output

```
Total = 252
Average = 84.0
```

A function can return multiple values. In this example, Python returns them as a tuple, and the caller unpacks the two elements into total_marks and average_marks.

8. Tabular Dry Run

Execution step	m1	m2	m3	total	average
Function call begins	78	84	90	—	—
Calculate total	78	84	90	252	—
Calculate average	78	84	90	252	84.0
Return to caller	—	—	—	252	84.0

The values 78, 84 and 90 are copied into the local parameter names m1, m2 and m3. The function calculates its results and returns them. The caller receives the returned values and stores them in its own variables.

Important: local variables

The names m1, m2, m3, total and average are local to this function. They are created for the function call and are not automatically available as variables in the calling code.

9. Local and Global Scope

A variable created inside a function is normally local to that function. A variable defined at the top level of a module is global to that module. Reading a global variable is possible inside a function, but assigning to it requires an explicit global declaration if the intention is to rebind the global name.

```
school = "Green Valley"

def display_school():
    print(school)

display_school()
```

Output: Green Valley

Avoid changing global variables unnecessarily. Passing values as arguments and returning results generally makes functions easier to test and reuse.

10. Common Errors to Avoid

• Defining a function but forgetting to call it.
• Calling a function with a wrong number of required arguments.
• Confusing print() with return.
• Forgetting indentation inside the function body.
• Trying to use a local variable outside the function.
• Writing a function call before its definition has been executed in a normal top-to-bottom script.

11. Check Your Understanding

Q1. What is the difference between a parameter and an argument?

Q2. What does a Python function return if there is no explicit return statement?

Q3. What is the output of the following code?
def test(x):
return x + 4
print(test(6))

Q4. Identify the default parameter in: def show(message, count=1):

Suggested Answers

A1. A parameter is named in the function definition; an argument is supplied in the function call.
A2. None.
A3. 10.
A4. count, with default value 1.

Where to Go Next

A complete Class 12 programming sequence can continue with recursive functions, string processing, list and dictionary operations, file handling, SQL connectivity and practical programming questions, according to the learner's prescribed syllabus.

In guided classes: students can practise writing functions from problem statements, tracing parameter values, debugging return logic and solving progressively harder examination-style questions.

About this resource
TG Academic Resource Series | CS12-PY-001 | Version 1.0
Prepared by Tanmoy Gupta - Computer Science Educator & Industry Professional

Original educational material prepared for student learning and revision. Redistribution or commercial reproduction without permission is not authorised.