Functions in Python

Introduction to Functions

A **function** is a block of code that performs a specific task and can be reused whenever needed.

👉 Why Functions?

- Avoids writing the same code multiple times.
- Makes code modular, organized, and easy to debug.
- Increases reusability (write once, use many times).

Everyday analogy:

- Think of a **function like a washing machine**.
- You provide input (dirty clothes, detergent).
- It processes them.
- You get output (clean clothes).

Why Functions Matter in DevOps?

In DevOps scripting:

- Functions allow automation scripts to be **reusable** and **organized**.
- Example: Instead of writing the same health-check code multiple times, write one function and call it whenever needed.
- Functions make CI/CD scripts more modular (log processing, deployment steps, retries).

Example DevOps Use-Cases:

1. A function to **check disk space** before deployment.
2. A function to **restart a service** if it fails.

Defining and Calling Functions in Python

Function Definition Syntax

```
def function_name(parameters):
    # block of code
    return value
```

- def → keyword to define a function
- function_name → name of the function

- parameters → inputs (optional)
- return → sends result back (optional)

Example 1: Simple Function

```
def greet():
    print("Hello, welcome to Python!")


greet()
```

Output:

Hello, welcome to Python!

Example 2: Function with Parameters

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


greet_user("Alice")
```

Output:

Hello Alice

Example 3: Function with Return Value

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


result = add(5, 10)
print(result)
```

Output:

15

Types of Functions

1. **Built-in Functions** → already available in Python.
    - Examples: len(), max(), min(), sum(), print().
2. **User-defined Functions** → created by programmers for specific needs.

Parameters vs Arguments

- **Parameters** → variables inside function definition.
- **Arguments** → values you pass when calling a function.

Example:

```
def square(x):   # x is parameter
    return x * x


print(square(5))  # 5 is argument
```

Default Parameters

Functions can have **default values** if no argument is provided.

Example:

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


greet()
greet("Alice")
```

Output:

Hello Student

Hello Alice

DevOps Real-Time Scenarios

⚠ Note: These show **where functions are useful in DevOps**. Not directly runnable, without infrastructure.

Scenario 1: Service Restart Function

```
def restart_service(service_name):
    print("Restarting", service_name, "...")
    # Real script would run a system command here
```

👉 Used in automation scripts to restart services like Jenkins, Docker.

Scenario 2: Log Checking Function

```
def check_logs(file):
    print("Scanning", file, "for errors...")
    # Real script would parse actual logs
```

Helps in CI/CD pipelines to check build logs.

Python Practice Tasks

1. **Greeting Function**
    - Write a function that prints "Hello, Python Learner!"
2. **Square Calculator**
    - Write a function that takes a number and returns its square.
3. **Simple Calculator Function**
    - Create a function calculator(a, b, op) where op can be +, -, *, /.
    - Perform the correct operation using conditionals.
4. **Maximum of Three Numbers**
    - Write a function that takes three numbers and returns the maximum.
5. **Factorial Function**
    - Write a function that takes a number n and returns n!.
6. **Palindrome Checker**
    - Write a function that checks whether a string is a palindrome (same forwards and backwards).

Modules & Packages

Introduction

As projects grow, putting all code in one file becomes messy.
Python solves this problem using:

- **Modules** → a single Python file containing reusable code (functions, variables, classes).
- **Packages** → a collection of related modules organized in directories.

👉 This allows developers to **reuse existing code** instead of rewriting it.

Why Modules & Packages Matter in DevOps?

In DevOps automation:

- Scripts often rely on **external libraries** for tasks like cloud management, monitoring, or networking.
- Instead of writing everything from scratch, engineers **import packages** (like os, subprocess, boto3 for AWS, paramiko for SSH).
- Modularization makes scripts **clean, reusable, and maintainable**.

Example (DevOps relevance):

- Importing a module to check system health.
- Using a cloud SDK package to deploy infrastructure (e.g., AWS boto3).
- Creating your own **custom module** for common DevOps tasks.

Using Modules in Python

1. Importing a Module

```
import math

print(math.sqrt(16))
```

Output:

```
4.0
```

2. Importing Specific Functions

```
from math import pi, sqrt

print(pi)

print(sqrt(25))
```

3. Aliasing Modules

```
import math as m
print(m.factorial(5))
```

Built-in vs Third-Party Modules

1. **Built-in modules** → already available with Python (e.g., os, sys, math, datetime).
2. **Third-party modules** → need to be installed separately using pip.

Example:

```
pip install requests
import requests
response = requests.get("https://example.com")
print(response.status_code)
```

Creating a Custom Module

Suppose you have a file mymodule.py:

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

Then in another file:

```
import mymodule
print(mymodule.greet("Alice"))
```

Packages

- A **package** is a folder containing multiple modules, plus a special __init__.py file.
- Example structure:

```
mypackage/
    __init__.py
    module1.py
    module2.py
```

Usage:

```
from mypackage import module1
```

DevOps Real-Time Scenarios

⚠ Not runnable directly on laptops without proper infrastructure.

- Using boto3 (AWS SDK) package to create an EC2 instance.
- Using paramiko package to SSH into servers.
- Using requests package to trigger webhooks in CI/CD.

Python Practice Tasks

1. **Math Module Task**
    - Import the math module and find: square root of 144, factorial of 6, and value of pi.
2. **Datetime Module Task**
    - Use the datetime module to print the current date and time.
3. **Random Module Task**
    - Import random and generate 5 random numbers between 1 and 100.
4. **Custom Module Task**
    - Create a file calculator.py with functions add, subtract, multiply, divide.
    - Import this module in another file and perform calculations.
5. **Third-Party Module Task (Optional, Internet Required)**
    - Install requests using pip.
    - Fetch and print the status code of "https://www.python.org".