Python Starter Pack

A Beginner's Guide to Python Programming

📖 Table of Contents

---

1. Introduction to Python

🔹 What is Python?

Python is a **high-level**, **interpreted**, and **general-purpose** programming language. It is widely used in web development, data science, artificial intelligence, automation, and more.

🔹 Why Learn Python?

✅ **Easy to Read & Learn** – Python's syntax is simple and beginner-friendly.
✅ **Versatile** – Used for Web Development, Machine Learning, AI, Automation, and more.
✅ **Huge Community Support** – Many libraries and frameworks are available.
✅ **Used by Major Companies** – Google, Facebook, NASA, and many more rely on Python.

Example Usage of Python:

- Automating tasks (e.g., sending emails, renaming files)

- Building websites (using Django, Flask)
- Data analysis & visualization (Pandas, Matplotlib)
- AI & Machine Learning (TensorFlow, PyTorch)

---

2. Setting Up Python

🔹 Installing Python

1. **Download Python** from .org.
2. **Run the Installer** and ensure **"Add Python to PATH"** is checked.

Verify Installation by running the command:

```
--version
```

3.

🔹 Choosing a Code Editor

- **VS Code (Recommended)** – Lightweight & supports Python extensions.
- **PyCharm** – Best for large Python projects.
- **Jupyter Notebook** – Ideal for Data Science & Machine Learning.

---

3. Basic Python Syntax

🔹 Printing Output

Python uses print() to display output.

```
print("Hello, World!")
```

🔹 Comments in Python

Comments help explain code but are ignored during execution.

```
# This is a single-line comment
"""
This is a
multi-line comment
"""
```

🔹 Indentation (No Braces {} in Python!)

Python uses **indentation** instead of {} to define code blocks.

```
if 5 > 2:
  print("5 is greater than 2") # Indented correctly
```

🚨 **Incorrect Indentation will cause an error!**

---

4. Variables and Data Types

🔹 Declaring Variables

Python is **dynamically typed**, meaning you don't need to declare variable types explicitly.

```
name = "Ramu"    # String
age = 25       # Integer
height = 5.7     # Float
is_student = True # Boolean
```

🔹 Checking Data Types

```
print(type(name)) # Output: <class 'str'>
```

🔹 Type Conversion

Convert one data type to another using functions like int(), float(), str().

```
age_str = str(age) # Converts integer to string
```

---

5. Control Flow (If-Else, Loops)

🔹 Definition

Control flow determines how a program executes based on conditions. Python uses if-else statements and loops (for, while) to control the execution flow.

✅ Key Points

- if, elif, and else are used to execute code conditionally.
- Loops (for, while) allow repeated execution of code blocks.
- break stops the loop early; continue skips an iteration.

🔹 If-Else Statements

An if-else statement checks a condition and executes the corresponding block of code.

```
x = 10
if x > 5:
  print("x is greater than 5")
elif x == 5:
  print("x is exactly 5")
else:
  print("x is less than 5")
```

🔹 Loops in Python

For Loop

A for loop iterates over a sequence, such as a list or range of numbers.

```
for i in range(3):  # Loops 3 times (0, 1, 2)
  print("Iteration:", i)
```

While Loop

A while loop executes as long as a condition is True.

```
count = 0
while count < 3:
  print(count)
  count += 1 # Increases count each iteration
```

Using break and continue

```
for i in range(5):
  if i == 3:
    break  # Stops loop when i is 3
  print(i)

for i in range(5):
  if i == 3:
    continue  # Skips 3 and continues the loop
  print(i)
```

---

6. Functions and Modules

🔹 Definition

A function is a reusable block of code that performs a specific task. Modules are external files containing functions that can be imported into programs.

✅ Key Points

- Functions **increase code reusability** and **make code organized**.
- def is used to define a function in Python.
- Functions can have parameters and return values.
- Python has built-in modules like math and random.

🔹 Creating a Function

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

print(greet("Alice")) # Output: Hello, Alice!
```

🔹 Default and Keyword Arguments

```
def add(x, y=10):  # Default value for y is 10
  return x + y

print(add(5))  # Output: 15
print(add(5, 20)) # Output: 25
```

🔹 Using Modules

Python has built-in and third-party modules.

```
import math
print(math.sqrt(25)) # Output: 5.0
```

```
import random
print(random.randint(1, 10)) # Generates a random number between 1 and 10
```

---

7. Lists, Tuples, and Dictionaries

🔹 Definition

- **Lists**: Ordered, mutable collection of elements.
- **Tuples**: Ordered, immutable collection.
- **Dictionaries**: Key-value pairs for quick lookups.

✅ Key Points

- Lists are dynamic and can be modified (add, remove, change elements).
- Tuples are immutable (cannot be changed after creation).
- Dictionaries allow fast lookups based on keys.

🔹 Lists (Mutable)

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange") # Adds "orange" to the list
print(fruits)
```

🔹 Tuples (Immutable)

```
coordinates = (10, 20) # Tuple with two elements
print(coordinates[0]) # Output: 10
```

🚨 **Tuples cannot be modified after creation!**

🔹 Dictionaries (Key-Value Pairs)

```
student = {"name": "Alice", "age": 25}
print(student["name"]) # Output: Alice
```

🔹 **Adding a new key-value pair:**

```
student["grade"] = "A"
print(student) # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

---

8. File Handling in Python

🔹 Definition

Python allows reading and writing files easily using built-in functions.

✅ Key Points

- Use `open(filename, mode)` to open a file.
- Modes:
  - `"r"` → Read mode (default).
  - `"w"` → Write mode (overwrites content).
  - `"a"` → Append mode (adds content).
  - `"r+"` → Read and write.
- Always close files after use or use `with` statement for automatic closing.

🔹 Reading a File

```
with open("file.txt", "r") as file:
  content = file.read()
  print(content)
```

🔹 Writing to a File

```
with open("file.txt", "w") as file:
  file.write("Hello, Python!")
```

🔹 Appending to a File

```
with open("file.txt", "a") as file:
  file.write("\nAppending new line")
```

---

9. Error Handling & Debugging

🔹 Definition

Error handling allows programs to handle runtime errors gracefully instead of crashing.

✅ Key Points

- try-except blocks catch and handle exceptions.
- Common errors: ZeroDivisionError, IndexError, KeyError, TypeError.
- Debugging techniques: print() statements, logging, and breakpoints.

🔹 Using Try-Except Blocks

```
try:
  result = 10 / 0
except ZeroDivisionError:
  print("Cannot divide by zero!")
```

🔹 Using finally to Execute Code Regardless of Errors

```
try:
  file = open("test.txt", "r")
except FileNotFoundError:
  print("File not found!")
finally:
  print("This code runs no matter what.")
```

🔹 Debugging with Print Statements

```
x = 5
print(f"x is: {x}") # Helps track variable values
```

---

10. Next Steps & Learning Resources

🔹 Definition

Once you have a good grasp of Python basics, the next step is to apply your skills and explore real-world projects.

✅ Key Points

- Explore Python projects like **web development, automation, data analysis, and AI.**
- Participate in coding challenges on **LeetCode, CodeWars, and HackerRank.**
- Join Python communities on **Reddit, Stack Overflow, and Discord.**