Chapter 8: Advanced Python – Revision and Functions

Introduction

In this chapter, we revisit the core concepts of Python that you have learned earlier and delve deeper into advanced function-related concepts. Functions are the building blocks of modular programming in Python. By mastering them, we can write reusable, organized, and clean code — an essential skill in Artificial Intelligence and real-world applications.

Python’s simple syntax and powerful features allow us to build logic efficiently. Whether you're working with AI models, data preprocessing, or automation, understanding how functions work is vital.

8.1 Revision of Python Basics

Before diving into functions, let us briefly revise the following foundational Python topics:

8.1.1 Python Data Types

- **Numbers:** `int`, `float`, `complex`
- **Strings:** Immutable sequences of characters, created using quotes (`'Hello'` or `"World"`)
- **Booleans:** `True` and `False`
- **Lists:** Ordered, mutable collection: `[1, 2, 3]`
- **Tuples:** Ordered, immutable collection: `(1, 2, 3)`
- **Dictionaries:** Key-value pairs: `{'name': 'AI', 'year': 2025}`

8.1.2 Control Structures

- **If-else statements:** Used for decision-making.

```
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

- **Loops:** `for` and `while` loops for iteration.

8.1.3 Python Operators

- Arithmetic: `+`, `-`, `*`, `/`, `//`, `%`
- Logical: `and`, `or`, `not`
- Comparison: `==`, `!=`, `<`, `>`, `<=`, `>=`

8.2 Functions in Python

Functions are a block of organized, reusable code that is used to perform a single, related action.

8.2.1 Types of Functions

- **Built-in Functions:** Already available in Python (`print()`, `len()`, `type()`, `range()`, etc.)
- **User-defined Functions:** Defined by the programmer using `def`.

8.2.2 Defining a Function

```
def greet():
    print("Hello, AI World!")
```

8.2.3 Calling a Function

```
greet()  # Output: Hello, AI World!
```

8.2.4 Function with Parameters

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

8.2.5 Function with Return Value

```
result = add(3, 4)
print(result)  # Output: 7
```

8.3 Parameters and Arguments

8.3.1 Positional Arguments

Arguments are matched by position.

```
def student(name, age):
    print(name, age)

student("Alice", 17)
```

8.3.2 Keyword Arguments

Arguments are passed with the parameter name.

```
student(age=17, name="Alice")
```

8.3.3 Default Arguments

Provide a default value.

```
def student(name, age=18):
    print(name, age)
```

```
student("Bob")  # Output: Bob 18
```

8.3.4 Variable-Length Arguments

- **Arbitrary Positional Arguments `*args`:**

```
def total_marks(*marks):
    return sum(marks)

total = total_marks(90, 85, 75)
```

- **Arbitrary Keyword Arguments `**kwargs`:**

```
def display_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key} : {value}")

display_info(name="AI", year=2025)
```

8.4 Scope and Lifetime of Variables

8.4.1 Local vs Global Variables

- **Local:** Declared inside a function and accessible only there.
- **Global:** Declared outside all functions and accessible everywhere.

```
x = 10  # Global

def show():
    x = 5  # Local
    print(x)

show()  # Output: 5
print(x)  # Output: 10
```

8.4.2 The `global` Keyword

To modify a global variable inside a function.

```
x = 10

def modify():
    global x
    x = 20

modify()
print(x)  # Output: 20
```

8.5 Lambda Functions

8.5.1 What is a Lambda Function?

- Anonymous, single-expression functions.
- Syntax: `lambda arguments: expression`

```
square = lambda x: x**2
print(square(4))  # Output: 16
```

Useful in:

- Sorting
- Mapping
- Filtering

Example:

```
nums = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, nums))
```

8.6 Recursion in Python

A function calling itself.

Example: Factorial using Recursion

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5))  # Output: 120
```

Be cautious: Recursion can lead to memory overflow if not handled correctly.

8.7 Docstrings and Comments

8.7.1 Single-Line Comment

```
# This is a comment
```

8.7.2 Multi-Line Comment / Docstring

```
def greet():
    """This function greets the user"""
    print("Hello!")
```

Use `help(greet)` to read the docstring.

8.8 Advantages of Using Functions

- **Modularity**: Split code into smaller chunks.
- **Reusability**: Write once, use multiple times.
- **Maintainability**: Easier to debug and maintain.
- **Readability**: Clear structure.

Summary

In this chapter, we revised Python basics and explored the concept of functions in detail. You learned about user-defined functions, parameters and arguments, variable scope, lambda functions, and recursion. Functions play a crucial role in organizing code effectively, especially in AI projects where complex logic is often split into smaller, manageable parts. With a strong foundation in functions, you’re now equipped to build more advanced and modular Python applications.
