Python Fundamentals

* Introduction to Python and its Features (simple, high-level, interpreted language).
* Python is a powerful, versatile, and widely-used programming language known for its simplicity and readability. Created by **Guido van Rossum** in 1991, Python emphasizes code readability with its clean syntax and indentation structure. It is designed to be beginner-friendly while being robust enough for advanced applications such as web development, data analysis, artificial intelligence, and more.

**Features of Python:-**

* Easy to Learn and Use
* Interpreted Language
* High-Level Language
* Portability
* Extensive Standard Library
* Dynamically Typed
* Object-Oriented and Functional Programming
* Rich Ecosystem and Frameworks
* Community Support
* Scalability
* Integration Capabilities
* Readable and Maintainable Code
* Open Source
* History and evolution of Python.
* Python was created by **Guido van Rossum** in the late 1980s and officially released in **1991**. It was designed as an easy-to-read and versatile programming language, drawing inspiration from languages like **ABC, Modula-3, C, C++, and Algol-68**. The goal was to make programming fun and accessible while being powerful for real-world applications.
* Advantages of using Python over other programming languages.
* Python stands out among programming languages due to its simplicity, versatility, and vast ecosystem. Here are some key advantages:
* Ease of Learning and Use
* Versatility Across Domains
* Large Standard Library
* Cross-Platform Compatibility
* Extensive Ecosystem and Community Support
* Faster Development Cycle
* Strong Support for Integration
* Suitability for Rapid Prototyping
* Thriving in Emerging Fields
* Cost-Efficiency
* Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).
* Step 1: Download Python
* Visit the official Python website: <https://www.python.org/downloads/>.
* Download the latest stable version for your operating system (Windows, macOS, or Linux).
* Step 2: Install Python
* Run the downloaded installer.
* **Important**: Check the box **"Add Python to PATH"** before proceeding.
* Follow the installation prompts and choose default settings unless you have specific needs.
* Step 3: Verify Installation
* Open your terminal or command prompt and type:
* Python –version

2. Programming Style

* Understanding Python’s PEP 8 guidelines.
* PEP 8 is the **Style Guide for Python Code** and is an essential document that provides conventions for writing clean and readable Python code. It is widely followed by the Python community to ensure uniformity across Python projects.
* Indentation, comments, and naming conventions in Python.
* Indentation: Proper indentation is critical in Python because it defines the structure of your code.
* Use 4 spaces per indentation level
* **Nested Blocks**: Indent each level consistently with 4 spaces.
* **Hanging Indentation**: For long lines, align with the first argument, or use an extra indentation level.
* Comments: Comments are essential for explaining the intent of the code and helping others (or your future self!) understands it. PEP 8 outlines best practices for both **inline** and **block comments.**
* Block Comments
* Inline Comments
* Docstrings
* Naming Conventions: PEP 8 encourages naming conventions to maintain readability and clarity
* Variables and Functions
* Constants
* Classes
* Private Members
* Avoid Ambiguity
* Module and Package Names
* Writing readable and maintainable code.
* Writing **readable** and **maintainable** code is essential for collaboration, debugging, and long-term project sustainability. Here are principles and practices that align with Pythonic standards and general coding best practices.

Follow PEP 8 Guidelines, Write Self-Documenting Code, Use Comments Wisely, Organize Code Logically, Keep Functions Short and Focused, Avoid Magic Numbers, Handle Errors Gracefully, Modularize and Reuse Code, Write Unit Tests, Use Version Control, Optimize for Performance, Document Your Code.

3. Core Python Concepts

* Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.
* Integers (int) :- Represents whole numbers (positive, negative, or zero), No limit on the size of integers (limited by system memory).
* Float (float):- Represents numbers with a decimal point or in exponential notation, Used for real numbers.
* String (str) :- Represents a sequence of characters, Defined using single ('), double (") quotes, or triple quotes (''' or """) for multi-line strings.
* List (list) :- An ordered, mutable (modifiable) collection of items, Can contain elements of different data types.
* Tuple (tuple):- An ordered, immutable (unchangeable) collection of items, Often used to represent fixed data.
* Dictionary (dict):- An unordered collection of key-value pairs, Keys are unique and immutable (strings, numbers, or tuples); values can be any data type.
* Set (set):- An unordered collection of unique items, Used to store distinct values and perform mathematical set operations.
* Python variables and memory allocation.
* **Python variables and memory allocation** work is key to writing efficient and bug-free code. Python handles memory allocation and management dynamically, so the programmer does not need to explicitly allocate or free memory.

1. Variables and Objects

2. Memory Allocation

3. Reference Counting

4. Garbage Collection

5. Mutable vs Immutable Objects

6. Variable Reassignment

7. id() and Object Identity

8. Small Integer and String Interning

9. Shallow Copy vs Deep Copy

10. Example: Memory Management in Action

4. Conditional Statements

* Introduction to conditional statements: if, else, elif.
* Conditional statements are a fundamental concept in programming that allow a program to make decisions and execute different actions based on specific conditions. Here’s an introduction to the basic conditional statements: if, else, and elif.

1. The if Statement:-

The if statement evaluates a condition and executes the block of code only if the condition is **true**.

**Syntax:**

if condition:

# Code to execute if the condition is true

2. The else Statement:-

The else statement provides an alternative block of code that runs if the if condition is **false**.

**Syntax:**

if condition:

# Code to execute if the condition is true

else:

# Code to execute if the condition is false

3. The elif Statement (Else If):- The elif statement checks another condition if the previous if condition is **false**. It allows multiple conditions to be evaluated sequentially.

**Syntax:**

if condition1:

# Code to execute if condition1 is true

elif condition2:

# Code to execute if condition2 is true

else:

# Code to execute if all conditions are false

* Nested if-else conditions.
* A **nested if-else** condition refers to an if-else block placed inside another if or else block. This allows for more complex decision-making by evaluating multiple conditions in a structured manner.

Syntax:

if condition1:

if condition2:

# Code to execute if condition1 and condition2 are true

else:

# Code to execute if condition1 is true and condition2 is false

else:

# Code to execute if condition1 is false

5. Looping (For, While)

* Introduction to for and while loops.

1. For Loop: The **for loop** is used to iterate over a sequence (like a list, tuple, string, or range) or other iterable objects. It executes a block of code for each item in the sequence.

**Syntax**:

for variable in sequence:

# Code block to execute for each item

2. While Loop: The **while loop** runs as long as a specified condition is true. It is typically used when the number of iterations is not known in advance.

**Syntax**:

while condition:

# Code block to execute

* How loops work in Python.
* **loops** allow you to repeatedly execute a block of code until a specific condition is met or a sequence is exhausted. This is achieved using two main types of loops: **for loops** and **while loops**.
* Using loops with collections (lists, tuples, etc.).
* Using loops with collections like **lists**, **tuples**, **sets**, and **dictionaries** is a powerful way to process data in Python. Each type of collection has unique characteristics, and Python's loops provide an intuitive way to work with them.

6. Generators and Iterators

* Understanding how generators work in Python.
* **generators** are a way to create iterators in a more concise and memory-efficient manner. They allow you to iterate through a sequence of values without the need to store the entire sequence in memory.
* Difference between yield and return.

| **Aspect** | **yield** | **return** |
| --- | --- | --- |
| **Usage** | Used to produce a value in a generator function and pause its execution. | Used to return a value and terminate the function. |
| **Type of Function** | Only valid in a generator function. | Used in regular functions. |

* Understanding iterators and creating custom iterators.
* an **iterator** is an object that enables you to traverse through a sequence of values, one at a time, without the need for an index. Iterators are central to Python's iterable protocol, which allows for looping over objects like lists, tuples, and dictionaries.

7. Functions and Methods

* Defining and calling functions in Python.
* Functions in Python are blocks of reusable code designed to perform a specific task. They make programs more modular and reduce code repetition.

Defining a Function: To define a function, use the def keyword followed by the function name and parentheses ().

Calling a Function: To execute a function, use its name followed by parentheses ()

* Function arguments (positional, keyword, default).
* **function arguments** allow you to pass information into a function. There are several types of arguments: **positional**, **keyword**, and **default**.

Positional Arguments:- Positional arguments are passed to a function in the same order as the parameters are defined. They are the simplest form of arguments.

Keyword Arguments:- Keyword arguments allow you to specify the argument names explicitly when calling the function. This makes the function call more readable and eliminates the need to remember the order of arguments.

Default Arguments:- Default arguments allow you to provide default values for parameters. If the caller does not provide a value for that parameter, the default value is used.

* Scope of variables in Python.
* the **scope** of a variable determines where in the code the variable is accessible. Variables can be classified based on where they are declared and how they are used.

Types of Scope:

1. Local Scope
2. Global Scope
3. Enclosing Scope (Nonlocal)
4. Built-in Scope

These follow the **LEGB rule**: **Local → Enclosing → Global → Built-in**

* Built-in methods for strings, lists, etc
* Python provides numerous built-in methods for commonly used data types like **strings**, **lists**, **dictionaries**, **tuples**, and more. These methods simplify data manipulation and are integral to Python programming.

### ****Built-in Methods for Strings****

Strings are immutable sequences of characters, and Python provides many methods for working with them.

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| lower() | Converts string to lowercase. | "Hello".lower() → 'hello' |
| upper() | Converts string to uppercase. | "Hello".upper() → 'HELLO' |
| strip() | Removes leading/trailing whitespace or specified characters. | " Hello ".strip() → 'Hello' |
| split() | Splits string into a list based on a delimiter. | "a,b,c".split(",") → ['a', 'b', 'c'] |
| join() | Joins elements of a list into a string using a delimiter. | " ".join(['a', 'b', 'c']) → 'a b c' |
| find() | Returns index of first occurrence of substring, or -1. | "hello".find("e") → 1 |
| replace() | Replaces occurrences of a substring with another. | "hello".replace("e", "a") → 'hallo' |
| startswith() | Checks if string starts with a specified substring. | "hello".startswith("he") → True |
| endswith() | Checks if string ends with a specified substring. | "hello".endswith("lo") → True |
| isalpha() | Checks if all characters are alphabetic. | "Hello".isalpha() → True |
| isdigit() | Checks if all characters are digits. | "123".isdigit() → True |

### ****Built-in Methods for Lists****

Lists are mutable sequences used to store collections of items.

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| append() | Adds an item to the end of the list. | my\_list.append(5) |
| extend() | Extends the list by appending elements from another list. | my\_list.extend([6, 7]) |
| insert() | Inserts an item at a specified index. | my\_list.insert(1, 10) |
| remove() | Removes the first occurrence of a specified value. | my\_list.remove(10) |
| pop() | Removes and returns the item at a specified index. | my\_list.pop(0) |
| index() | Returns the index of the first occurrence of a value. | my\_list.index(6) |
| count() | Counts the occurrences of a specified value. | my\_list.count(7) |
| reverse() | Reverses the elements of the list in place. | my\_list.reverse() |
| sort() | Sorts the list in ascending order (in-place). | my\_list.sort() |
| clear() | Removes all elements from the list. | my\_list.clear() |

8. Control Statements (Break, Continue, Pass)

* Understanding the role of break, continue, and pass in Python loops
* The break, continue, and pass statements play important roles in controlling the flow of loops in Python. They allow you to modify the default behavior of loops (like for and while) based on certain conditions.

### ****1.**** break ****Statement****

The break statement is used to **exit a loop prematurely** (before it naturally finishes). Once the break statement is executed, the loop is terminated, and the control is passed to the next statement after the loop.

### ****2.**** continue ****Statement****

The continue statement is used to **skip the current iteration** of the loop and proceed to the next iteration. It does not terminate the loop but skips the remaining code inside the loop for the current iteration.

### ****3.**** pass ****Statement****

The pass statement is a **null operation**. It does nothing and serves as a placeholder. It’s typically used in situations where you need to have an empty code block, such as in defining functions or handling certain loop conditions where no action is needed.

9. String Manipulation

* Understanding how to access and manipulate strings.

strings are sequences of characters enclosed in either single (') or double (") quotes. Strings are immutable, meaning they cannot be changed after they are created. However, you can access, manipulate, and create new strings using various techniques.

### ****1. Accessing Strings****

You can access individual characters of a string using **indexing** and **slicing**.

#### ****Indexing****

Each character in a string has a corresponding index, starting from 0 for the first character. You can use these indices to access specific characters.

**Positive indexing**: Starts from 0 and goes to the end of the string.

**Negative indexing**: Starts from -1 for the last character and goes to the beginning of the string.

### ****2. Slicing Strings****

Slicing allows you to extract a part of a string.

### ****3. String Concatenation and Repetition****

* **Concatenation**: Joining two or more strings using the + operator.
* **Repetition**: Repeating a string using the \* operator.

10. Advanced Python (map(), reduce(), filter(), Closures and Decorators)

* How functional programming works in Python.
* Functional programming in Python is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing state or mutable data. Python supports functional programming features, even though it's primarily an object-oriented and imperative language. Functional programming focuses on using functions as first-class citizens, which means they can be passed around as arguments, returned from other functions, and assigned to variables.

### Key Concepts in Functional Programming

**First-Class Functions**: In functional programming, functions are first-class citizens. This means that you can:

* + Assign functions to variables.
  + Pass functions as arguments to other functions.
  + Return functions from other functions.

**Higher-Order Functions**: These are functions that take other functions as arguments or return them. This allows for more abstract and reusable code.

* Using map(), reduce(), and filter() functions for processing data.

These built-in functions are widely used in functional programming for processing collections (like lists or tuples) in a functional way.

**map()**: Applies a function to every item of an iterable (like a list) and returns a map object (an iterator).

**filter()**: Filters elements of an iterable based on a function that returns True or False

**reduce()**: From the functools module, reduce() applies a binary function (a function that takes two arguments) cumulatively to the items of an iterable, from left to right.

* Introduction to closures and decorators.

A **closure** is a function that "remembers" the environment (the variables) in which it was created. This happens when a nested function refers to a variable from its enclosing scope. Even after the outer function has finished executing, the nested function can still access those variables.

#### Key Characteristics of Closures:

1. **Nested functions**: A closure involves a function defined inside another function.
2. **Free variables**: The inner function refers to variables in the outer function’s scope.
3. **Function object**: The inner function is returned by the outer function, and the returned function has access to the outer function's variables even after the outer function's execution has finished.

### ****Decorators in Python****

A **decorator** is a function that modifies the behavior of another function or class. In Python, decorators are often used to add functionality to functions or methods in a reusable and modular way.

#### How Decorators Work:

1. **A decorator is a function**: A decorator takes a function as input and returns a new function that enhances the original function's behavior.
2. **Wrapping functions**: The decorator can execute code before and after calling the original function.