Module 1 –
Introduction to Python

Intro to Python

➢ Python is a general-purpose, dynamic, high-level, and interpreted programming language.

➢ It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

➢ Python is an easy-to-learn yet powerful and versatile scripting language, which makes it attractive for Application Development.

Intro to Python

➢With its interpreted nature, Python's syntax and dynamic typing make it an ideal language for scripting and rapid application development.

➢Python makes development and debugging fast because no compilation step is included in Python development, and the edit-test-debug cycle is very fast.

➢We don't need to use data types to declare variable because it is dynamically typed, so we can write a=10 to assign an integer value in an integer variable.

Journey of Python

Origin:

◦ Python was created by **Guido van Rossum** in the late 1980s.

◦ The first version, Python 0.9.0, was released in February 1991.

Guiding Principles:

◦ **Guido van Rossum** aimed to create a language that was easy to read and write, emphasizing code readability and simplicity.

◦ He was influenced by ABC, a general-purpose programming language.

Journey of Python

Python 2 vs. Python 3:

◦ Python 2.x series was the prevalent version for many years.

◦ Python 3, a major revision, was released in December 2008, introducing backward-incompatible changes to the language.

◦ Python 2 was officially sunset on January 1, 2020, and developers are encouraged to migrate to Python 3.

Journey of Python

Community Growth:

◦ Python's popularity grew steadily over the years, thanks to its simplicity and versatility.

◦ It gained traction in various fields such as web development, scientific computing, data analysis, artificial intelligence, and more.

Journey of Python

Major Releases and Enhancements:

◦ Python continues to evolve with regular releases, introducing new features, improvements, and optimizations.

◦ Notable releases include Python 2.7 (the last release in the 2.x series), Python 3.0 (the initial release of Python 3), and subsequent versions such as Python 3.6, 3.7, 3.8 to 3.12 till date with 3.13 ready to release.

Journey of Python

Adoption by Tech Giants:

◦ Python gained significant traction in the tech industry, being embraced by major companies like Google, Facebook, Amazon, and Dropbox for various purposes including web development, data analysis, and automation.

Current Status and Future:

◦ Python has become one of the most popular programming languages worldwide, with a large and active community of developers and its future looks promising with ongoing development efforts focused on performance improvements, language enhancements, and ecosystem expansion.

Journey of Python

Feature	Compiler	Interpreter
Process	Converts entire source code into machine code at once	Processes and executes code line by line
Output	Generates executable file or intermediate code (e.g., bytecode)	Executes code directly without generating intermediate files
Efficiency	Generally produces faster execution as machine code is optimized	May have slower execution as code is interpreted line by line
Error Handling	Identifies errors before execution (compile-time errors)	Identifies errors during execution (runtime errors)
Memory Usage	Typically requires more memory during compilation	Typically requires less memory as it executes code directly
Portability	Compiled programs may not be directly portable between different architectures or platforms	Interpreted programs can be more easily portable between different platforms
Debugging	Debugging may be more complex as it involves examining machine code or intermediate representations	Debugging is typically easier as interpreters can provide detailed error messages and support interactive debugging
Examples	C, C++, Java	Python, JavaScript, Ruby, PHP

How Python is different from Java

Syntax:

- Python emphasizes readability with its concise and clean syntax, utilizing indentation to define code blocks.
- Java syntax is more verbose, requiring explicit declaration of data types, and uses braces to define code blocks.

Typing:

- Python is dynamically typed, where variable types are determined at runtime, providing flexibility but potentially leading to runtime errors.
- Java is statically typed, requiring explicit declaration of variable types at compile time, providing type safety but potentially leading to more verbose code.

How Python is different from Java

Interpretation vs. Compilation:

◦ Python is interpreted, meaning code is executed line by line without prior compilation, offering faster development cycles but potentially slower runtime performance.

◦ Java is compiled to bytecode, which is then executed by the Java Virtual Machine (JVM), offering better runtime performance but requiring compilation before execution.

Platform Independence:

◦ Python code is typically platform-independent, running on various operating systems without modification.

◦ Java is known for its "write once, run anywhere" capability, with compiled bytecode being able to run on any device with a Java Virtual Machine (JVM), offering greater portability.

How Python is different from Java

Programming Paradigms:

◦ Python supports multiple programming paradigms including procedural, object-oriented, and functional programming.

◦ Java is primarily object-oriented, with support for procedural programming as well.

Memory Management:

◦ Python uses automatic memory management and garbage collection, handling memory allocation and deallocation automatically.

◦ Java also utilizes automatic memory management and garbage collection, but typically provides better control over memory allocation and deallocation through features like explicit memory management.

Features of Python

Python is a versatile and high-level programming language known for its simplicity, readability, and flexibility.

Here are the key features of Python explained in detail:

Readability:

◦ Python's syntax is designed to be readable and clean.
◦ The use of indentation instead of braces for block delimiters promotes code readability.
◦ The language enforces a coding style that emphasizes clarity, making it easier for developers to understand and maintain code.

Features of Python

Simplicity:

◦ Python's design philosophy prioritizes simplicity and ease of use.

◦ It aims to have a clear and straightforward syntax that allows developers to express concepts in fewer lines of code compared to languages like C++ or Java.

◦ The simplicity of Python contributes to its suitability for beginners and its rapid development capabilities.

Features of Python

Versatility:

◦ Python is a general-purpose programming language, meaning it can be used for a wide range of applications and domains.

◦ It is used in web development, data science, artificial intelligence, scientific computing, automation, scripting, and more.

Features of Python

Interpreted and Interactive:

◦ Python is an interpreted language, meaning that code can be executed line by line without the need for a separate compilation step.

◦ It supports an interactive mode, where developers can test and run code snippets interactively, making it useful for learning and experimentation.

Features of Python

High-Level Language:

◦ Python is a high-level language, which means that it abstracts many low-level details and provides constructs that allow developers to express complex operations more easily.

◦ This high-level nature simplifies the development process and allows for faster prototyping.

Features of Python

Object-Oriented:

◦ Python supports object-oriented programming (OOP) principles, allowing developers to structure their code using classes and objects.

◦ OOP features in Python include encapsulation, inheritance, and polymorphism.

Features of Python

Dynamically Typed:

◦ Python is dynamically typed, meaning that the type of a variable is determined at runtime. Developers don't need to specify variable types explicitly.

◦ This flexibility allows for faster development but may require careful consideration of variable types during runtime.

Features of Python

Expressive Language:

◦ Python allows developers to express concepts in fewer lines of code, enhancing readability and reducing the amount of boilerplate code.

◦ Features like list comprehensions, lambda functions, and powerful built-in functions contribute to the expressiveness of the language.

Features of Python

Extensive Standard Library:

◦ Python comes with a comprehensive standard library that provides modules and packages for various tasks, from file handling to web development.

◦ The standard library reduces the need for developers to write code from scratch for common functionalities.

Features of Python

Community and Ecosystem:

◦ Python has a large and active community of developers who contribute to its growth and improvement.

◦ Python's development process involves the input and collaboration of the community through the Python Enhancement Proposal (PEP) process.

◦ Community involvement ensures that the language evolves to meet the needs of a diverse range of users.

Features of Python

Cross-Platform Compatibility:

◦ Python code can run on different operating systems without modification, enhancing its cross-platform compatibility.

◦ This feature makes it easier to develop and deploy Python applications across various environments.

Features of Python

Open Source :

◦ Python is an open-source language, allowing developers to view, modify, and distribute the source code freely.

◦ This openness fosters collaboration and innovation within the Python community.

Features of Python

Support for Integration:

◦ Python can easily integrate with other languages and technologies. It supports interfaces to many system calls and libraries, as well as tools for integrating C/C++ code.

Security:

◦ Python includes security features, such as exception handling and standard libraries for secure communications, which contribute to building robust and secure applications.

Use Cases of Python

Web Development:

◦ Frameworks like Django and Flask are popular for building scalable web applications.
◦ Python's simplicity and extensive libraries streamline web development processes.

Data Science and Machine Learning:

◦ Python is a dominant language in data science and machine learning due to libraries like NumPy, pandas, and TensorFlow.
◦ Its simplicity and readability facilitate data manipulation, analysis, and model building.

Use Cases of Python

Scientific Computing:

- Python is widely used in scientific computing for simulations, data visualization, and analysis.
- Libraries like SciPy, matplotlib, and Jupyter make complex scientific tasks easier.

Artificial Intelligence and Natural Language Processing (NLP):

- Python's versatility makes it suitable for AI and NLP applications.
- Libraries like NLTK, and transformers are widely used in these domains.

Use Cases of Python

Automation and Scripting:

◦ Python's concise syntax and cross-platform compatibility make it ideal for automation and scripting tasks.
◦ It's used for tasks like system administration, network programming, and task automation.

Game Development:

◦ Python, with libraries like Pygame and Panda3D, is used in game development, particularly for rapid prototyping and scripting.
◦ It's also used in game AI development and tool creation.

Use Cases of Python

Education:

◦ Python's simplicity and readability make it a popular choice for teaching programming concepts to beginners.

◦ Many educational institutions use Python as the primary language for introductory programming courses.

DevOps:

◦ Python is commonly used in DevOps for tasks like infrastructure management, configuration management (with tools like Ansible), and deployment automation.

◦ Its wide range of libraries simplifies DevOps workflows.

Use Cases of Python

Desktop GUI Applications:

◦ Python, with libraries like Tkinter, PyQt, and PyGTK, is used for developing desktop GUI applications.

◦ Its simplicity and cross-platform compatibility make it suitable for building such applications.

Python's versatility, simplicity, extensive libraries, and strong community support contribute to its popularity across a wide range of industries and domains.

Python Pre-requisites

Python is known for its simplicity and ease of learning, making it an excellent language for beginners.

Here are some prerequisites and recommendations for getting started with Python:

1. **None or Minimal Programming Experience:**
   - ➤ Python is often chosen as a first programming language for beginners.
   - ➤ It is designed to be beginner-friendly, and you can start learning Python without any prior programming experience.

Python Pre-requisites

2. Basic Computer Skills:

◦ A basic understanding of how to use a computer, navigate files and directories, and perform simple tasks is helpful.

◦ Familiarity with text editors or integrated development environments (IDEs) can be beneficial.

3. Installation of Python:

◦ To start programming in Python, you need to have Python installed on your computer.

◦ You can download the latest version of Python from the official Python website (https://www.python.org/).

Python Pre-requisites

4. Text Editor or IDE:

◦ While Python code can be written in any text editor, using a dedicated Integrated Development Environment (IDE) can enhance your development experience.

◦ Popular Python IDEs include PyCharm, Visual Studio Code, and Jupyter Notebook.

5. Understanding of Basic Programming Concepts:

◦ It's helpful to have a basic understanding of fundamental programming concepts like variables, data types, loops, and conditionals.

◦ However, Python's syntax is designed to be straightforward, making it accessible for beginners to pick up these concepts quickly.

Python Pre-requisites

6. **Internet Connection:**
- ◦ Having an internet connection is useful for accessing documentation, tutorials, and community forums where you can seek help and find resources.

7. **Learning Resources:**
- ◦ Gather learning resources such as tutorials, online courses, and documentation.
- ◦ Python has extensive documentation available on its official website, including the Python Standard Library documentation.

Python Pre-requisites

8. Curiosity and Persistence:

◦ Python, like any programming language, requires a degree of curiosity and persistence.

◦ Be prepared to experiment, make mistakes, and learn from them.

◦ Problem-solving is a crucial aspect of programming.

9. Optional: Basic Command-Line Knowledge:

◦ While not strictly necessary, having basic command-line knowledge can be beneficial.

◦ Python can be run and scripts can be executed from the command line or terminal.

Environment Setup for Python

Setting up the Python environment involves installing Python itself, a text editor or integrated development environment (IDE), and configuring your system for Python development.

Here are the steps for setting up a Python environment:

1. Install Python:

➤ Visit the official Python website at https://www.python.org/ and go to the "Downloads" section.

➤ Download the latest stable version of Python for your operating system (Windows, macOS, or Linux).

➤ Run the installer and make sure to check the box that says "Add Python to PATH" during installation.

Environment Setup for Python

2. **Verify Installation:**

Open a command prompt or terminal and type *'python --version'* or *'python -V'*. This should display the installed Python version. You can also use '**python**' to enter the interactive Python shell.

3. **Choose a Text Editor or IDE:**

While Python code can be written in any text editor, using a dedicated IDE can enhance your development experience. Some popular Python IDEs include:

◦ **PyCharm:** A powerful IDE with a community edition that is free to use.

◦ **Visual Studio Code:** A lightweight, open-source code editor with excellent Python support.

◦ **Jupyter Notebook:** An interactive web-based notebook environment, especially popular for data science.

Environment Setup for Python

4. Install and Configure the Chosen IDE or Text Editor:

➢If you choose an IDE like PyCharm or Visual Studio Code, follow the installation instructions provided on their respective websites.

➢Install Python extensions or plugins for your chosen IDE or text editor to enable Python-specific features.

5. Package Manager (pip):

➢Python comes with a package manager called '**pip**' that is used to install, upgrade, and manage Python packages.

➢Verify that '**pip**' is installed by running **'pip --version**'. If it's not installed, you can install it separately.

First Program in Python

Write "Hello-World" as first program in Python

Program Structure in Python

Explain Program Structure step-by-step of your program.

Module 2 – Python Basics

Python Basic Data-Types

Python supports various basic data types that are fundamental for programming and data manipulation.

Here are some of the main basic data types in Python:

1. **Integer (`int`):**

Represents whole numbers without any decimal points.

Example: **`x = 5`**

2. **Float ('float'):**

Represents numbers with decimal points or in exponential form.

Example: **`y = 3.14`**

Python Basic Data-Types

3. String (`str`):

Represents a sequence of characters enclosed in single (**'**) or double (**"**) quotes.

Example: `**name = 'Python'**`

4. Boolean ('bool'):

Represents the binary values '**True**' or '**False**'.

Often used in logical expressions and control flow.

Example: `**is_python_fun = True**`

Python Basic Data-Types

5. **List (`list`):**

Represents an ordered, mutable collection of elements.

Elements can be of different data types.

Example: `**numbers = [1, 2, 3, 4]**`

6. **Tuple (`tuple`):**

Represents an ordered, immutable collection of elements.

Similar to lists, but elements cannot be modified after creation.

Example: `**coordinates = (3, 5)**`

Python Basic Data-Types

7. **Set (`set`):**

Represents an unordered collection of unique elements.

Used for mathematical set operations.

Example: `unique_numbers = {1, 2, 3, 4}`

8. **Dictionary (`dict`):**

Represents a collection of key-value pairs.

Keys must be unique, and values can be of any data type.

Example: `student = {'name': 'Alice', 'age': 20, 'grade': 'A'}`

Python Basic Data-Types

9. **NoneType (`None`):**

Represents the absence of a value or a null value.

Commonly used to indicate that a variable has no assigned value.

Example: `**result = None**`

These basic data types serve as the building blocks for more complex data structures and are widely used in Python programming.

It's important to note that Python is dynamically typed, meaning you don't need to explicitly declare the data type of a variable; it is inferred at runtime.

Python Variables

In Python, a variable is a named location in memory used to store data.

Unlike some other programming languages, you don't need to explicitly declare the data type of a variable;

Python dynamically infers the type during runtime.

Here are the key aspects of Python variables:

Python Variables

1. Variable Assignment:

Assigning a value to a variable is done using the `=` operator
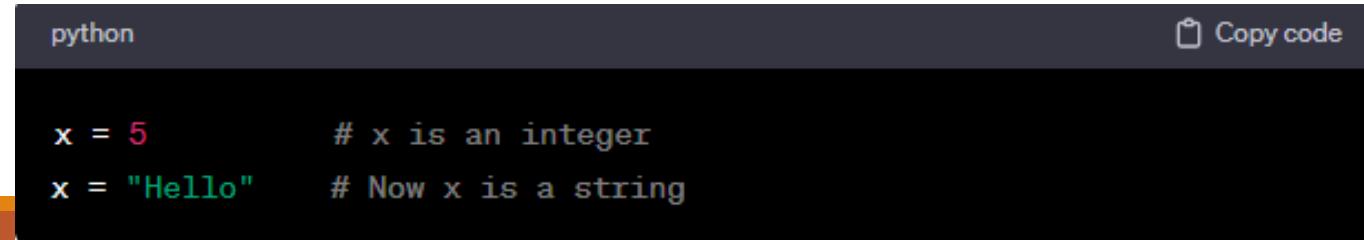
```python
age = 25
name = "John"
```

2. Dynamic Typing:

Python is dynamically typed, which means you can reassign a variable to different types.

```python
x = 5          # x is an integer
x = "Hello"    # Now x is a string
```

Python Variables

3. Variable Naming Rules:

Variable names can contain letters, numbers, and underscores.

They cannot start with a number.

Python is case-sensitive, so `variable` and `Variable` are different.

4. Data Types:

Variables can hold various data types, including integers, floats, strings, booleans, lists, tuples, sets, dictionaries, and more.

Python Variables

```python
age = 25          # Integer
height = 1.75     # Float
name = "John"     # String
is_student = True # Boolean
```

5. Reassignment:

You can reassign a variable to a new value at any time.

```python
x = 5
x = x + 1  # Reassigning x to a new value
```

Python Variables

6. Memory Allocation:

When a variable is assigned, Python allocates memory to store the data. The variable then references this memory location.

7. Deleting Variables:

You can use the `del` statement to delete a variable and free up the associated memory.

```python
x = 10
del x  # Deletes the variable x
```

Python Variables

8. **Multiple Assignment:**

You can assign values to multiple variables in a single line.

```python
a, b, c = 1, 2, 3
```

9. **Swapping Values:**

Python allows easy swapping of variable values without using a temporary variable.

```python
a, b = 5, 10
a, b = b, a   # Swapping values
```

Python Variables

10. **Global and Local Variables:**

Variables can be defined within a specific scope. Variables defined inside a function are local, while those defined outside are global.

```python
global_variable = 10


def example_function():
    local_variable = 5
    print(global_variable)  # Accessing global variable within the func
```

Python Variables

11. NoneType:

Python has a special `**None**` value representing the absence of a value. It can be assigned to variables to indicate no initial value.

```python
result = None
```

Python variables are versatile and play a crucial role in storing and manipulating data within programs.

Understanding their dynamic typing and flexibility is fundamental to effective Python programming.

Python Keywords

In Python, keywords are reserved words that have a specific meaning and cannot be used as identifiers (variable names, function names, etc.).

These keywords are an integral part of the language's syntax and define the structure and rules of Python programs.

Here is a detailed explanation of some important Python keywords:

Python Keywords

1. **and:**
   ◦ Logical operator used for conjunction.
   ◦ Example: `if x > 0 and y < 10:`

2. **as:**
   ◦ Used to create an alias when importing a module or assigning a variable.
   ◦ Example: `import math as m`

3. **or:**
   ◦ Logical operator used for disjunction.
   ◦ Example: `if x > 0 or y < 10:`

Python Keywords

4. assert:
  ◦ Used for debugging purposes. Raises an exception if a given expression is False.
  ◦ Example: `**assert x > 0, "Value must be positive"**`

5. break:
  ◦ Used to exit from the loop prematurely.
  ◦ Example: `**while True: break**`

6. import:
  ◦ Used to import a module.
  ◦ Example: `**import math**`

Python Keywords

7. continue:
  ◦ Skips the rest of the loop's code and continues with the next iteration.
  ◦ Example: `while x < 10: x += 1; continue`

8. def:
  ◦ Defines a function.
  ◦ Example: `def add(a, b): return a + b`

9. True:
  ◦ Boolean value indicating True.
  ◦ Example: `is_valid = True`

Python Keywords

10. is:
- ◦ Checks if two variables reference the same object.
- ◦ Example: `if a is b:`

11. lambda:
- ◦ Creates an anonymous function (lambda function).
- ◦ Example: `square = lambda x: x**2`
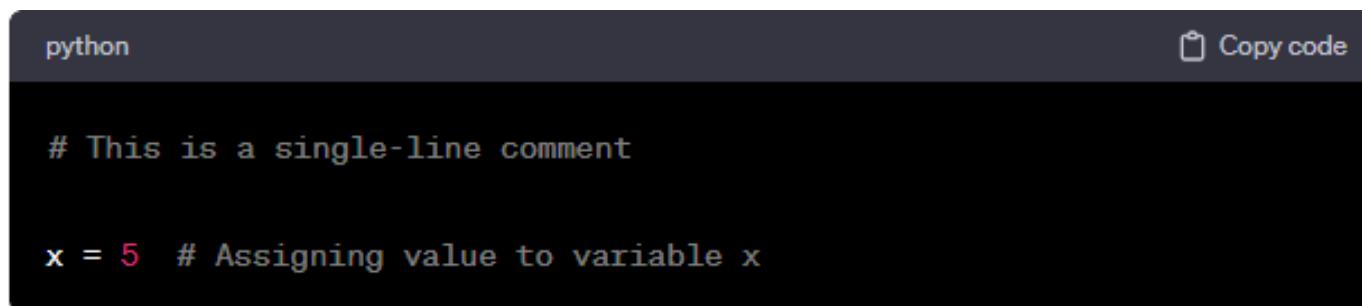
12. None:
- ◦ Represents the absence of a value.
- ◦ Example: `result = None`

Comments in Python

Comments are used to annotate code and provide additional information.

In Python, comments start with the `#` symbol. Anything after `#` on a line is considered a comment and is ignored during execution.
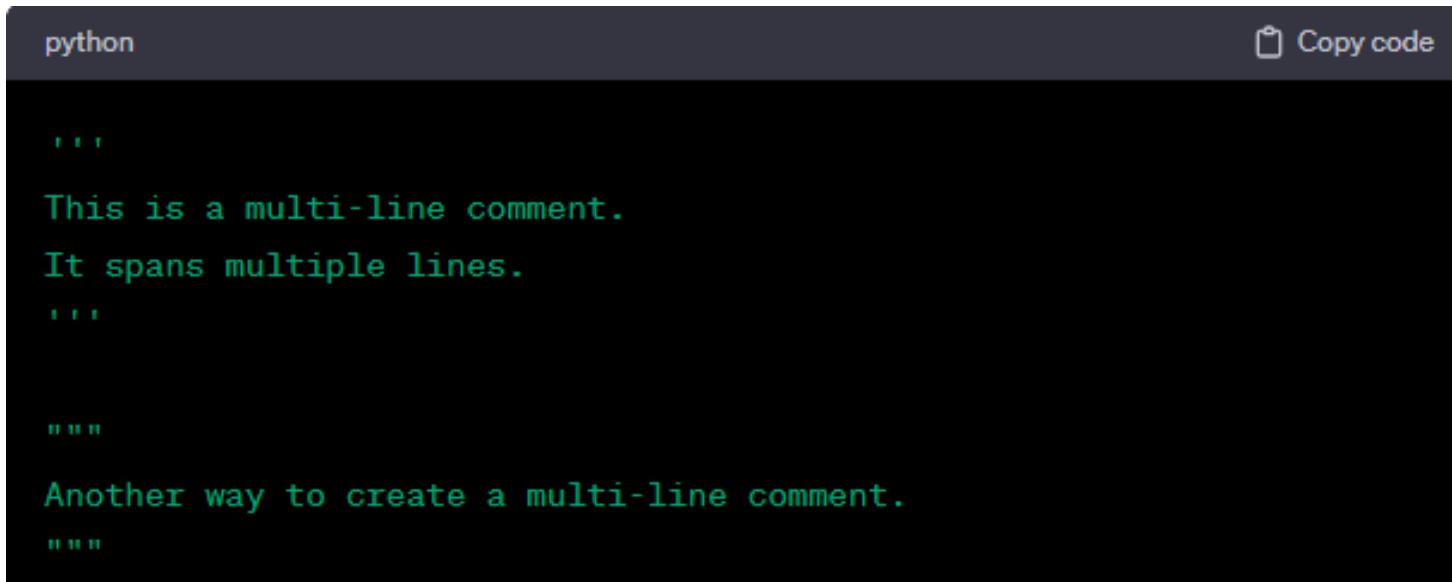
```python
# This is a single-line comment


x = 5    # Assigning value to variable x
```

For multi-line comments, Python does not have a specific syntax, but you can use triple-quotes (`'''`or `" " "`) to create multi-line strings, and they are often used as a workaround for multi-line comments:

Comments in Python

```python
'''
This is a multi-line comment.
It spans multiple lines.
'''



"""

Another way to create a multi-line comment.
"""
```

While the use of triple-quotes for comments is a convention, keep in mind that these strings are still valid Python code, and they create string objects.

They're not true comments in the sense that they are ignored during runtime; they just serve a similar purpose.

Intro to Python Strings

A string is a sequence of characters in Python. Strings are immutable, meaning their values cannot be changed once they are created.

Here are some key aspects of Python strings:

String Creation

String Operations

String Formatting

String Methods

Slicing in Python Strings

String slicing allows you to extract a portion of a string by specifying start and end indices. The syntax is `**string[start:stop:step]**`. Here's an overview:

1. **Basic Slicing:**

```python
my_string = "Hello, World!"
print(my_string[0:5])  # Output: Hello
```

2. **Omitting Indices:**

   - If you omit the start index, it defaults to 0.

   - If you omit the stop index, it defaults to the length of the string.

```python
print(my_string[:5])    # Output: Hello
print(my_string[7:])    # Output: World!
```

Slicing in Python Strings

3. Negative Indices:

Negative indices count from the end of the string.

```
print(my_string[-6:])    # Output: World!
```

4. Reverse String:

You can reverse string using :: option here.

```
my_string = "hello world"
reversed_string = my_string[::-1]
print(reversed_string)
```

```
dlrow olleh
```

String Formatting in Python

String formatting allows you to embed values within a string.

There are multiple methods for string formatting in Python:

1. **Old-Style Formatting (% Operator):**

```python
name = "Alice"
age = 25
print("My name is %s, and I am %d years old." % (name, age))
```

String Formatting in Python

2. str.format() Method:

```python
print("My name is {}, and I am {} years old.".format(name, age))
```

3. f-strings(Formatted String Literals):

```python
print(f"My name is {name}, and I am {age} years old.")
```

Built-in Methods in Python Strings

Python provides numerous built-in methods for strings.

Here are some commonly used ones:

1. **len():**

   Returns the length of the string.

   ```
   length = len(my_string)
   ```

2. **upper() and lower():**

   Converts the string to uppercase or lowercase.

   ```
   uppercase_str = my_string.upper()
   lowercase_str = my_string.lower()
   ```

Built-in Methods in Python Strings

3. strip():

Removes leading and trailing whitespace.

```
stripped_str = my_string.strip()
```

4. split():

Splits the string into a list of substrings based on a delimiter.

```
words = my_string.split(",")  # Output: ['Hello', ' World!']
```

Built-in Methods in Python Strings

5. join():

Joins elements of an iterable (e.g., a list) into a string.

```
new_string = "-".join(['Hello', 'World!'])   # Output: Hello-World!
```

6. replace():

Replaces occurrences of a substring with another substring.

```
replaced_str = my_string.replace("Hello", "Hi")
```

Lists in Python

In Python, a list is a versatile and mutable data structure that can store a collection of items.

Lists are ordered, indexed, and can contain elements of different data types.

They are defined by enclosing comma-separated values within square brackets [] .

Here's a detailed explanation of lists in Python with examples:

Lists in Python

Creating Lists:

You can create a list in Python using square brackets [ ] and separating elements with commas.

```python
# Creating a list of integers
numbers = [1, 2, 3, 4, 5]


# Creating a list of strings
fruits = ["apple", "banana", "orange"]


# Creating a list with mixed data types
mixed_list = [1, "apple", True, 3.14]
```

Lists in Python

Accessing Elements:

Elements in a list are accessed using indices. Python uses 0-based indexing, meaning the first element is at index 0.

```python
# Accessing elements by index
print(numbers[0])    # Output: 1
print(fruits[1])     # Output: "banana"

# Negative indexing (accessing elements from the end)
print(numbers[-1])   # Output: 5 (last element)
```

Lists in Python

Slicing Lists:

You can extract a portion of a list using slicing. Slicing syntax is list[start:end:step], where start is inclusive and end is exclusive.

```python
# Slicing a list
print(numbers[1:4])  # Output: [2, 3, 4]

# Slicing with step
print(numbers[::2])  # Output: [1, 3, 5]
```

Lists in Python

Modifying Lists:

Lists are mutable, meaning you can change their elements after creation.

```python
# Modifying elements
fruits[0] = "grape"
print(fruits)  # Output: ["grape", "banana", "orange"]


# Appending elements
fruits.append("kiwi")
print(fruits)  # Output: ["grape", "banana", "orange", "kiwi"]


# Removing elements
fruits.remove("banana")
print(fruits)  # Output: ["grape", "orange", "kiwi"]
```

Lists in Python

List Operations:

Lists support various operations such as concatenation (+), repetition (*), length (len()), membership (in), and iteration.

```python
# Concatenation
combined_list = numbers + fruits
print(combined_list)  # Output: [1, 2, 3, 4, 5, "grape", "orange", "kiwi"]

# Repetition
repeated_list = fruits * 2
print(repeated_list)  # Output: ["grape", "orange", "kiwi", "grape", "orange", "kiwi"

# Length
print(len(fruits))  # Output: 3

# Membership
print("apple" in fruits)  # Output: False
```

Lists in Python

List Methods:

Python provides several built-in methods to manipulate lists efficiently.

```python
# Sorting
numbers.sort()
print(numbers)  # Output: [1, 2, 3, 4, 5]

# Reversing
numbers.reverse()
print(numbers)  # Output: [5, 4, 3, 2, 1]

# Counting occurrences
print(fruits.count("orange"))  # Output: 1

# Clearing the list
fruits.clear()
print(fruits)  # Output: []
```

Lists in Python

Nested Lists:

Lists can contain other lists, allowing for the creation of nested data structures.

```python
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(nested_list[0][1])  # Output: 2
```

List Comprehensions:

List comprehensions provide a concise way to create lists based on existing lists.

```python
# Squaring numbers
squared_numbers = [x ** 2 for x in numbers]
print(squared_numbers)  # Output: [25, 16, 9, 4, 1]
```

Lists in Python

Exercise :

Write a program to create sum of elements in a list

Write a program to create two lists and use concatenation

Write a program to reverse elements in a list