PYTHON UNIT-1

INTRODUCTION TO PYTHON PROGRAMMING LANGUAGE

INTRODUCTION TO PYTHON LANGUAGE:

Python is a high-level, versatile, and interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability and a clean, concise syntax, making it an excellent choice for both beginners and experienced programmers.

Python has a thriving ecosystem of libraries, frameworks, and tools, which contributes to its popularity and versatility. Some of the most well-known libraries and frameworks include NumPy, pandas, Django, Flask, TensorFlow, Matplotlib, and more. Overall, Python's clean and readable code, broad range of applications, and extensive community support make it an excellent choice for both beginners and professionals in various fields of programming and development. It continues to evolve and adapt to the ever-changing landscape of technology and software development.

STRENGTHS :

Web Development:

Python has frameworks like Django and Flask for building web applications and websites.

Data Analysis and Visualization:

Libraries such as NumPy, pandas, and Matplotlib are used for data analysis and visualization.

Machine Learning and AI:

Python is popular for machine learning and artificial intelligence with libraries like TensorFlow and scikit-learn.

Scientific Computing:

Scientists use Python for simulations, data analysis, and scientific research.

Automation:

Python is used for automating repetitive tasks, making it a popular choice for scripting and process automation.

Game Development:

Python can be used for game development with libraries like Pygame.

Desktop Applications:

Desktop applications, including software tools and graphical user interfaces (GUIs), are built using libraries like PyQt and Tkinter. Network Programming: Python is used for creating network applications, including web scraping and network analysis.

WEAKNESSES:

Memory Consumption:

Python's memory consumption can be higher compared to languages like C or C++, which might not be suitable for memory-constrained applications.

Not Ideal for Systems Programming: Python is generally not the best choice for low level systems programming due to its reliance on a runtime environment. Languages like C and Rust are better suited for such tasks.

Limited Support for Multi threading:

Python's multi threading support is limited due to the GIL. For CPU-bound tasks, you may need to consider multiprocessing to take full advantage of multiple cores.

Less Suitable for Real-Time Applications:

Python's performance and predictability are not ideal for real-time applications that require extremely precise timing.

BUILT-IN FUNCTIONS :

Built-in functions in Python are per-defined functions that are available for use in any

Python program without the need for additional imports or library installations. These functions serve various purposes and are an integral part of the Python

programming language

eg print("Hello, World!") -print()

user_input = input("Enter something: ") - input()

x = 5

y = "Hello" print(type(x))

print(type(y))

numbers = range(1, 6)

my_list = [1, 2, 3, 4, 5] -----total = sum(my_list)

max()

mini()

x = 3.7 rounded_x = round(x)

my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5] sorted_list = sorted(my_list)

CONVERSIONS :

in Python, "conversions" refer to the process of converting one data type into another. Python provides built-in functions for various data type conversions to facilitate data manipulation and type compatibility in your programs

- Explicit Type Conversion.
- Implicit Type Conversion.
- Binary, Octal, and Hexadecimal Conversion.
- Custom Conversions.

Explicit Type Conversion:

You can explicitly convert one data type to another by using the appropriate constructor function. For example, if you have a string containing a numeric value, you can convert it to an integer or float explicitly. int(): Converts a value to an integer. float(): Converts a value to a floating-point number. str(): Converts a value to a string. x = "5"

y = int(x) # Converts string to integer

z = float(x) # Converts string to float

s = str(42)

Imlicit Type Conversion :

Python often performs implicit type conversion, also known as type inference, when performing operations involving different data types. For example, adding an integer and a

floating-point number results in a float. a = 5

b = 2.0

result = a + b # result is a float (7.0)

Binary, Octal, and Hexadecimal Conversion:

bin(): Converts an integer to a binary string.

oct(): Converts an integer to an octal string.

hex(): Converts an integer to a hexadecimal string.

NUMERIC DATA TYPES:

In Python, numeric data types are used to represent numbers. Python provides several built-in numeric data types to work with different kinds of numbers, including integers, floating-point numbers, and

complex numbers. Integers (int):

Integers are whole numbers without a fractional or decimal part. They can be positive, negative, or zero. Python's int type can represent integers of arbitrary length, allowing you to work with very large numbers. Floating-

Point Numbers (float):

Floating-point numbers are numbers that can have a fractional part. They are represented using the float data type.

Complex Numbers (complex):

Complex numbers consist of a real part and an imaginary part, represented by the complex data type. The imaginary part is indicated with a "j" or "J" suffix. These numeric data types allow you to perform various mathematical operations, such as addition, subtraction, multiplication, division, and more. Additionally, Python provides built-in functions and libraries for more advanced mathematical and numeric computations, making it a versatile choice for scientific and engineering applications.

STRING OPERATORS :

A string is a data type in computer programming used to represent text or a sequence of characters. In Python and many other programming languages, a string is a series of characters enclosed in either single (' '), double (" "), or triple (''' ''' or """ """) quotation marks. Strings can contain letters, numbers, symbols, spaces, and even special characters. In Python, single quotes (' '), double quotes (" "), and triple quotes (''' ''' or """ """) are all used to define string literals, and they are functionally equivalent. However, they have different use cases and can be chosen based on your specific needs.

Single Quotes (' '):

Single quotes are used to define a string enclosed within single quotes. This is the most common and straightforward way to define strings in Python. Single quotes can be useful when you need to include double quotes as part of the string. eg: single_quoted = 'This is a string.' multiline_string = '''This is a
Multiline string.

''' Double Quotes (" "):

Double quotes are used to define a string enclosed within double quotes. They are functionally the same as single quotes, and you can choose between them based on your Preference .

eg:double_quoted = "This is another string." Triple Quotes (''' ''' or """ """):

Triple quotes:

Triple quotes can be used to define strings that span multiple lines. They are useful for creating multiline strings,for embedding single and double quotes within the string without escaping.

eg : multiline_string = '''This is a multiline string.'''

embedded_quotes = 'He said, "Hello, World!"'

STRING OPERATORS :

Strings are a fundamental data type in Python, and they come with a variety of operators and built-in functions to manipulate and work with them.

Concatenation:

Combining two or more strings to create a new string.

Repetition:

Repeating a string a specified number of times.

Membership Operators:

Used to check if a substring exists within a string.

Comparison Operators:

Used to compare two strings based on their lexicographical order.

Slice Operator:

Extracting a portion of a string by specifying a range of indices.

String Multiplication with Numbers:

Repeating a string a certain number of times. String Length Operator (len()):

A function used to determine the length (number of characters) of a string.

Concatenation Operator (+):

The + operator is used to concatenate (join together) two or more strings.

```
str1 = "Hello"
str2 = " World"
result = str1 + str2
print(result) # Output: "Hello World"
```

Repetition Operator (*):

The * operator is used to repeat a string a specified number of times.

```
str1 = "Repeat "
result = str1 * 3
print(result) # Output: "Repeat Repeat Repeat "
```

Membership Operators (in and not in):

These operators are used to check if a substring exists within a string. text = "Python is great" print("Python" in text) # Output: True

print("Java" not in text) # Output: True

Comparison Operators (==, !=, <, >, <=, >=):

These operators allow you to compare two strings based on their lexicographical order. str1 = "apple"

str2 = "banana" print(str1 < str2) # Output: True (compares the strings based on dictionary order)

Slice Operator ([: :]):

The slice operator allows you to extract a portion of a string by specifying a range of indices. text = "Python Programming"

sub_string = text[7:18] # Extracts

"Programming" Start-End-Step [2:5:2]

NAMING CONVENTIONS :

Naming conventions in Python are a set of guidelines and recommendations for naming variables, functions, classes, and other identifiers in your code. These conventions are not strict rules enforced by the Python interpreter but are widely followed practices to ensure code readability and consistency. Adhering to naming conventions helps make your code more understandable to both yourself and others who may read or work with your code.

Variable Names:

Use lowercase letters for variable names. Separate words in a variable name with underscores (_) to improve readability (e.g., my_variable, user_name). Choose descriptive and meaningful variable names to make the code self-explanatory.

Constant Names:

Use uppercase letters for constant names.

If a constant name consists of multiple words, separate them with underscores (e.g., MAX_VALUE, PI). Function Names:

Use lowercase letters for function names. Separate words in a function name with underscores (e.g., calculate_total, process_data). Choose meaningful and descriptive names for functions.

Class Names:

Use CamelCase (also known as PascalCase) for class names, where each word begins with an uppercase letter and there are no underscores (e.g., MyClass, PersonInfo).

Module Names:

Use lowercase letters for module names. Avoid using hyphens in module names; use underscores instead (e.g., my_module, not my- module).

Package Names:

Use lowercase letters for package names. Avoid using underscores in package names; use hyphens (dashes) to separate words (e.g., my_package, not my_package).

Method Names:

Follow the same conventions as function names (i.e., lowercase with underscores) for method names within classes.

Private Names:

Prefix variable, function, and method names with an underscore (_) to indicate that they are intended to be private or for internal use. For example, _my_private_var.

Magic Method Names:

Magic methods (also known as dunder methods) that have a special meaning in Python, such as __init__, should be surrounded by double underscores.

Acronyms and Initialisms:

When using acronyms or initialisms, capitalize all letters (e.g., XMLParser, not XmlParser). Avoid Using Reserved Words: Do not use Python's reserved words (keywords) as variable or identifier names.

PEP 8:

For more detailed naming conventions and coding style recommendations, refer to

Python's PEP 8 style guide, which is widely followed by the Python community.

By following these naming conventions, you can create Python code that is more consistent, readable, and maintainable. It also helps you avoid common naming conflicts and issues in

your code.

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):

- An Integrated Development Environment is a comprehensive software application that provides a complete set of tools and features to facilitate software development.
- IDEs are used by software developers to write, edit, debug, and manage code efficiently. They often include code editors, integrated debuggers, build automation tools, and more.
- Python has several popular IDEs, including PyCharm, Visual Studio Code, and IDLE. These IDEs offer features like code auto-completion, debugging, and code navigation.

- IDEs are suitable for professional software development, where productivity, code quality, and collaboration are important.

LEARNING ENVIRONMENT:

- A Learning Environment in the context of Python is a specialized software or web platform designed to teach Python programming to beginners and learners.
- These environments are often interactive and beginner-friendly, providing a hands-on approach to learning Python without the complexities of setting up a full fledged development environment.
- Learning environments can include web-based platforms like Codecademy, Coursera, or edX, which offer interactive Python courses.
- There are also educational coding platforms like Scratch, Thonny, and Trinket, which are designed for teaching programming and Python to beginners.
- Learning environments are ideal for individuals who are new to programming or Python and want to grasp the fundamentals without getting overwhelmed by the intricacies of professional development tools.

In summary, Integrated Development Environments (IDEs) are professional tools for software development, including Python, and are suitable for experienced developers. Learning Environments, on the other hand, are educational platforms and tools designed for beginners and learners who are taking their first steps in Python programming.The

choice between an IDE and a learning environment depends on your level of experience and the purpose of your Python usage.

DYNAMIC TYPING :

In Python, dynamic typing is a programming concept where the data type of a variable is determined at runtime, rather than being explicitly declared at the time of variable creation. This means that a variable can hold values of different data types during its lifetime, and you can change the data type of a variable by assigning a value of a different type to it.

- **No Explicit Type Declarations:** Python does not require you to explicitly declare the data type of a variable. The type is automatically determined based on the value assigned to the variable.
- **Data Type Changes:** You can reassign a variable to a value of a different data type, and Python will adapt to the new data type without errors.
- **Type Checking at Runtime:** Python performs type checking at runtime, which means that type errors are often discovered when the code is executed, rather than at compile time. This can be both an advantage and a potential source of runtime errors.
- **Code Flexibility:** Dynamic typing provides code flexibility, making Python easy to use and allowing for rapid development.

While dynamic typing offers flexibility, it can also lead to certain challenges and errors, especially in larger code bases. It's important to be mindful of the data types you are working with and to write code that

ensures that variables are used appropriately for their intended purpose. Additionally, dynamic typing may not catch type-related errors until runtime, so testing and careful code review are essential to identify issues early in the development process