Python Programming – Theory Notes

1. Introduction to Python

Python is a simple, high-level, interpreted programming language designed to be easy to read and write. It allows developers to express concepts in fewer lines of code compared to other languages. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Features of Python:

1. **Simple and Easy to Learn:** Python's syntax is clean and easy to understand.
2. **Interpreted Language:** Code execution happens line-by-line, which makes debugging easier.
3. **Cross-Platform:** Runs on Windows, macOS, Linux, and many others without modification.
4. **Open Source:** Freely available for everyone.
5. **Extensive Libraries:** Comes with a vast collection of built-in modules and third-party libraries.
6. **Object-Oriented:** Supports encapsulation, inheritance, and polymorphism.
7. **Dynamic Typing:** No need to declare variable types explicitly.

History and Evolution of Python:

Python was created by Guido van Rossum in the late 1980s at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. It was officially released in 1991. The language was named after the British comedy group "Monty Python's Flying Circus."
Major versions include Python 2.x (introduced in 2000) and Python 3.x (released in 2008), with Python 3 being the modern standard.

Advantages of Python over Other Languages:

1. **Readable and Maintainable:** Clear syntax makes it easy to read and maintain.
2. **Rapid Development:** Suitable for fast prototyping and development.
3. **Large Community Support:** A vast global community ensures continuous updates and support.
4. **Integration Capabilities:** Works well with C, C++, Java, and other languages.
5. **Wide Application Range:** Used in web development, data science, AI, ML, and

automation.

Installing Python and Setting Up Environment:

Python can be installed from its official website (python.org). Popular IDEs include:
- **Anaconda:** Best for data science and machine learning.
- **PyCharm:** A powerful IDE for professional developers.
- **VS Code:** Lightweight editor with Python extensions.
After installation, you can verify it using the command:

```
python --version
```

Writing and Executing Your First Python Program:

Example:

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

You can run a Python program by saving it with a .py extension and executing it through the terminal or IDE.

2. Programming Style

Python promotes readability and simplicity. Following PEP 8, the official style guide, helps maintain consistency in code.

Key Elements of Good Programming Style:

1. **Indentation:** Python uses indentation (usually 4 spaces) instead of braces.
2. **Comments:** Use # for single-line comments and triple quotes for multi-line comments.
3. **Naming Conventions:**
   - Variables: lowercase_with_underscores
   - Classes: PascalCase
   - Constants: UPPERCASE
4. **Readable Code:** Keep lines short and functions focused on one task.

3. Core Python Concepts

Python provides several built-in data types and powerful operators for computation.

Data Types in Python:

1. **int** – Whole numbers (e.g., 10)
2. **float** – Decimal numbers (e.g., 10.5)
3. **str** – Sequence of characters (e.g., "Python")
4. **list** – Ordered, mutable collection (e.g., [1, 2, 3])
5. **tuple** – Ordered, immutable collection (e.g., (1, 2, 3))
6. **dict** – Key-value pairs (e.g., {"name": "Karan"})
7. **set** – Unordered collection of unique elements (e.g., {1, 2, 3})

Variables and Memory Allocation:

Variables act as containers for storing data values. Memory allocation happens automatically when you assign a value to a variable. Python uses reference counting and garbage collection for memory management.

Operators in Python:

1. **Arithmetic Operators:** +, -, *, /, %, **, //
2. **Comparison Operators:** ==, !=, >, <, >=, <=
3. **Logical Operators:** and, or, not
4. **Bitwise Operators:** &, |, ^, ~, <<, >>

4. Conditional Statements

Conditional statements help control the flow of a program based on conditions.

Types of Conditional Statements:

1. **if Statement:** Executes a block if a condition is true.
   Syntax:
   ```
   if condition:
       statement
   ```

2. **if-else Statement:** Executes one block if true, another if false.
   Syntax:
   ```
   if condition:
       statement1
   else:
       statement2
   ```

3. **if-elif-else Ladder:** Checks multiple conditions.
  Syntax:
    if condition1:
      statement1
    elif condition2:
      statement2
    else:
      statement3

4. **Nested if:** if inside another if statement.

5. Looping (For, While)

Loops allow repeated execution of a block of code until a condition is met.

For Loop:

Used to iterate over sequences like lists, tuples, or strings.
Syntax:
  for variable in sequence:
    statement

While Loop:

Repeats a block as long as a condition is true.
Syntax:
  while condition:
    statement

6. Generators and Iterators

Generators are special functions that return values one at a time using the `yield` keyword. They are memory efficient and useful for large datasets.

Iterators are objects that allow sequential access to elements in a collection using `__iter__()` and `__next__()` methods.

7. Functions and Methods

Functions are reusable blocks of code that perform specific tasks.
Syntax:

```
def function_name(parameters):
    statement
    return value
```

Types of Arguments:
1. Positional Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable-length Arguments (*args, **kwargs)

Scope: Local and Global variables determine accessibility.

String Methods: Python provides built-in methods like upper(), lower(), replace(), find(), and strip() for manipulation.

8. Control Statements (Break, Continue, Pass)

These statements control the execution flow within loops.

- **break:** Exits the loop immediately.
- **continue:** Skips the current iteration and moves to the next.
- **pass:** Acts as a placeholder where code is syntactically required but not implemented yet.

9. String Manipulation

Strings in Python are immutable sequences of Unicode characters.

Operations:
1. Concatenation (+)
2. Repetition (*)
3. Slicing [start:end:step]
4. Methods: upper(), lower(), title(), strip(), replace(), find(), count()

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

Python supports functional programming using higher-order functions.

map(): Applies a function to all items in an iterable.
Syntax: map(function, iterable)

reduce(): Applies a rolling computation to items in a sequence.
Syntax: reduce(function, sequence)

filter(): Filters elements based on a condition.
Syntax: filter(function, iterable)

Closures: Functions defined inside another function that remember the outer scope's variables even after the outer function has finished execution.

Decorators: Functions that modify the behavior of another function without changing its code. Defined using the @decorator_name syntax.