**### \*Introduction to Python Programming:\***

Python is a high-level, interpreted, and dynamically typed programming language. It emphasizes code readability with its clear syntax and indentation. Python supports multiple programming paradigms such as procedural, object-oriented, and functional programming.

**### \*Features of Python:\***

1. \*Simple and Easy to Learn\*: Python has a simple syntax, making it easy to learn and understand.

2. \*Interpreted Language\*: Python code is executed line-by-line, which simplifies debugging.

3. \*Dynamically Typed\*: Python doesn’t require declaring the data type of variables, allowing more flexibility.

4. \*Cross-platform\*: Python is portable and can run on various operating systems like Windows, macOS, and Linux.

5. \*Extensive Standard Library\*: Python has a rich set of built-in modules and libraries.

6. \*Open Source\*: Python is free and open-source, meaning you can contribute to its development.

7. \*Object-Oriented\*: Python supports object-oriented programming with classes and objects.

8. \*Embeddable and Extensible\*: You can embed Python in other languages or extend it with additional modules.

**### \*Execution of a Python Program:\***

Python programs can be executed in two main ways:

1. \*Interactive Mode\*: You can run Python commands directly in the Python interpreter.

2. \*Script Mode\*: You can save Python code in a .py file and run it via the command line or an Integrated Development Environment (IDE).

**### \*Variables and Identifiers:\***

- \*Variables\*: Containers that store data values.

Example:

python

x = 10

name = "Alice"

**- \*Identifiers**\*: Names given to variables, functions, classes, etc. They must start with a letter or underscore, followed by letters, digits, or underscores.

Example: my\_variable, \_hiddenVar.

**### \*Data Types:\***

Python has various built-in data types:

1. \*Numeric\*: int, float, complex

2. \*Sequence\*: list, tuple, range

3. \*Text\*: str

4. \*Boolean\*: bool (True or False)

5. \*Set\*: set, frozenset

6. \*Dictionary\*: dict

**### \*Input Operation:\***

To take input from the user:

python

name = input("Enter your name: ")

age = int(input("Enter your age: "))

**### \*Comments:\***

Comments are used for documentation and are ignored by the interpreter.

- Single-line comment: # This is a comment

- Multi-line comment:

python

"""

This is a

multi-line comment

"""

**### \*Reserved Words:\***

Python has a set of reserved words that cannot be used as identifiers (variable or function names). Examples include if, else, while, for, break, continue, etc.

**### \*Indentation:\***

Python uses indentation to define the structure and flow of the program instead of curly braces {}. Consistent indentation is crucial in Python.

Example:

python

if x > 5:

print("x is greater than 5")

### \*Operators and Expressions:\*

Python supports various operators:

1. \*Arithmetic Operators\*: +, -, \*, /, //, %, \*\*

2. \*Comparison Operators\*: ==, !=, <, >, <=, >=

3. \*Logical Operators\*: and, or, not

4. \*Assignment Operators\*: =, +=, -=, \*=, etc.

5. \*Membership Operators\*: in, not in

6. \*Identity Operators\*: is, is not

**### \*Expressions in Python:\***

An expression is a combination of values, variables, and operators that evaluates to a value.

Example:

python

result = 5 + 3 \* 2

**### \*Decision Control Statements:\***

1. **\*if Statement\*:** Used to execute a block of code if a condition is true.

python

if age >= 18:

print("You are an adult")

2. **\*if-else Statement\*:** Executes one block of code if the condition is true, otherwise another block is executed.

python

if age >= 18:

print("You are an adult")

else:

print("You are a minor")

**### \*Branching Statements:\***

1**. \*if-elif-else\*:** Used when multiple conditions need to be checked.

python

if age < 18:

print("Minor")

elif age == 18:

print("Just an adult")

else:

print("Adult")

**### \*Iterative Statements (Loops):\***

**1. \*for Loop\*:** Iterates over a sequence (like a list, string, or range).

python

for i in range(5):

print(i)

2. \***while Loop**\*: Repeats as long as a condition is true.

python

i = 0

while i < 5:

print(i)

i += 1

**### \*break, continue, pass, and else with Loops:\***

**- \*break\*:** Exits the loop.

python

for i in range(5):

if i == 3:

break

print(i)

- **\*continue**\*: Skips the current iteration and proceeds to the next.

python

for i in range(5):

if i == 3:

continue

print(i)

**- \*pass\*:** Acts as a placeholder in the loop.

python

for i in range(5):

pass # Do nothing for now

- \*else\*: Executes when the loop completes without a break.

python

for i in range(5):

print(i)

else:

print("Loop completed")

**### \*Comparisons between C and Python:\***

| Feature | C | Python |

|-----------------------|------------------------------------------|----------------------------------------------|

| \*Type\* | Compiled, statically typed | Interpreted, dynamically typed |

| \*Syntax\* | Requires braces {} for code blocks | Uses indentation for code blocks |

| \*Memory Management\* | Manual (malloc/free) | Automatic (Garbage collection) |

| \*Data Types\* | Primitive types (int, char, float) | Built-in types (int, float, list, dict) |

| \*Control Statements\*| if-else, switch | if-else, elif |

| \*Loops\* | for, while, do-while | for, while |

| \*Functions\* | Needs explicit declaration | Function declarations are flexible |

| \*Pointers\* | Supports pointers | Does not support pointers |

| \*Libraries\* | Standard and third-party libraries | Extensive built-in and third-party libraries |

| \*Performance\* | Faster due to lower-level access | Slower but more efficient for development |

Python's simplicity and versatility make it a popular choice for various applications, while C offers more control and efficiency for low-level programming