**INTRODUCTION TO PYTHON**

**1.Introduction to python and its features.**

-python is widely used general programming language.

-it was designed by guido van Rossum in 1991 and developed by python software management.

-it was mainly developed for emphasis on code readability and it is an interpreted,object oriented high-level programming language.

**Key Features**

* **Easy to Learn**: Clean, readable syntax.
* **Interpreted**: Runs code line-by-line.
* **Dynamic**: No need to define variable types.
* **Versatile**: Supports multiple programming paradigms.
* **Rich Libraries**: Tools for various tasks like web, data, and AI.
* **Cross-Platform**: Works on any OS.

**2.History and evolution of python.**

In 1989, Guido van Rossum wanted a programming language to be easy and fun, unlike all those complex ones at that time. So, in 1991, he published his language, naming it after his favorite comedy group: Monty Python.  
  
Over time, Python was developed  
  
2000: Python 2 rendered the coding more efficient, though problematic at times.  
In 2008, Python 3 incorporated all current features that addressed these issues but was not fully compatible with Python 2.  
2020: Python 2 was formally retired and Python 3 became the widely used version.

**3. Advantages of Python Over Other Programming Languages**

-Python is an interpreted programming language.

-it is designed to be easy to learn and master.

-it has clean and clear syntax.

-Runs almost everywhere.

-designed to be extensible using c/c++, allowing access to many external libraries

-python supports modules and packages which encourages program modularity and code reuseability.

-in python,variables doesn’t need any declaration.

-reduced development time.

**4.writing and executing your first python program**

**Write Your First Program**

Here’s a simple "Hello, World!" program:

python

Copy code

# This is your first Python program

print("Hello, World!")

Steps:

1. Open your editor.
2. Write the code above.
3. Save the file with a .py extension, for example, hello.py.

**Execute Your Python Program**

You can run your Python program in several ways:

Using a Terminal or Command Prompt

1. Open the terminal or command prompt.
2. Navigate to the directory where you saved your file.

**Programming Style**

**1.understanding python's pep 8 guidelines**

**1. Code Layout**

* **Indentation:** Use 4 spaces per level.
* **Line Length:** Limit to 79 characters.
* **Blank Lines:** Use 2 blank lines for top-level functions/classes and 1 for methods.

**2. Naming Conventions**

* **Variables/Functions:** Use snake\_case.
* **Classes:** Use PascalCase.
* **Constants:** Use UPPERCASE.

**3. Whitespace**

* Avoid unnecessary spaces around operators, brackets, and commas.

**4. Comments**

* Write clear, meaningful comments and use docstrings for functions/classes.

**5. Imports**

* Place imports at the top of the file, grouped as:
  1. Standard library imports.
  2. Third-party imports.
  3. Local imports.

**2.indentation, comments, and naming conventions in python.**

**1. Indentation**

* Use 4 spaces for each level of indentation.

Ex

Num1 = int(input(“enter num1”)

Num2 =int(input(“enter num2”)

If num1>num2:

print(“num1 is greater”)

else:  
 print(“num 2 is greater”)

**2. Comments**

* Single-Line Comment: Use # for brief explanations.

python

# This function adds two numbers

* Docstrings: Use """ for function/class documentation.

"""Return the square of a number.""

**3. Naming Conventions**

* Variables/Functions: Use snake\_case.

Python

total\_sum = 0

def calculate\_total():

pass

* Classes: Use PascalCase.

python

Copy code

class DataProcessor:

pass

* Constants: Use UPPERCASE.

python

Copy code

MAX\_LIMIT = 100

**3.writing readable and maintainable code.**

1. Follow PEP 8: Use consistent indentation, naming conventions, and line length guidelines.
2. Use Clear Names: Choose descriptive and meaningful names for variables, functions, and classes.
3. Add Comments and Docstrings: Clearly explain the purpose and functionality of your code.
4. Keep Code Modular: Break code into reusable functions and classes to improve organization and readability.
5. Test and Document: Write tests and maintain documentation to make your code easier to debug and maintain.
6. **Core python concepts**

Python is a dynamically typed language, meaning you don't need to explicitly declare the data type of a variable. The interpreter infers the type at runtime. Here are some fundamental data types:

**1. Integers:**

* Represent whole numbers.
* Examples: 10, -5, 0

**2. Floats:**

* Represent real numbers with decimal points.
* Examples: 3.14, -2.5, 1.0

**3. Strings:**

* Represent sequences of characters.
* Enclosed in single quotes (') or double quotes (").
* Examples: 'Hello', "Python is fun"

**4. Lists:**

* Ordered collections of items.
* Mutable (can be changed).
* Enclosed in square brackets [].
* Examples: [1, 2, 3], ['apple', 'banana', 'cherry']

**5. Tuples:**

* Ordered collections of items.
* Immutable (cannot be changed).
* Enclosed in parentheses ().
* Examples: (1, 2, 3), ('a', 'b', 'c')

**6. Dictionaries:**

* Unordered collections of key-value pairs.
* Keys must be unique and immutable.
* Enclosed in curly braces {}.
* Examples: {'name': 'Alice', 'age': 30}, {'fruits': ['apple', 'banana']}

**7. Sets:**

* Unordered collections of unique items.
* Enclosed in curly braces {} or using the set() function.

Examples: {1, 2, 3}, {'a', 'b', 'c'}

**Python Variables and Memory Allocation**

* Variables are names given to memory locations to store data.
* When you assign a value to a variable, Python allocates memory to store that value.
* The type of the variable determines the amount of memory allocated.
* Python uses a garbage collector to automatically reclaim memory that is no longer needed.

**Python Operators**

**1. Arithmetic Operators:**

* + (addition)
* - (subtraction)
* \* (multiplication)
* / (division)
* // (floor division)
* % (modulo)
* \*\* (exponentiation)

**2. Comparison Operators:**

* == (equal to)
* != (not equal to)
* < (less than)
* > (greater than)
* <= (less than or equal to)
* >= (greater than or equal to)

**3. Logical Operators:**

* and (logical AND)
* or (logical OR)
* not (logical NOT)

**4. Bitwise Operators:**

* & (bitwise AND)
* | (bitwise OR)
* ^ (bitwise XOR)
* ~ (bitwise NOT)
* << (left shift)
* >> (right shift)

1. **Conditional statements**

**1.Introduction to conditional statements: if, else, elif**

1. if Statement

The if statement checks a condition. If the condition evaluates to True, the code block under it executes.

python

Copy code

age = 20

if age >= 18:

print("You are an adult.")

* If age >= 18 is True, it prints "You are an adult."
* If the condition is False, the block is skipped.

2. else Statement

The else statement provides an alternative block of code to execute if the if condition is False.

python

Copy code

age = 16

if age >= 18:

print("You are an adult.")

else:

print("You are a minor.")

* If age >= 18 is False, it executes the else block and prints "You are a minor."

3. elif Statement

The elif (short for "else if") statement allows you to check multiple conditions. It comes after the if and before the else.

python

Copy code

marks = 85

if marks >= 90:

print("Grade: A")

elif marks >= 75:

print("Grade: B")

else:

print("Grade: C")

* If marks >= 90 is True, it prints "Grade: A".
* If the first condition is False, the elif condition (marks >= 75) is checked.
* If all previous conditions are False, the else block executes.

2**.Nested if-else conditions.**

Nested if-else conditions involve placing one if, elif, or else block inside another. This is useful for checking multiple levels of conditions. Here's how it works:

Syntax of Nested if-else

python

Copy code

if condition1:

if condition2:

# Block executed if condition1 and condition2 are True

else:

# Block executed if condition1 is True but condition2 is False

else:

# Block executed if condition1 is False

Example 1: Checking Age Groups

python

Copy code

age = 25

if age > 0: # Outer condition

if age < 18: # Nested condition

print("You are a minor.")

else:

print("You are an adult.")

else:

print("Invalid age.")

* If age > 0 is True, it checks the inner condition (age < 18).
* If age > 0 is False, the program skips to the outer else block.

**5.looping(for,while)**

**Loops in Python: Iterating Over Code**

Loops are a fundamental programming construct that allows you to repeatedly execute a block of code. Python offers two primary types of loops: for loops and while loops.

**1. For Loops**

A for loop is used to iterate over a sequence of items, such as a list, tuple, or string.

Python

for item in sequence:

# Code to be executed for each item

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Example:**

Python

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

Use code [with caution.](https://d.docs.live.net/faq#coding)

**2. While Loops**

A while loop repeatedly executes a block of code as long as a certain condition is true.

Python

while condition:

# Code to be executed while the condition is True

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Example:**

Python

count = 0

while count < 5:

print(count)

count += 1

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Using Loops with Collections**

**Iterating over Lists and Tuples:**

Python

my\_list = [1, 2, 3, 4, 5]

for number in my\_list:

print(number)

my\_tuple = (10, 20, 30)

for item in my\_tuple:

print(item)

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Iterating over Strings:**

Python

name = "Python"

for char in name:

print(char)

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Iterating over Dictionaries:**

Python

my\_dict = {"name": "Alice", "age": 30, "city": "New York"}

for key, value in my\_dict.items():

print(key, value)

Use code [with caution.](https://d.docs.live.net/faq#coding)

**Key Points:**

* **Indentation:** The code within a loop must be indented to be recognized as part of the loop.
* **Break and Continue:**
  + break: Exits the loop immediately.
  + continue: Skips the current iteration and moves to the next.
* **Loop Control:**
  + Use range() function to generate a sequence of numbers:

Python

for i in range(5):

print(i)

# **8.control statements**

1. Break:

* Purpose: Terminates the loop immediately.
* Usage: When you want to exit the loop prematurely, regardless of the loop condition.

This code will print numbers from 0 to 4, and then the loop will break.

2. Continue:

* Purpose: Skips the current iteration of the loop.
* Usage: When you want to skip certain iterations based on a condition.

3. Pass:

* Purpose: Does nothing.
* Usage: As a placeholder to ensure syntactically correct code, especially when a block is required but no specific action is needed.

9.string manuplation

1.Understanding how to access and manipulate strings.

Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

String slicing.

-->Concatenation: The + operator combines two strings into one.

Example: "Hello" + " World" results in "Hello World".

Repetition: The \* operator repeats a string multiple times.

Example: "Hello " \* 3 results in "Hello Hello Hello ".

String Methods:

upper(): Converts all characters in the string to uppercase.

lower(): Converts all characters in the string to lowercase.

title(): Capitalizes the first character of each word.

String Slicing:

You can extract parts of a string using slicing: string[start:end].

Example: "Hello World"[0:5] results in "Hello".

Negative Indexing:

Negative indices allow you to count from the end of the string.

Example: str[-1] gives the last character.

Advanced Slicing:

You can also define a step size in slicing: string[start:end:step].

Example: "Hello World"[0:10:2] results in "Hoo ol".

String Length: len() function returns the length of the string.

Example: len("Hello") returns 5.

String Membership: The in operator checks if a substring exists in a string.

Example: 'Hello' in 'Hello World' returns True.