**Python Fundamentals**

**Name: Limbasiya Vishvash**

**Theory:**

Q.1:- Introduction to Python and its Features.

Ans:-

Introduction to Python

Python is a simple, high-level, and interpreted programming language known for its readability and versatility. It is widely used in web development, data science, AI, automation, and more.

1.Simple & Easy to Learn – Uses clear and readable syntax.

2. High-Level Language – No need to manage memory or low-level operations.

3.Interpreted – Runs code line by line, making debugging easier.

4. Dynamically Typed – No need to declare variable types.

5. Cross-Platform – Works on Windows, macOS, and Linux.

6. Extensive Libraries – Supports AI, data science, web development, and more.

Q.2:- History and evolution of Python.

Ans:-

Python is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed to emphasize code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Q.3:- Advantages of using Python over other programming languages.

Ans:-

1. Simple & Readable Syntax – Easier to learn and write compared to languages like C, C++, or Java.

2. Interpreted Language – No need for compilation; executes line by line, making debugging easier.

3. Cross-Platform – Works on Windows, macOS, Linux, etc., without modification.

4. Large Standard Library – Comes with built-in modules for web development, AI, data science, etc.

5. Dynamically Typed – No need to declare variable types explicitly.

6. Strong Community Support – Large global community with extensive documentation and libraries.

7. Versatility – Used in web development, AI, automation, data science, game development, etc.

8. Integration & Extensibility – Easily integrates with C, C++, Java, and other languages.

9. Open Source & Free – Available for everyone without licensing fees.

10. Productivity & Rapid Development – Requires fewer lines of code compared to Java or C++.

Q.4:- Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

Ans:-

**Installing Python**

1. Image from Python.org. The most popular way of installing Python is through Anaconda Distribution. ...
2. Image from Anaconda. ...
3. Image from Visual Studio Code. ...
4. Create a new file. ...
5. Save Python file. ...
6. Select the interpreter. ...
7. Run a Python file. ...
8. Install VSCode Python extension.

Q.5:- Writing and executing your first Python program.

Ans:-

**1. Using Python Interpreter (Quick Test)**

1. Open **Command Prompt/Terminal**
2. Type python and press **Enter**
3. Run:

Python:

print("Hello, Python!")

1. Output:

Hello, Python!

Q.6:- Understanding Python’s PEP 8 guidelines.

Ans:-

Key PEP 8 Guidelines:

Indentation: Use 4 spaces per indentation level (no tabs).

Line Length: Keep lines ≤ 79 characters for better readability.

Blank Lines: Use 2 blank lines between top-level functions and classes.

Naming Conventions:

Variables & functions: lower\_case\_with\_underscores

Classes: CapWords (PascalCase)

Constants: ALL\_CAPS

Whitespace: Avoid extra spaces inside parentheses, brackets, or braces.

✅ my\_list = [1, 2, 3]

❌ my\_list = [ 1, 2, 3 ]

Imports: Place imports at the top and group them as:

Standard libraries

Third-party packages

Local application imports

Comments:

Use # for inline comments.

Write meaningful docstrings for functions and classes.

Q.7:- Indentation, comments, and naming conventions in Python.

Ans:-

**1. Indentation (Code Structure)**

* Python **uses indentation instead of braces** {}.
* Always use **4 spaces** per indentation level (no tabs).
* Indentation is **mandatory** in Python.

**2. Comments (Code Explanation)**

* **Single-line comment:** Use #
* **Multi-line comment:** Use triple quotes """ """ or ''' '''

**3. Naming Conventions (PEP 8 Guidelines)**

* **Variables & Functions:** snake\_case
* **Constants:** ALL\_CAPS
* **Classes:** PascalCase
* **Private Variables:** \_single\_leading\_underscore
* **Dunder (Magic) Methods:** \_\_double\_leading\_underscore\_\_

Q.8:- Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

Ans:-

**1. Numeric Types**

* **Integer (int)** → Whole numbers (e.g., 10, -5)
* **Float (float)** → Decimal numbers (e.g., 3.14, -0.99)

**2. String (str)**

* A sequence of characters enclosed in **quotes** (' ' or " ").
* Strings are **immutable** (cannot be changed).

**3. List (list)**

* **Ordered, mutable (changeable), allows duplicates**
* Uses **square brackets** []

**4. Tuple (tuple)**

* **Ordered, immutable (unchangeable), allows duplicates**
* Uses **parentheses** ()

**5. Dictionary (dict)**

* **Key-value pairs, unordered, mutable, no duplicate keys**
* Uses **curly braces** {}

**✅ 6. Set (set)**

* **Unordered, mutable, no duplicates**
* Uses **curly braces** {}

Q.9:- Python variables and memory allocation.

Ans:-

* Dynamic Typing: Python variables don't have fixed types, and the type is determined at runtime. When you assign a value to a variable, Python internally keeps track of the type of the object.
* Reference Counting: Python uses reference counting to manage memory. Each object has a count of how many references (variables, structures, etc.) point to it. When the reference count reaches zero, the object is automatically deallocated by the garbage collector.
* Collection: Python also uses garbage collection to remove unreachable objects and free up memory, especially when there are cyclic references (objects that refer to each other).
* Immutable vs Mutable: Immutable objects (like integers and strings) can't be changed after creation, while mutable objects (like lists and dictionaries) can be modified in place.

Q.10:- Python operators: arithmetic, comparison, logical, bitwise.

Ans:- 1. Arithmetic Operators

These operators are used to perform basic arithmetic operations:

+ : Addition

- : Subtraction

\* : Multiplication

/ : Division (returns float)

// : Floor Division (returns the integer part of the division)

% : Modulus (returns the remainder of the division)

\*\* : Exponentiation (raises a number to a power)

**2. Comparison Operators**

These operators are used to compare two values:

* == : Equal to
* != : Not equal to
* > : Greater than
* < : Less than
* >= : Greater than or equal to
* <= : Less than or equal to

3. Logical Operators

These operators are used to combine conditional statements:

and : Returns True if both conditions are true

or : Returns True if at least one condition is true

not : Reverses the boolean value (returns True if the condition is false)

Q.11:- Introduction to conditional statements: if, else, elif.

Ans:-

**1. if Statement**

The if statement is used to check a condition. If the condition is true, the block of code under the if statement will be executed.

if condition:

# Code to execute if condition is true

**2. else Statement**

The else statement provides an alternative block of code that will be executed if the if condition is **false**. It is optional and is used to handle the case when the if condition isn't satisfied.

if condition:

# Code to execute if condition is true

else:

# Code to execute if condition is false

**3. elif (else if) Statement**

The elif statement is used to check multiple conditions. If the if condition is false, it checks the next condition in the elif block. You can have multiple elif statements for different conditions.

if condition1:

# Code to execute if condition1 is true

elif condition2:

# Code to execute if condition1 is false but condition2 is true

else:

# Code to execute if neither condition1 nor condition2 is true

Q.12:- Nested if-else conditions.

Ans:-

Nested if-else conditions refer to placing one if-else statement inside another. This allows for more complex decision-making by checking multiple layers of conditions. The basic structure looks like this:

if condition1:

if condition2:

# Code to execute if condition1 and condition2 are both true

else:

# Code to execute if condition1 is true, but condition2 is false

else:

# Code to execute if condition1 is false

Q.13:- Introduction to for and while loops.

Ans:- **1. for Loop**

A for loop is typically used when you know how many times you want to iterate over a sequence (like a list, string, or range). It automatically picks each item in the sequence and assigns it to a variable, then executes the block of code for each item.

**Syntax:**

for variable in sequence:

# Code to execute for each item in sequence

**2. while Loop**

A while loop is used when you want to repeat a block of code as long as a condition is true. It will continue looping until the condition evaluates to False.

**Syntax:**

while condition:

# Code to execute as long as condition is true

Q.14:- How loops work in Python.

Ans:-

**1. How the for Loop Works**

The for loop is used to iterate over a sequence (such as a list, tuple, string, or range). The loop automatically picks each element from the sequence and runs the code block for each element.

**Flow of Execution:**

1. The for loop starts by selecting the first element of the sequence.
2. The code block inside the loop is executed with the current element.
3. The loop moves to the next element in the sequence and repeats the process.
4. The loop continues this process until it reaches the end of the sequence.

**2. How the while Loop Works**

The while loop is used when you want to repeat a block of code as long as a condition is true. The loop will continue executing until the condition evaluates to False.

**Flow of Execution:**

1. The condition is evaluated before each iteration.
2. If the condition is True, the code block inside the while loop runs.
3. After the code block executes, the condition is evaluated again.
4. This process repeats until the condition becomes False.

Q.15:- Using loops with collections (lists, tuples, etc.).

Ans:-

**1. Lists and Tuples**

Both lists and tuples are ordered collections, and you can use a for loop to iterate through their elements.

Python:

# List example

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

for fruit in fruits:

print(fruit)

# Tuple example

coordinates = (1, 2, 3)

for coordinate in coordinates:

print(coordinate)

**2. Dictionaries**

For dictionaries, you can loop through keys, values, or both. By default, a for loop iterates over the dictionary's keys.

python

# Dictionary example

person = {"name": "Alice", "age": 25}

for key in person:

print(key, person[key]) # prints key and value

# Looping through values

for value in person.values():

print(value)

# Looping through both keys and values

for key, value in person.items():

print(key, value)

**3. Sets**

Sets are unordered collections, but you can still use a for loop to iterate over their elements.

python

CopyEdit

# Set example

colors = {"red", "green", "blue"}

for color in colors:

print(color)

Q.16:- Understanding the role of break, continue, and pass in Python loops.

Ans:-

**1. break**

The break statement is used to exit the loop entirely, regardless of the loop's condition. It stops further iterations.

python

CopyEdit

for i in range(10):

if i == 5:

break # Exits the loop when i is 5

print(i)

**2. continue**

The continue statement is used to skip the current iteration and proceed to the next iteration of the loop. The loop condition is checked again.

python

CopyEdit

for i in range(10):

if i == 5:

continue # Skips printing 5 and continues with the next iteration

print(i)

**3. pass**

The pass statement is a placeholder that does nothing. It's useful when a statement is syntactically required but you don’t want to implement any action yet (e.g., in empty loops or functions).

python

for i in range(3):

if i == 1:

pass # Does nothing when i is 1

else:

print(i)

Q.17:- Understanding how to access and manipulate strings.

Ans:-

1. **Access Characters**: Use indexing (string[index]) and slicing (string[start:end]).

2. **Concatenate**: Join strings with +.

3. **Repeat**: Repeat strings with \*.

4. **Change Case**: Use methods like upper(), lower().

5. **String Methods**: Use methods like strip(), replace(), split().

6. **Check Substrings**: Use in to check if a substring exists.

7. **Escape Characters**: Use \ for special characters (e.g., \", \n).

Q.18:- Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

Ans:-

\* **Concatenation**: Join two or more strings using +.

text = "Hello" + " " + "World" # "Hello World"

\* **Repetition**: Repeat a string using \*.

text = "Ha" \* 3 # "HaHaHa"

\* **String Methods**:

* **upper()**: Converts the string to uppercase.

text = "hello"

print(text.upper()) # "HELLO"

* **lower()**: Converts the string to lowercase.

text = "HELLO"

print(text.lower()) # "hello"

* **strip()**: Removes leading and trailing whitespace.

text = " hello "

print(text.strip()) # "hello"

* **replace(old, new)**: Replaces occurrences of old with new.

text = "hello world"

print(text.replace("world", "Python")) # "hello Python"

* **split()**: Splits the string into a list based on a delimiter.

text = "hello world"

print(text.split()) # ["hello", "world"]

Q.19:- String slicing.

Ans:-

1.**start**: Where the slice begins (inclusive).

2.**end**: Where the slice ends (exclusive).

3.**step**: The interval between characters (optional).