Python Theory Assignment

Module - 1

Q1) Introduction to Python and its Features (simple, high-level, interpreted language).

Ans:

- **Introduction**:
  - Python is a high-level, interpreted, and general-purpose programming language.
  - It was created by Guido van Rossum and first released in 1991.
  - Python is designed to be easy to read and write, making it perfect for beginners and professionals alike.
  - Python is widely used in various fields like:
    - Web development (Django, Flask)
    - Data science and machine learning (Pandas, NumPy, scikit-learn)
    - Automation and scripting
    - Game development
    - IoT, AI, and more
  - Its versatility makes it a top choice in the tech industry.

Python is known for its simple syntax, which makes it easy to read and write code, even for beginners.

- **Features :**

1. **Simple and easy to learn**

➢ **Clean and readable syntax**

Python code is written in a way that looks like plain English. Example:

print("Hello").

➢ **Beginner-friendly**

Python is often the first language taught in schools and colleges because it's easy

to understand.

➢ **Less code, more work**

You can do more with fewer lines of code compared to other languages like C or

Java.

➢ **No need to declare variables**

Just use a variable, and Python figures out the type.

2. High-Level Language

➢ **Focus on logic, not system details**

You don't have to worry about things like memory management or CPU instructions.

➢ **Write once, run anywhere**

Python is platform-independent, so you can run the same code on Windows,

Mac, or Linux.

➢ **Human-like syntax**

You can understand the logic just by reading it, even without deep programming knowledge.

3. Interpreted Language

➢ **No need for compilation**

Unlike C or Java, Python runs the code directly without converting it into machine code first.

➢ **Line-by-line execution**

Python runs your code one line at a time, which makes it easy to find and fix errors.

➢ **Quick testing and debugging**

You can test your code in small parts using the Python shell or terminal.

➢ **Saves time during development**

Faster testing means faster development, especially for small scripts.

Q2) History and evolution of Python.

Ans:

- Python was created by Guido van Rossum in 1989 at CWI, Netherlands.
- The first official release (Python 0.9.0) came in 1991, featuring functions and core data types.
- It was influenced by the ABC language and designed to be simple and readable.
- Python 1.0 was released in 1994 with features like lambda, map(), filter(), and reduce().
- Python 2.0 came in 2000, adding list comprehensions, garbage collection, and the set type.
- Python 3.0 was launched in 2008 with major changes like improved Unicode support and cleaner syntax.
- Python 2 reached end-of-life on January 1, 2020, after years of transition to Python 3.

- The language is developed as open-source under the Python Software Foundation (PSF).
- It gained massive popularity in the 2010s, especially in AI, data science, and web development.
- Modern versions (Python 3.10, 3.11, etc.) continue to evolve with new features, better performance, and developer-friendly too.

Q3) Advantages of using Python over other programming languages.

Ans:

- Python code looks like plain English, making it easy to write, read, and understand — great for beginners and professionals alike.
- Due to its high-level nature and minimal boilerplate, Python allows you to develop software faster than many other languages like Java or C++.
- Python comes with a powerful standard library and has millions of third-party packages (NumPy, Pandas, Flask, Django, etc.).
- Python runs on all major operating systems — Windows, macOS, Linux — with little or no code modification.
- You can use Python for web development, data science, machine learning, automation, scripting, game development, and more — all with the same language.
- Python has one of the largest and most active communities, meaning lots of tutorials, forums, libraries, and help available for learners and developers.
- Python can easily integrate with C/C++, Java, .NET, databases, and web services, making it suitable for building complex systems.
- You don’t need to declare variable types, which makes coding faster and reduces verbosity.
- Python allows you to build and test ideas quickly — ideal for startups, research, and innovation.
- Python is trusted by companies like Google, Facebook, Netflix, NASA, YouTube, and many more — making Python skills highly valuable in the job market.

Q4) Writing and executing your first Python program.

Ans:

```
a = 10

b = 20

sum = a + b

print("The sum is:", sum)
```

Q5) Understanding Python's PEP 8 guidelines.

Ans:

- Indentation: Use 4 spaces per level.
- Variable names: Use lowercase with underscores (user_name).
- Line length: Limit to 79 characters.
- Avoid extra spaces: Don't use unnecessary spaces around operators.
- Imports: Place them at the top of the file.
- Goal: Improve readability, consistency, and team collaboration

Q6) Indentation, comments, and naming conventions in Python.

Ans:

➢ **Indentation**

- Python uses indentation to define code blocks (no {} like C/Java).
- Standard: 4 spaces per indentation level.

➢ **Comments**

- Used to explain code.

- Start with # for single-line comments.

➢ **Naming Conventions**

- Variables/functions: lower_case_with_underscores (e.g., student_name)
- Constants: ALL_UPPER_CASE (e.g., MAX_SIZE)
- Classes: CamelCase (e.g., StudentData)

Q7) Writing readable and maintainable code.

Ans:

- Use clear and meaningful names

  → Variables and functions should describe their purpose.

- Keep your code organized

  → Break large tasks into small, well-named functions.

- Add comments and docstrings

  → Explain why something is done, not just what.

- Follow PEP 8

  → Use proper indentation, spacing, and naming style.

- Avoid repetition

  → Reuse logic using loops or functions (DRY principle).

- Consistent formatting

  → Makes your code easy to read for everyone.

Q8) Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

Ans:

1. **Integers (int)**

   Whole numbers (no decimal)

   Example: 10, -5

2. **Floats (float)**

   Numbers with decimals

   Example: 3.14, -2.5

3. **Strings (str)**

   Text data enclosed in quotes

   Example: "Hello", '123'

4. **Lists (list)**

   Ordered, changeable, allows duplicates

   Example: [1, 2, 3], ["apple", "banana"]

5. **Tuples (tuple)**

   Ordered, unchangeable, allows duplicates

   Example: (1, 2, 3)

6. **Dictionaries (dict)**

   Key-value pairs, unordered (Python 3.6+ keeps order)

Example: {"name": "Alice", "age": 25}

7. Sets (set)

Unordered, no duplicates

Example: {1, 2, 3}

Q9) Python variables and memory allocation.

Ans:

➢ **Variables in Python**

- A variable stores data like numbers, text, etc.
- You don't need to declare the type (Python is dynamically typed).

```
x = 10   # integer
name = "Alice"   # string
```

➢ **Memory Allocation**

- When you assign a value to a variable, Python stores it in memory.
- The variable name points to the memory location of the object.
- Multiple variables can point to the same object in memory.

```
a = 5
b = a  # b also points to the same memory as a
```

Q10) Python operators: arithmetic, comparison, logical, bitwise.

Ans:

1. **Arithmetic Operators (Math operations)**

   + Add, - Subtract, * Multiply, / Divide, // Floor divide, % Modulus, ** Power

2. **Comparison Operators (Compare values, return True/False)**

   == Equal, != Not equal, > Greater, < Less, >= Greater or equal, <= Less or equal

3. **Logical Operators (Combine conditions)**

   and Both true, or One true, not Reverse result

4. **Bitwise Operators (Work on bits)**

   & AND, | OR, ^ XOR, ~ NOT, << Left shift, >> Right shif1

Q11) Introduction to conditional statements: if, else, elif.

Ans:

1. **if Statement**

   Runs a block of code if condition is true.

   ```
   age = 18
   if age >= 18:
       print("You are an adult")
   ```

2. **else Statement**

Runs if condition is false.

```
age = 16
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

3. elif (else if)

Checks multiple conditions.

```
marks = 75
if marks >= 90:
    print("Grade A")
elif marks >= 60:
    print("Grade B")
else:
    print("Grade C")
```

Q12) Nested if-else conditions.

Ans:

➢ **Nested if-else**

An if-else inside another if-else.

Used to check multiple levels of conditions.

```
age = 20
if age >= 18:
```

```
        if age >= 60:
            print("Senior Citizen")
        else:
            print("Adult")
    else:
            print("Minor")
```

Q13) Introduction to for and while loops.

Ans:

➢ **Loops**

Used to repeat a block of code multiple times.

1. for loop

Used to iterate over a sequence (like list, string, range).

```
for i in range(5):
        print(i)
```

2. while loop

Repeats while a condition is true.

```
i = 0
while i < 5:
        print(i)
        i += 1
```

Q14) Using loops with collections (lists, tuples, etc.).

Ans:

Using Loops with Collections

- Python lets you loop through collections like:
- Lists, Tuples, Strings, Sets, Dictionaries.

1. Looping through a List :

```
fruits = ['apple', 'banana', 'mango']
for fruit in fruits:
        print(fruit)
```

2. Looping through a Tuple :

```
numbers = (1, 2, 3)
for num in numbers:
        print(num)
```

3. Looping through a Dictionary

```
student = {'name': 'Amit', 'age': 20}
for key in student:
        print(key, ":", student[key])
```

Q15) Understanding how generators work in Python.

Ans:

Generators are a special type of function that remember their state and return values

one at a time using yield.

- Use yield instead of return.
- Each time the generator is called, it resumes from where it left off.
- Efficient for large data (saves memory).

```
def my_gen():
    yield 1
    yield 2
    yield 3
for val in my_gen():
    print(val)
```

Q16) Difference between yield and return.

Ans:

- **Difference between yield and return:**

1. return is used to end a function and give back one value.
2. yield is used in a generator to return multiple values, one at a time.
3. return terminates the function completely.
4. yield pauses the function and resumes from the same point on next call.
5. return is used in normal functions.

6. yield is used in generator functions.
7. return stores all data in memory at once.
8. yield is memory-efficient; it gives data on demand.

Q17) Understanding iterators and creating custom iterators.

Ans:

- An iterator is an object that allows you to loop through a sequence of values, one at a time.

- An iterator must implement two methods:
  - __iter__() → returns the iterator object itself.
  - __next__() → returns the next item in the sequence.
  - Raises Stop Iteration when no more items.

Q18) Defining and calling functions in Python.

Ans:

- A function is a block of organized, reusable code that performs a specific task when called.

  Define a function :

```
def function_name(parameters):
        # function body
        # optional return value
```

```
        return result
```

Calling function :

```
function_name(arguments)
```

- A function is a reusable block of code that performs a specific task when called.
- A function in Python is defined using the def keyword followed by the function name and parentheses (). It may accept parameters and can return a value using the return statement.
- To execute a function, you use its name followed by parentheses. If the function requires arguments, you provide them inside the parentheses during the call.

Q19) Function arguments (positional, keyword, default).

Ans:

1. Positional Arguments

- Values passed in order.
- Example: func("Raj", 20)

2. Keyword Arguments

- Use parameter names to assign values.
- Example: func(name="Mahek", age=20)

3. Default Arguments

- Provide a default value in the function definition.

Example :

```
def greet(name="Friend"):
        print("Hello", name)
```

Q20) Scope of variables in Python.

Ans:

- Scope refers to where a variable can be accessed or used in your code.

1.Local Scope

- Variable declared inside a function.
- Only accessible within that function.

```
def my_func():
        x = 10  # local variable
        print(x)
```

2.Global Scope

- Variable declared outside all functions.
- Can be accessed anywhere in the program.

```
x = 5  # global variable
 def show():
        print(x)
```

3.Enclosed Scope (Nonlocal)

- Variable in a nested (inner) function, referring to the variable in the outer function.

```
def outer():
        x = "outer"
def inner():
        print(x)  # enclosed variable   inner()
```

4.Built-in Scope

- Includes Python's predefined functions and keywords.
- Example: print(), len(), etc.

Q20) Built-in methods for strings, lists, etc.

Ans:

- **String**
  - lower(), upper(): Change case
  - strip(): Remove spaces
  - replace(): Replace text
  - split(): Split into list
  - find(), count(): Search/count
- **List**
  - append(), insert(): Add items
  - remove(), pop(): Remove items
  - sort(), reverse(): Sort/reverse
  - extend(): Join lists
- **Tuple**
  - count(), index(): Count/find item
- **Dictionary**
  - get(): Get value
  - keys(), values(), items(): View data
  - update(): Add items
  - pop(): Remove item

➢ **Set**

- add(), remove(), discard(): Modiy items
- union(), intersection(): Set operations

Q21) Understanding the role of break, continue, and pass in Python loops.

Ans:

1. break

- Stops the loop immediately.

```
for i in range(5):
    if i == 3:
    break
    print(i)
```

2. continue

- Skips the current iteration and moves to the next.

```
for i in range(5):
    if i == 3:
        continue
        print(i)
```

3. pass

- Does nothing (placeholder statement).

```
for i in range(5):
    if i == 3:
    pass
    print(i)
```

Q22) Understanding how to access and manipulate strings.

Ans:

Accessing

- s[0]: First character
- s[-1]: Last character
- s[1:4]: Slice from index 1 to 3

Manipulation

- lower(), upper(): Change case
- strip(): Remove spaces
- replace(a, b): Replace text
- split(): Break into list
- len(s): Length of string

Q23) Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

Ans:

1. Concatenation

- Joining two or more strings using +.

```
s1 = "Hello"
s2 = "World"
result = s1 + " " + s2
print(result)  # Output: Hello World
```

2. Repetition

- Repeating a string using *.

```
text = "Hi "
print(text * 3)  # Output: Hi Hi Hi
```

3. upper()

- Converts all characters in the string to uppercase.

```
text = "hello"
print(text.upper())  # Output: HELLO
```

4. lower()

- Converts all characters in the string to lowercase.

```
text = "HELLO"
print(text.lower())  # Output: hello
```

5. replace(old, new)

- Replaces all occurrences of a substring (old) with another (new).

```
text = "good morning"
print(text.replace("morning", "night"))  # Output: good night
```

Q24) String slicing.

Ans:

- String slicing means extracting parts (substrings) of a string using index numbers.

  Syntax :

  string[start : end : step]

- start: index where slice starts (default is 0)
- end: index before which to stop (not included)
- step: how many characters to skip (default is 1)

- String slicing allows you to extract a portion (substring) of a string using indexing.
- It is useful when you want to access part of a string, like a word, character group,or to reverse it.
- Python strings are zero-indexed, meaning the first character has index 0.

Q25) How functional programming works in Python.

Ans:

- Functional programming is a style of programming that treats functions as first-class citizens — meaning you can assign them to variables, pass them as arguments, and return them from other functions.

1. Pure Functions

- Functions that depend only on inputs and have no side effects.

2. First-Class Functions

- Functions can be assigned to variables, passed as arguments, or returned from other functions.

3. Lambda Functions

- Small anonymous functions defined using the lambda keyword.

4. Higher-Order Functions

- Functions that take other functions as arguments or return functions.

5. Built-in Functional Tools

- map(): Apply function to all elements
- filter(): Filter elements by condition
- reduce(): Reduce list to a single value

- zip(): Combine multiple sequences

Q26) Using map(), reduce(), and filter() functions for processing data.

Ans:

1. **map(function, iterable)**
   - Applies a function to each element in an iterable.
   - Returns a new iterable (usually converted to a list).
   - Use: Transform or modify data items.
2. **filter(function, iterable)**
   - Filters elements for which the function returns True.
   - Returns a new iterable (usually converted to a list).
   - Use: Remove unwanted items based on condition.
3. **reduce(function, iterable)**
   - Repeatedly applies the function to the iterable to reduce it to a single value.
   - Requires importing from functools.
   - Use: Summing, multiplying, combining items into one result.

Q27) Introduction to closures and decorators.

Ans:

Closures

- A closure is a function defined inside another function that remembers variables from the outer function, even after the outer function is finished.
- It allows data to be retained across calls without using global variables.
- Use: When you want to create functions with private data or behavior.

Decorators

- A decorator is a function that modifies the behavior of another function without changing its code.
- It takes a function as input, wraps it with extra functionality, and returns it.
- Use: For logging, access control, timing, modifying inputs/outputs, etc.