Sr. no.	Topic	Duration (Mins)	Session No (2 Hours)	Session No (4 Hours)
1	Overview of Python	15	1	1
2	Features, Benefits, Uses of python	15		1
3	Installation and Setup of Python Environment	30	1	1
4	Python Console based application and Web Application using Flask	30	1	1
5	Deploying and Consuming Python Applications	30	1	1
6	Practical	60	2	1
7	Practical	60	2	1

Overview of Python

1. Overview of Python

Information

- Python is a high-level, interpreted, and general-purpose programming language known for its simplicity, readability, and versatility.
- Created by Guido van Rossum and first released in 1991, Python has since gained popularity among developers for its clean syntax, extensive standard library, and vast ecosystem of third-party packages.

1. Readability and Simplicity:
- Python's syntax emphasizes readability, making it easier to write and understand code.
- It uses indentation and whitespace to delimit code blocks, avoiding the need for explicit braces or semicolons.
- This promotes clean and structured code.

2. Interpretation and Portability:
- Python is an interpreted language, meaning that it does not require explicit compilation before running.
- Python programs are executed by the Python interpreter, which interprets the code line by line.

- This makes Python highly portable across different platforms and operating systems.

3. Dynamic Typing and Memory Management:
- Python is dynamically typed, allowing variables to be assigned different data types at runtime.
- It also features automatic memory management through garbage collection, relieving developers from manual memory allocation and deallocation.

4. Extensive Standard Library:
- Python provides a comprehensive standard library with a wide range of modules and packages covering areas such as file I/O, networking, database access, string manipulation, regular expressions, and more.
- The standard library enables developers to accomplish various tasks without needing external dependencies.

5. Large Ecosystem of Third-Party Packages:
- Python has a rich ecosystem of third-party packages and libraries, such as NumPy, Pandas, Django, Flask, TensorFlow, and many more.
- These packages extend Python's capabilities for data analysis, web development, machine learning, scientific computing, and various other domains.

6. Object-Oriented Programming (OOP) Support:
- Python supports object-oriented programming paradigms, allowing the creation of classes, objects,

and the implementation of encapsulation, inheritance, and polymorphism.

7. Scripting and Rapid Prototyping:
- Python is often used as a scripting language for automating tasks, creating scripts, and writing prototypes quickly.
- Its concise syntax and high-level abstractions make it well-suited for rapid development and prototyping.

8. Community and Resources:
- Python has a vibrant and supportive community of developers, offering extensive documentation, tutorials, forums, and online resources.
- The community-driven nature of Python fosters collaboration, knowledge-sharing, and continuous improvement of the language and its ecosystem.

- Python is used in a wide range of applications, including web development, data analysis, machine learning, scientific computing, network programming, automation, and more.
- Its versatility, ease of use, and extensive libraries make it a popular choice for both beginner programmers and experienced developers alike.

2. Features, Benefits, Uses of Python

Information

A. Features of Python:

1. Readability:
- Python emphasizes code readability and uses a clean and easy-to-understand syntax, making it more readable than many other programming languages.

2. Simplicity:
- Python's simple syntax and easy-to-learn nature make it an ideal language for beginners.
- It promotes code clarity and reduces the learning curve.

3. Interpretation:
- Python is an interpreted language, allowing for quick and interactive development.
- Developers can write and execute code line by line, making it suitable for rapid prototyping and scripting.

4. Dynamic Typing:
- Python is dynamically typed, meaning variable types are determined at runtime.

- This flexibility allows for faster development and easier code maintenance.

5. Vast Standard Library:
- Python has a comprehensive standard library, offering a wide range of modules and packages for various tasks like file I/O, networking, database access, regular expressions, and more.
- The standard library reduces the need for external dependencies.

6. Large Ecosystem:
- Python has a vast ecosystem of third-party packages and libraries available through the Python Package Index (PyPI).
- These packages extend Python's functionality for specific domains like data science, web development, machine learning, scientific computing, and more.

7. Cross-platform Compatibility:
- Python code can run on different platforms and operating systems, including Windows, macOS, Linux, and more, making it highly portable.

B. Benefits of Python:

1. Productivity:
- Python's simplicity and high-level abstractions enable developers to write code faster and with

fewer lines compared to other programming languages, increasing productivity.

2. Flexibility:
- Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming, offering flexibility in coding styles and problem-solving approaches.

3. Easy Integration:
- Python can easily integrate with other languages like C, C++, Java, and more, allowing developers to leverage existing code and libraries.

4. Community and Support:
- Python has a large and active community of developers worldwide.
- It provides extensive documentation, tutorials, forums, and online resources, making it easy to find help and support when needed.

C. Uses of Python:

1. Web Development:
- Python offers frameworks like Django and Flask for building scalable and feature-rich web applications.
- It is used to develop server-side code, handle HTTP requests, and interact with databases.

2. Data Analysis and Visualization:

- Python, along with libraries such as NumPy, Pandas, and Matplotlib, is widely used for data analysis, manipulation, and visualization.
- It is popular in data science, machine learning, and scientific computing.

3. Scripting and Automation:
- Python's scripting capabilities make it suitable for automating tasks, writing scripts, and performing system administration tasks.

4. Machine Learning and Artificial Intelligence:
   - Python, with libraries like TensorFlow, PyTorch, and scikit-learn, is extensively used for machine learning, deep learning, and artificial intelligence applications.

5. Scientific Computing:
   - Python, along with libraries like SciPy and NumPy, provides a powerful environment for scientific computing, simulations, and numerical computations.

6. Prototyping and Rapid Development:
   - Python's simplicity and quick development cycle make it ideal for prototyping new ideas and building proof-of-concept applications.

7. Desktop Applications:
   - Python can be used to develop cross-platform desktop applications using frameworks such as PyQt and Tkinter.

- Python's versatility and rich ecosystem of libraries make it applicable to a wide range of domains, from web development to data science, machine learning, scientific computing, automation, and more.

3.Installation and Setup of Python Environment

Information

To install and set up a Python environment, follow these steps:

1. Download Python:
- Visit the official Python website at www.python.org and navigate to the "Downloads" section. Choose the appropriate Python version for your operating system (Windows, macOS, or Linux) and download the installer.

2. Run the Installer:
- Locate the downloaded Python installer and run it.
- Follow the prompts in the installation wizard to install Python on your computer.
- During the installation, you may have the option to customize the installation directory and add Python to the system PATH (recommended).

3. Verify Installation:
- Open a command prompt or terminal and type python --version to verify that Python is installed correctly.
- You should see the version number of Python displayed.

4. Install an Integrated Development Environment (IDE):

- While not mandatory, an IDE can enhance your Python development experience.
- There are several popular options available, such as PyCharm, Visual Studio Code, Anaconda, and IDLE (which comes bundled with Python).

5. Configure the IDE (optional):

- If you decide to use an IDE, follow the specific instructions for configuring the environment.
- This may include setting the Python interpreter path, configuring project settings, and installing any necessary plugins or extensions.

6. Update Package Manager (pip):

- Pip is the default package manager for Python.
- It allows you to install, upgrade, and manage Python packages.
- To update pip to the latest version, open a command prompt or terminal and run the following command:

```
python -m pip install --upgrade pip
```

7. Install Packages:

- Python offers a vast ecosystem of third-party packages that extend its functionality.
- You can install packages using pip.
- For example, to install the popular NumPy package, run the following command:

```
pip install numpy
```

8. Create and Run a Python Script:
   - Create a new file with a .py extension and write some Python code.
   - For example, you can create a file named hello.py and add the following code:


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

- You should see the output "Hello, Python!" displayed.

- With these steps, you have successfully installed and set up a Python environment on your computer.
- You can now start writing and running Python code using your preferred text editor or IDE.
- Remember to explore the rich Python ecosystem and utilize packages to enhance your development experience and extend the capabilities of your Python programs.

4.Python Console based application and Web Application using Flask

Information

Flask :

- Flask is a lightweight and flexible web framework for Python.
- It is designed to make web development quick and easy, with a focus on simplicity and extensibility.
- Flask is known for its minimalistic approach, allowing developers to have greater control over the structure and components of their web applications.

To develop a Python console-based application and a web application using Flask, follow these steps:

1. Developing a Python Console-based Application:

- Determine the functionality and purpose of your console-based application.
- Write the Python code to implement the desired functionality. You can use any text editor or integrated development environment (IDE) to write your code.
- Save the Python code in a '.py' file.

- Open a terminal or command prompt and navigate to the directory where the file is saved.
- Run the Python code by executing the following command:

```
python filename.py
```

- Replace filename with the actual name of your Python file.

- Test and verify the output of your console-based application in the terminal or command prompt.

Here's an example of a Python console-based application and a web application using the Flask framework.

1. Example Python Console-based Application:

```
def greet(name):
    print(f"Hello, {name}! Welcome to the Console
Application.")

def main():
    name = input("Enter your name: ")
    greet(name)

if __name__ == "__main__":
    main()
```

- In this example, the greet function takes a name as input and prints a greeting message to the console.
- The main function prompts the user for their name and calls the greet function.
- When executed, the application will ask for the user's name and display a personalized greeting message.

2. Developing a Web Application using Flask:

- To create a web application using Flask, follow these steps:

1. Install Flask by running the following command in your terminal or command prompt:

```
Pip install flask
```

2. Create a new Python file, e.g., app.py, and add the following code:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, Flask!"

@app.route("/greet", methods=["GET", "POST"])
def greet():
    if request.method == "POST":
    name = request.form.get("name")
    return f"Hello, {name}! Welcome to the Flask Web
Application."
 else:
    return render_template("greet.html")

if __name__ == "__main__":
      app.run()
```

- We define two routes: '/' for the homepage that returns a simple greeting message, and '/greet' that handles GET and POST requests.
- When accessed via a web browser, the '/greet' route displays a form where the user can enter their name.
- On submission, it returns a greeting message.

3. Create a new directory called templates in the same location as your 'app.py' file.

- Inside the 'templates' directory, create a new HTML file named 'greet.html' and add the following code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Greet Form</title>
</head>
<body>
  <h1>Greet Form</h1>
  <form method="POST" action="/greet">
    <label for="name">Enter your name:</label>
    <input type="text" name="name" id="name"
required>
    <button type="submit">Greet</button>
  </form>
</body>
</html>
```

- Save the files, and in your terminal or command prompt, navigate to the directory where your app.py file is located.

4. Run the Flask application by executing the following command:

```
python app.py
```

- Open a web browser and enter the URL 'http://localhost:5000'.

- You should see the greeting message or the form to enter your name.

- That's it! You have created a Python console-based application and a web application using Flask.
- Feel free to customize the code according to your requirements and explore more Flask features for building robust web applications.

Compilation and Execution of Python based application:

- In Python, code is typically executed directly without an explicit compilation step.
- Python is an interpreted language, meaning that the Python interpreter reads the source code line by line and executes it on-the-fly.
- However, there are still a few steps involved in running a Python-based application:

1. Write the Python Code:
- Create a new file with a .py extension and write your Python code using a text editor or an integrated development environment (IDE).

2. Save the Python File:

- Save the file with a meaningful name and the .py extension. For example, my_program.py.

3. Open a Terminal or Command Prompt:
- Open a terminal or command prompt on your computer.

4. Navigate to the Directory:
- Use the cd command to navigate to the directory where your Python file is saved.
- For example, if your file is saved in the Documents folder, you can navigate to it by running:

```
cd Documents
```


5. Run the Python File:
- In the terminal or command prompt, execute the Python file by typing python followed by the name of the Python file.
- For example:

```
python my_program.py
```

6. Observe the Output:
- The Python interpreter will execute the code in the file, and any output generated by the program will be displayed in the terminal or command prompt.

- Note: Make sure you have Python installed on your system and that it is accessible from the command line. You can verify the installation and version of Python by running python --version in the terminal or command prompt.

- If your Python program has dependencies on external libraries, you may need to install them using the pip package manager before running your program.
- To install a package, you can use the following command:

```
pip install package_name
```

- Replace ‘package_name’ with the name of the package you want to install.

- Remember to save your Python file after making any changes and rerun it to observe the updated output.

- That's it! You can now compile and execute your Python-based application by following these steps.

5. Deploying and Consuming Python Applications

Information

Deploying and consuming Python applications typically involves the following steps:

1. Package the Application:
- Prepare your Python application for deployment by organizing the necessary files and dependencies.
- Create a directory or archive that contains your application code, configuration files, and any required dependencies.

2. Choose a Deployment Platform:
- Decide where you want to deploy your Python application.
- Common deployment platforms include local servers, cloud platforms (such as AWS, Google Cloud, or Azure), and containerization platforms (such as Docker).

3. Setup the Deployment Environment:
- Set up the deployment environment by installing the required dependencies and ensuring the necessary runtime environment is available.
- This includes installing Python and any additional libraries or frameworks needed by your application.

4. Deploying to Local Server:
- If you're deploying to a local server, copy your application directory or archive to the appropriate location on the server.
- Configure the server to run your application using the appropriate web server software, such as Apache or Nginx, and configure the necessary routing and permissions.

5. Deploying to Cloud Platform:
- If you're deploying to a cloud platform, follow the platform's documentation and guidelines for deploying Python applications.
- This typically involves creating a deployment package, configuring the platform-specific settings, and using their tools or APIs to deploy the application to the cloud.

6. Testing and Monitoring:
- Test your deployed application to ensure it functions as expected.
- Set up monitoring tools to track the application's performance, logs, and any potential issues.

7. Consuming the Application:
- To consume a deployed Python application, you need to interact with its exposed endpoints or user interfaces.
- This can involve making HTTP requests to the application's APIs, accessing web pages hosted by

the application, or using other communication mechanisms provided by the application.

8. Handling Application Updates:

- As you make updates or enhancements to your Python application, follow a similar deployment process to deploy the updated version.
- Ensure proper version control, backup mechanisms, and strategies for handling database migrations or data updates.

- Remember that the specific details of deploying and consuming a Python application can vary depending on the deployment platform, web framework, or architecture you are using.
- It's important to refer to the documentation and guidelines specific to your chosen deployment environment to ensure a smooth deployment and consumption process.

Data Types:

- In Python, there are several built-in data types that you can use to store and manipulate different kinds of data.

1. Numeric Types:

- int: Represents integers, such as 1, 100, -10, etc.
- float: Represents floating-point numbers with decimal places, such as 3.14, 2.5, -0.5, etc.

- complex: Represents complex numbers in the form a + bj, where a and b are real numbers and j represents the imaginary unit.

2. Boolean Type:

- bool: Represents Boolean values, which can be either 'True' or 'False'.
- Boolean values are often used for logical comparisons and control flow in Python.

3. Sequence Types:

a. str:
- Represents a sequence of characters enclosed in single quotes ('') or double quotes ("").
- Strings are used to store textual data.

b. list:
- Represents an ordered collection of items enclosed in square brackets ([]).

- Lists can contain elements of different data types and are mutable, meaning that you can modify their contents.

c. tuple:
- Similar to lists, tuples represent an ordered collection of items enclosed in parentheses (()).
- However, unlike lists, tuples are immutable, meaning that their elements cannot be changed once defined.

4. Mapping Type:

a. dict:

- Represents a collection of key-value pairs enclosed in curly braces ({}).
- Dictionaries are used to store data in the form of key-value mappings.
- The keys must be unique, and the values can be of any data type.

5. Set Types:

a. set:

- Represents an unordered collection of unique elements enclosed in curly braces ({}).
- Sets are useful for performing mathematical set operations like union, intersection, and difference.

b. frozenset:

- Similar to sets, frozensets are immutable versions of sets.
- Once defined, the elements in a frozenset cannot be modified.

6. None Type:

a. None:

- Represents a special value indicating the absence of a value or the null value.
- It is commonly used to denote the absence of a return value from a function or as an initial/default value for variables.

- These are some of the primary data types in Python.

- Python is a dynamically-typed language, which means that you don't need to explicitly declare the data type of a variable before using it.
- Python infers the data type based on the value assigned to the variable.
- However, you can also explicitly assign a data type to a variable if needed.

- In addition to these built-in data types, Python also supports user-defined data types through classes and objects, allowing you to create custom data structures and encapsulate data and behavior within objects.

Classes:

- In Python, classes are a fundamental concept of object-oriented programming (OOP).
- A class is a blueprint or template for creating objects that share common attributes (data) and behaviors (methods).
- Objects are instances of a class, and they encapsulate data and functionality within a single entity

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def say_hello(self):
        print(f"Hello, my name is {self.name} and I am
{self.age} years old.")

# Create an instance of the Person class
person1 = Person("John", 30)

# Access the attributes and call the method of the
person1 object
print(person1.name)  # Output: John
print(person1.age)   # Output: 30
person1.say_hello()  # Output: Hello, my name is John
and I am 30 years old.
```

- To create an instance of the Person class, we use the class name followed by parentheses, similar to calling a function.
- We assign the newly created object to the variable person1.
- We can then access the attributes (name and age) and call the method (say_hello) using dot notation (object.attribute or object.method()).

- Classes provide a way to organize and structure code by grouping related data and behaviors together.
- They promote code reusability, modularity, and encapsulation.
- Through inheritance, classes can inherit attributes and methods from other classes, enabling code reuse and creating hierarchies of related classes.

- By defining and instantiating objects from classes, you can create complex systems with multiple interacting objects that model real-world entities or abstract concepts.

- Note: The self parameter in the class methods refers to the instance of the class itself.
- It is a convention in Python to use self as the first parameter in class methods, but you can use any name you prefer.

Practical :

1. Create a Console based Python Application:

Certainly! Here's an example of a console-based Python application that calculates the area of a circle:

```
import math

def calculate_circle_area(radius):
    area = math.pi * radius**2
    return area

def main():
    print("Circle Area Calculator")
    radius = float(input("Enter the radius of the circle: "))
    area = calculate_circle_area(radius)
    print("The area of the circle is:", area)

if __name__ == "__main__":
    main()
```

- In this example, we define a function called 'calculate_circle_area' that takes the radius of a circle as input and calculates its area using the formula 'pi * radius^2', where 'pi' is a constant value provided by the 'math' module.

- The ‘math’ function serves as the entry point of the application.
- It prompts the user to enter the radius of the circle, calls the ‘calculate_circle_area’ function, and then prints the calculated area to the console.

- To run this console-based application, save the code in a Python file, such as ‘circle_area_calculator.py’.
- Open a terminal or command prompt, navigate to the directory containing the Python file, and execute the following command:

```
python circle_area_calculator.py
```


- The application will prompt you to enter the radius of the circle.
- After providing the input, it will display the calculated area on the console.

- Feel free to modify and enhance this example according to your needs.
- You can add error handling, additional calculations, or expand the functionality as desired.

2. Create a Web Application using Flask

- Certainly! Here's an example of a simple web application using Flask:
- First, make sure you have Flask installed.
- You can install it using pip:

```
pip install flask
```

Create a new Python file, e.g., app.py, and add the following code:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello, Flask!'

@app.route('/greet', methods=['GET', 'POST'])
def greet():
    if request.method == 'POST':
        name = request.form.get('name')
        return render_template('greet.html', name=name)
    else:
        return render_template('greet.html')

if __name__ == '__main__':
    app.run()
```

- In this example, we import the Flask class from the flask module and create a Flask application instance named app.

- The @app.route('/') decorator associates the root URL (/) with the index function.

- When a user visits the root URL, the index function is executed, and it returns the string 'Hello, Flask!' as the response.

- The @app.route('/greet', methods=['GET', 'POST']) decorator associates the /greet URL with the greet function. This function handles both GET and POST requests.

- If the request method is POST, it retrieves the value of the 'name' field from the submitted form and renders the 'greet.html' template, passing the name as a variable.

- If the request method is GET, it simply renders the 'greet.html' template without any name.

- The render_template function is used to render HTML templates.

- In this case, we have a template file named 'greet.html', which will be rendered when the /greet URL is visited.

- It contains a form to input a name and display a greeting message.

- To run the application, open a terminal or command prompt, navigate to the directory containing app.py, and run the following command:

```
python app.py
```

- You should see the Flask development server start, and it will provide you with a URL (usually 'http://127.0.0.1:5000/').
- Open that URL in your web browser, and you will see the "Hello, Flask!" message displayed.
- Visit 'http://127.0.0.1:5000/greet' to access the greeting form.

- This is a basic example of a Flask web application.
- You can expand on it by adding more routes, templates, and functionality according to your requirements.
- Flask provides a flexible and powerful framework for building web applications in Python.

StarAgile