DAY 2

Virtual Environments in Python :

A **virtual environment (venv)** in Python is an isolated workspace that allows you to install dependencies specific to a project without affecting the global Python environment.

Python Comments:

Comments in Python are used to make code more readable and to provide explanations. They are ignored by the interpreter and do not affect execution.

1. Single-Line Comments

Use the # symbol to add a comment on a single line:

python
CopyEdit
```
# This is a single-line comment
print("Hello, World!")  # Comment at the end of a line
```

---

2. Multi-Line Comments (Using #)

Python does not have a built-in multi-line comment syntax, but you can use multiple # symbols:

python
CopyEdit
```
# This is a multi-line comment
# spanning multiple lines
print("Hello, Python!")
```

---

3. Multi-Line String as a Comment

A triple-quoted string (`"""` or `'''`) can act as a multi-line comment when not assigned to a variable:

python
CopyEdit
```python
"""
This is a multi-line string
used as a comment.
"""
print("Python is fun!")
```

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

```
a = 33

b = 200

if b > a:

  print("b is greater than a")
```

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

```
i = 1

while i < 6:

  print(i)
  i += 1
```

Python For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

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

for x in fruits:

  print(x)
```

Python functions –

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"

car2 = "Volvo"

car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

```
class Person:

  def __init__(self, fname, lname):

    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Polymorphism-

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Function Polymorphism

An example of a Python function that can be used on different objects is the `len()` function.

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: `Car`, `Boat`, and `Plane`, and they all have a method called `move()`:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")
```

```
car1 = Car("Ford", "Mustang")       #Create a Car object

boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object

plane1 = Plane("Boeing", "747")     #Create a Plane object


for x in (car1, boat1, plane1):

  x.move()
```

Regular Expressions (Regex) in Python

Regular expressions (regex) are powerful tools for searching, matching, and manipulating text based on patterns. Python provides the `re` module to work with regular expressions.

RegEx Functions

The `re` module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match

sub	Replaces one or many matches with a string

PEP in Python Coding Standards

PEP stands for **Python Enhancement Proposal**. These are design documents that provide guidelines and best practices for Python development. The most well-known PEP related to coding standards is:

PEP 8 – Style Guide for Python Code

PEP 8 is the official style guide for Python code, ensuring consistency and readability. Key principles include:

- **Indentation:** Use **4 spaces per indentation level** (avoid tabs).
- **Maximum Line Length:** Limit lines to **79 characters** (72 for docstrings).
- **Blank Lines:** Use **two blank lines** before top-level functions and classes, and **one blank line** inside functions to separate logic.
- **Imports:**
    - Use **one import per line** (avoid `import os, sys`).
    - Follow the order: **standard library imports, third-party imports, local application imports**.
- **Naming Conventions:**
    - Variables & functions: `lower_case_with_underscores`
    - Constants: `UPPER_CASE_WITH_UNDERSCORES`
    - Classes: `CamelCase`
- **Whitespace Usage:**
    - Avoid extra spaces inside parentheses, brackets, or braces.
    - Use a single space **before and after** assignment (`x = 5`, not `x=5`).
- **Docstrings:**
    - Use triple quotes (`"""docstring"""`) for documentation.
    - First line should be a summary, followed by details if necessary.

Docstrings in Python

Docstrings are a special type of comment used to document Python functions, classes, and modules. They are written in between triple quotes (`"""` or `'''`) and can be accessed programmatically to generate documentation.

Docstrings: Used to document code for future reference and provide a description of functions, classes, and modules. They help in understanding the code's behavior and are often used to generate documentation.

Comments: Provide **inline or block notes** for the code to explain specific sections, logic, or details for human understanding. They are primarily for readability and clarity during development.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `else` block lets you execute code when there is no error.

The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

```
try:

  print(x)

except:

  print("An exception occurred")
```

Finally

The `finally` block, if specified, will be executed regardless if the try block raises an error or not.

```
y:

  print(x)

except:
  print("Something went wrong")

finally:

  print("The 'try except' is finished")
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the `raise` keyword.

```
x = -1



if x < 0:

  raise Exception("Sorry, no numbers below zero")
```

File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

`"x"` - Create - Creates the specified file, returns an error if the file exists

Wrapper Functions in Python

A **wrapper function** is a function that calls another function while adding some additional behavior before, after, or around the call. Wrapper functions are commonly used in **decorators**, **logging**, **authentication**, and **performance monitoring**.

Logging in Python

Logging is essential for debugging, monitoring, and tracking application behavior. Python's built-in `logging` module provides a flexible way to log messages at different severity levels.

Python's `logging` module provides five levels of severity:

Level	**Value**	**Description**
`DEBUG`	10	Detailed information, useful for debugging
`INFO`	20	Confirmation that things are working as expected
`WARNING`	30	Indication of potential problems
`ERROR`	40	Serious issue preventing some functionality

CRITICAL	50	Severe error that may crash the program

Ruff: A Fast Python Linter & Formatter

Ruff is a fast, Python-based linter and formatter designed to enforce coding standards and improve code quality. It serves as an alternative to tools like **Flake8, Black, and isort,** offering high performance and broad rule coverage.

Agile & Scrum: Overview and Key Concepts

Agile is a project management approach focused on **flexibility, collaboration, and continuous improvement,** while **Scrum** is a framework within Agile that structures teamwork using defined roles, events, and artifacts.

1. What is Agile?

Agile is an **iterative and incremental** software development methodology that emphasizes:
✅ Customer collaboration over contract negotiation
✅ Responding to change over following a fixed plan
✅ Working software over comprehensive documentation
✅ Individuals & interactions over rigid processes

🔹 Agile was introduced in the **Agile Manifesto (2001)** and includes frameworks like **Scrum, Kanban, XP (Extreme Programming), and SAFe (Scaled Agile Framework).**

---

2. What is Scrum?

Scrum is a lightweight **Agile framework** that enables teams to deliver high-value products through:

- 🔹 **Short iterations (Sprints)** (typically 1-4 weeks)
- 🔹 **Regular feedback loops**
- 🔹 **Defined roles and responsibilities**
- 🔹 **Continuous improvement through retrospectives**

🔹 Scrum Workflow

1️⃣ **Product Backlog** → List of all tasks/features (managed by Product Owner)
2️⃣ **Sprint Planning** → Selecting backlog items for the next sprint
3️⃣ **Sprint Execution** → Team develops features in a time-boxed sprint (1-4 weeks)
4️⃣ **Daily Scrum (Standup)** → 15-min daily meeting to track progress
5️⃣ **Sprint Review** → Team presents completed work to stakeholders
6️⃣ **Sprint Retrospective** → Discussion on what went well & what to improve

---

3. Scrum Roles

Role	**Responsibility**
Product Owner	Defines product vision, prioritizes backlog, ensures value
Scrum Master	Facilitates Scrum process, removes blockers, coaches team

Development Team	Self-organizing group that builds the product

---

4. Scrum Artifacts

✅ **Product Backlog** – List of all features, enhancements, and bug fixes
✅ **Sprint Backlog** – Tasks selected for the current sprint
✅ **Increment** – A potentially shippable product version

---

5. Agile vs. Scrum: Key Differences

Aspect	**Agile**	**Scrum**
Definition	A broad methodology	A specific Agile framework
Flexibility	High flexibility	Structured process
Iterations	Continuous, flexible iterations	Fixed-length Sprints
Roles	No fixed roles	Defined roles (PO, SM, Dev Team)

Meetings	As needed	Daily standups, Sprint Planning, Reviews, Retrospectives

---

6. Scrum Best Practices

✅ Keep sprints **short and focused** (1-2 weeks recommended)
✅ Ensure **regular feedback** from stakeholders
✅ Use **burn-down charts** to track progress
✅ Prioritize backlog using **MoSCoW (Must, Should, Could, Won't)**
✅ Encourage a **collaborative, self-organizing team**

---

7. Tools for Agile & Scrum

🛠️ **JIRA, Trello, Azure DevOps, ClickUp, Monday.com** – Used for tracking sprints & backlog
🛠️ **Miro, Lucidchart** – Visual collaboration tools

Why Project Management is Crucial 📊

Project management is an essential discipline for ensuring that projects are completed efficiently, on time, and within budget. It involves planning, organizing, and overseeing the completion of a project, ensuring that it meets its goals and satisfies stakeholders. Here's why project management is crucial:

. Clear Goals and Objectives

Efficient Resource Utilization

Risk Management

Budget Control

Quality Assurance

Meeting Deadlines

Successful Project Delivery

The Need for Project Management 🛠️

Project management is an essential discipline that provides structure, control, and direction to projects, ensuring they are completed successfully. In today’s fast-paced and often unpredictable work environments, the need for effective project management is more critical than ever. Here’s why project management is crucial:

Unit Testing and Validation Checks 🧪

Unit testing and validation checks are two crucial components of software development that ensure code correctness, reliability, and quality. Here’s an overview of each:

---

1. Unit Testing

Definition:

Unit testing involves testing individual components (usually functions or methods) of a software program in isolation to ensure they behave as expected. It helps identify bugs early in the development process, making it easier to maintain and scale the codebase.

Why Unit Testing is Important:

- **Early Bug Detection:** Unit tests catch bugs early in the development cycle, reducing the cost of fixing issues later.
- **Improved Code Quality:** Writing unit tests forces developers to think about edge cases and design considerations, improving code quality.

- **Refactoring Confidence:** With unit tests in place, you can confidently refactor the code without fear of breaking existing functionality.
- **Regression Testing:** Unit tests act as a safety net when making changes, ensuring that previously working code continues to function correctly.

Popular Unit Testing Frameworks:

1. **unittest:** Built into Python, it provides a standard framework for writing and running tests.
2. **pytest:** A popular and more flexible testing framework that supports fixtures, parametrized tests, and better error reporting.
3. **nose2:** A successor to Nose, it provides easy test discovery and execution.

2. Validation Checks

Definition:

Validation checks refer to the process of verifying that the input data, outputs, or even intermediate results of a program are accurate, reliable, and adhere to the required constraints. Validation ensures that the system behaves as expected under different conditions and that it meets the user requirements.

Why Validation is Important:

- **Data Integrity:** Ensures that the inputs and outputs of the system conform to expected formats, ranges, and constraints.
- **Error Prevention:** Identifies potential errors or inconsistencies before they cause larger issues, such as crashes or incorrect computations.
- **Compliance:** Ensures that the system adheres to regulatory standards or business rules.

- **User Satisfaction:** Prevents incorrect or faulty data from being processed, leading to better user experience.

Mascot Principle

The **Mascot Principle** is a concept from **business and marketing** that involves creating a representative character or symbol (mascot) to embody and communicate the values, mission, or identity of a brand, organization, or product. Mascots are typically used to **engage** audiences, **humanize** a company, and create an **emotional connection** with customers.