

Today’s Notes

# Date: 04/02/2025

## Regarding Python….

Subtopics:

Global variables are **not used often** in Python because:

1. **Hard to Track** – Multiple functions modify them, causing confusion.
2. **Difficult to Debug** – Errors are harder to trace.
3. **Less Reusable Code** – Functions relying on them can't be reused easily.
4. **Multi-Threading Issues** – Can cause unexpected behavior.

💡 **Use local variables or function arguments instead!**

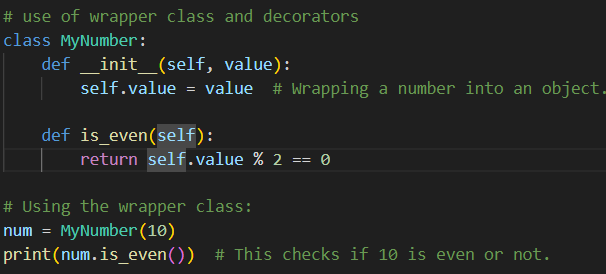
**What are four imp terms in error handling?**

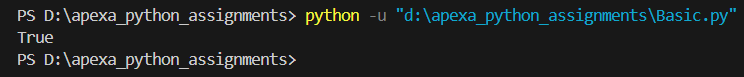
1. try: Used to write the code that might cause an error.
2. except: Catches and handles the error if it occurs
3. finally: Runs **no matter what** (whether an error happens or not).
4. raise: Manually triggers an error when needed.

What is the wrapper class and decorators?

In programming, a **wrapper class** is like **adding features** to something that already exists (like a number, string, etc.)

So basically with the help of wrapper class we can treat variable as object so to assign multiple functionalities to it!

.



Decorators are used to:

* Add **extra behavior** to a function without modifying it.
* **Reuse** code for multiple functions.
* Make code **cleaner and more organized**.

**How to identify the decorator ?**

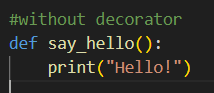
1. Decorator function is that which takes another function as an argument .

### Nested Function (Wrapper) Inside

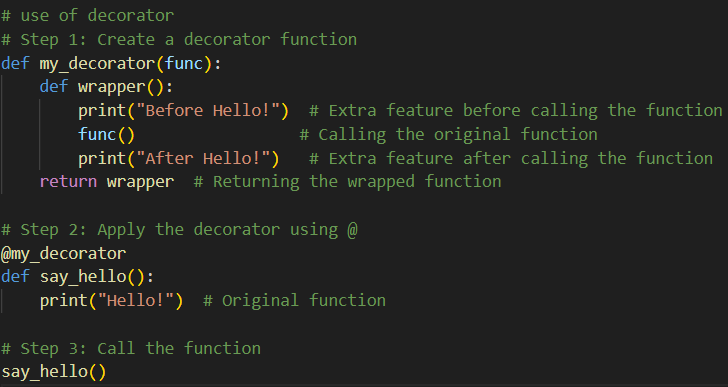
### Return the Wrapper Function

### The @decorator\_name Syntax

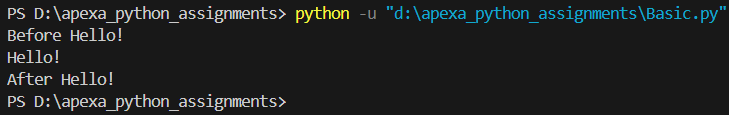
Example: If we are printing simple **hello** as output by calling say\_hello() we get,



Now if we want to add some lines before and after that also so still being called to say\_hello() we get,



**Output**:



**About Oops in Python…**

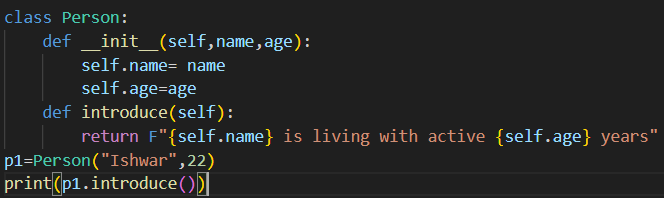
1. **Class: Class is the template which is used to create multiple objects. Each object created using this class can access the features of the same class.**

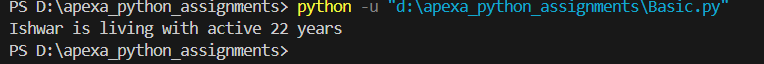
**Attributes are the variables in a class and represent the properties of a class.**

1. **Object: It is the instance of a class. It can have its own data.**

**Use of \_\_init\_\_()?**

Every time when an object is created it must need some attributes which are basically properties for proper working of objects. So everytime you need to create it manually but with the help of \_\_init\_\_ you can automate this tedious process, which automatically does all the necessary things required for initializing an object.





**!!Pending coding work for encapsulation, polymorphism and inheritance….**

**Threading and Multithreading:** Thread is a separate flow of execution and every thread has a task. Multithreading refers to doing the same or different tasks at the same time which are part of the same application. (eg. SBI portal is being used by several users and employees at same time.)

## Regarding Project Management…

## Python coding standards and various principles as per Industry..

Coding standards are guidelines that help developers write clean, readable, and maintainable code. For Python, the standards are defined by **PEP 8** (Python Enhancement Proposal 8), which is the official style guide for Python code.

1. **Naming Conventions**

Variable, Function and Method - **snake\_case**

Class name **- Pascal case or Camel case**

Constants **- UPPERCASE**

### **Indentation and Spacing**

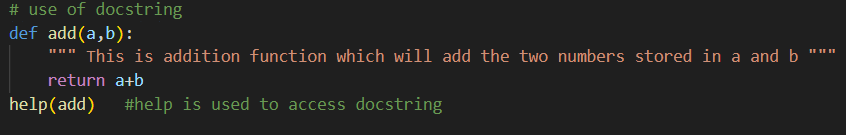
Indentation of 4 spaces

**2 blank lines** before defining a function or class

1. **Documentation and Comments**

* **Use of docstring and comment at necessary places (For docstring keep it 72)**

What is the difference between **docstring** and **comments** in python?



| **Feature** | **Docstring** | **Comment** |
| --- | --- | --- |
| **Definition** | Describes functions, classes, or modules. | Explains code in a line. |
| **Syntax** | """ """ | # |
| **Length** | Can be multi-line | Single-line |
| **Purpose** | Used for documentation. | Clarifies code logic. |
| **Access** | Accessed using help(). | Not accessible via help(). |
| **Example** | """Adds two numbers.""" | # Adds two numbers |

## 

## 

## 

## 

## 

* Keep line length less than 79 characters long
* Avoid Global variables
* Keep the variable name as easy as the user can understand.
* Use of \_\_init\_\_ () - Every time when an object is created it must need some attributes which are basically properties for proper working of objects. So everytime you need to create it manually but with the help of \_\_init\_\_ you can automate this tedious process, which automatically does all the necessary things required for initializing an object.

## Moscow Principle in Python

* **M**ust Have - These are the features or tasks that are **essential** and cannot be ignored.
* **S**hould Have - These are important, but **not as critical** as the "Must Have" items.
* **C**ould Have - These are **nice to have** features but not critical.
* **W**on't Have - These are features or tasks that are **not necessary** and will not be included in the current project or timeline.

How to keep remembering these guidelines manually is slightly difficult.

# Date: 06/02/2025

## Day 4) Assigned topics: You will get to know more about following topics

## Backward compatibility?

## Today’s Task: Create the python script for accessing the freely available API and get the data in json format and then finally test it using pytest

## REST APIs

API stands for application programming interface in which the rest is one of popular API called Representational State Transfer.

**Important terms 👍**

1. Endpoint: An **endpoint** is a URL where the API lives.

Example: https://api.example.com/users

### **HTTP Methods (CRUD Operations)**

These are the actions you can perform on data:

* **GET** → Read data (like viewing a menu).
* **POST** → Create data (like placing an order).
* **PUT/PATCH** → Update data (like changing your order).
* **DELETE** → Remove data (like canceling an order).

### 

### **Request & Response**

* **Request**: What you send to the server.
* **Response**: What the server sends back.

Response is usually in **JSON** format:

json

{

"id": 1,

"name": "John Doe",

"email": "john@example.com"

}

1. **Status Codes**

Servers reply with status codes to tell you what happened:

* **200 OK** → Success!
* **201 Created** → New resource created.
* **400 Bad Request** → Your request is wrong.
* **401 Unauthorized** → You’re not allowed without logging in.
* **404 Not Found** → URL is wrong.
* **500 Server Error** → Problem on the server side.
* **\*301 Moved Permanently** — Redirect

1. **Authentication and Authorization**

* Authentication : Who Are You?
* Authorization : What Are You Allowed to Do?

**6. XML and JSON**

**XML:** eXtensible Markup Language

XML is a **markup language** designed to store and transport data.

**Looks Like:** Similar to HTML with **tags** to define data.

#### **Example:** xml

<user>

<id>101</id>

<name>John Doe</name>

<email>john@example.com</email>

</user>

#### **Usages of XML:**

* **Data storage** in older applications
* **Web services (SOAP APIs)**
* **Configuration files** (e.g., Android development, MS Office files)

**JSON**: JavaScript Object Notation

JSON is a **lightweight data format** used for **data exchange**.

**Looks Like:** Data is organized in **key-value pairs**.

#### **Example: json**

{

"id": 101,

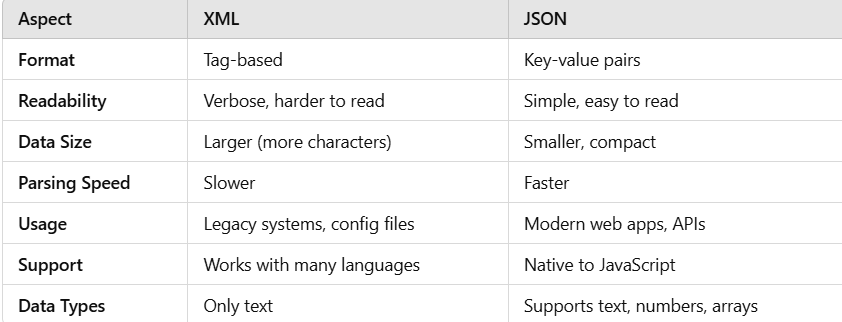
"name": "John Doe",

"email": "john@example.com"

}

#### **Usages of JSON:**

* **APIs (REST APIs)** for fast data transfer
* **Web development** (JavaScript-friendly)
* **Mobile apps** for syncing data



1. **Challenges and limitations**

* **Rate limiting**

Rate limiting controls how many requests a user or system can make to the API in a certain period. It’s like setting a speed limit on a road to avoid traffic jams.

* **Versioning**

Versioning is managing changes to the API without breaking the apps that rely on it. It’s like updating your phone apps—new features without crashing old ones.

* **Error Handling**

Error handling is how APIs respond when something goes wrong, like missing data, wrong inputs, or server issues.

1. REST URL Structure

**REST URL** (Uniform Resource Locator)

# Date: 06/02/2025

**Day - 5) Assigned topics: You will get to know more about following topics**

1. Technical debt management

2. Code optimization, code quality and maintenance

3. CI/CD deployment

4. Data privacy and compliance

5. Methodologies and best practices in Software dev

6. Networking ports and protocols

**1.Technical debt**

Technical debt is like a “**loan**” in coding—you save time now but “**pay**” later by fixing problems. Technical debt is **NOT a tool**. It’s a **concept** in software development. It means the **extra work** needed in the future because developers took **shortcuts** to finish code quickly.

### **Is It Good or Bad?**

* **Can be GOOD** if used wisely (to meet deadlines fast, then fix later).
* **BAD** if ignored for too long (leads to bugs, slow performance, and messy code).

### **Where Do Companies Face Technical Debt?**

* **Startups:** To launch products faster.
* **Big Companies:** During quick feature updates or bug fixes.
* **Legacy Systems:** Old code that hasn’t been updated.

### **Why Is It Used?**

* To **deliver fast** in tight deadlines.
* Focus on **quick wins** and fix technical issues later.

**2. Code optimization, code quality and maintenance**

### **Code Optimization**

Code Optimization = Making code faster, cleaner, and efficient.

### **✅ What to Remove for Better Optimization:**

1. **Unused Code:** Delete functions/variables you never use.
2. **Redundant Code:** Avoid repeating the same code—use functions instead.
3. **Inefficient Loops:** Reduce nested loops; use smart loops or list comprehensions.
4. **Repeated Calculations:** Calculate once, reuse the result.
5. **Memory Leaks:** Free up unused memory/resources.
6. **Excessive Database Calls:** Query once, reuse data.
7. **Hard-Coded Values:** Use variables, not fixed numbers everywhere.
8. **Unnecessary Imports:** Import only what you need.

### **Golden Rule:**

**"Write less, do more, and keep it clean!"**

### **Code Quality**

**Code Quality = Writing clean, understandable, and reliable code.**

### **✅ Key Points for High Code Quality:**

1. **Readable Code:** Use simple, clear names for variables and functions.
2. **No Complexity:** Avoid overcomplicating things—keep it simple.
3. **Error-Free:** Write code that does what it's supposed to and handles edge cases.
4. **Testable:** Make your code easy to test (write unit tests).
5. **Follow Standards:** Stick to industry standards and best practices.
6. **Use Comments:** Explain tricky parts of your code with short comments.

### **💡 Golden Rule:**

**"Clear code is happy code."** 🚀

### **Code Maintenance**

Code Maintenance = Keeping code working, updated, and bug-free over time.

### **✅ Key Practices for Good Code Maintenance:**

1. **Fix Bugs Quickly:** Address issues as soon as they pop up.
2. **Update for New Tech:** Keep the code compatible with new tools or platforms.
3. **Refactor:** Regularly improve code structure without changing its behavior.
4. **Document Well:** Keep documentation updated and helpful.
5. **Version Control:** Use Git to track changes and avoid mistakes.
6. **Regular Backups:** Always back up your code to avoid losing important work.

### **Golden Rule:**

**"Maintain your code like you maintain a car—regular checks keep it running smoothly."**

**3. CI/CD deployment**

**CI (Continuous Integration):**

* **What?:** Automatically integrates code changes into a shared codebase multiple times a day.
* **Why?**: Detects bugs early and ensures new code works with the existing code.
* **How?**: Developer pushes code → CI tool builds & tests it → If successful, it’s merged into the main branch.
* **Tools:** GitHub, Jenkins, Travis CI, GitHub Actions.

**CD (Continuous Delivery):**

* **What?:** Ensures the code is automatically prepared for release after passing tests.
* **Why?:** Speeds up releases, avoiding deployment day headaches.
* **How?:** Code is deployed to staging → Reviewed → Released to production (manual approval).
* **Tools: GitLab CI, Jenkins, CircleCI.**

**CD (Continuous Deployment) [Optional]:**

* **What?:** Automatically deploys code to production after successful testing without human intervention.
* **Why?:** Faster releases.
* **Tools:** Jenkins, GitLab CI.

### **⚡ CI/CD Pipeline Steps:**

1. **Code Commit:** Push code to version control (Git).
2. **Build:** CI tool builds the app.
3. **Test:** Automated tests run.
4. **Deploy:** Code deployed to staging.
5. **Review:** Check in staging.
6. **Production:** Deploy to live app.
7. **Monitor:** Ongoing monitoring.

### **Benefits of CI/CD:**

* Faster development & reliable delivery.
* Early bug detection & automation reduces human errors.
* Improved collaboration & consistent code.

### **4. Data Privacy and Compliance**

### **Data Privacy:**

* **What?**: Protecting personal and sensitive information from unauthorized access.
* **Why?**: To ensure people’s data is kept safe and used responsibly.
* **How?**:
  + **Data Encryption:** Secures data from hackers.
  + **Access Control:** Only authorized people can access data.
  + **Anonymization:** Data is modified so individuals cannot be identified.
* **Key Principles:**
  + **Data Minimization**: Collect only the data you need.
  + **Transparency**: Inform users about how their data is used.
  + **Consent**: Get permission before using or storing personal data.

### **Compliance:**

* **What?**: Following laws and regulations that govern how data should be managed.
* **Why?**: To avoid legal issues and fines.
* **How?**:
  + **GDPR (General Data Protection Regulation)**: Protects EU citizens’ data.
  + **CCPA (California Consumer Privacy Act)**: Protects personal data of California residents.
  + **HIPAA (Health Insurance Portability and Accountability Act)**: Protects health data.
  + **PCI-DSS (Payment Card Industry Data Security Standard)**: Ensures secure credit card transactions.
* **How Companies Ensure Compliance:**
  + Regular audits.
  + Data breach response plans.
  + Privacy policies and training.

**5. Methodologies and best practices in Software dev**

### **1. Software Development Methodologies:**

* **Agile**:
  + Iterative, flexible development in sprints.
  + Frequent meetings and updates, continuous testing.
* **Waterfall**:
  + Sequential, linear process (Fixed phases).
  + Detailed upfront planning for clear requirements.
* **DevOps**:
  + Combines Dev + IT Ops to automate and improve software delivery.
  + Focus on Continuous Integration (CI) and Continuous Delivery (CD).
* **Lean**:
  + Optimizes efficiency, reduces waste.
  + Focus on customer needs and streamlining processes.

### **2. Best Practices in Software Development:**

* **Code Quality**:
  + Write clean, readable, and efficient code.
  + Use meaningful names and follow standards.
* **Version Control (Git)**:
  + Track and manage code changes, use branches, and commit often.
* **Testing**:
  + Write unit tests and automate testing for faster feedback.
* **Documentation**:
  + Keep README files and inline comments updated for clarity.
* **Code Review**:
  + Review code regularly to catch bugs and improve quality.
* **CI/CD**:
  + Automate integration and delivery to speed up development and catch issues early.
* **Security**:
  + Follow secure coding practices, update dependencies, and protect sensitive data.

### **Key Takeaways:**

* **Methodologies** guide development approaches (Agile, Waterfall, etc.).
* **Best Practices** ensure high-quality, maintainable, and secure code.
* Combine both for faster, better, and more reliable software development.

### 

### **6. Network and Protocol**

#### **1. What is a Network?**

* **Definition**: A network connects multiple devices (like computers, phones) to share data and resources (like the internet, printers).
* **Example**: Wi-Fi, LAN (Local Area Network), and the internet.

#### **2. What is a Protocol?**

* **Definition**: A protocol is a set of rules that devices follow to communicate over a network.
* **Example**: HTTP (for web browsing), SMTP (for sending emails).

#### **3. Common Protocols and Their Uses**

* **HTTP (HyperText Transfer Protocol)**:
  + Used for browsing websites.
  + **Port**: 80
* **HTTPS (Secure HTTP)**:
  + Encrypted version of HTTP for secure browsing.
  + **Port**: 443
* **SMTP (Simple Mail Transfer Protocol)**:
  + Used for sending emails.
  + **Port**: 25
* **IMAP (Internet Message Access Protocol)**:
  + Used to view emails on a server without downloading them.
  + **Port**: 143
* **POP3 (Post Office Protocol)**:
  + Used to download emails to your device and remove them from the server.
  + **Port**: 110
* **FTP (File Transfer Protocol)**:
  + Used for transferring files between devices.
  + **Port**: 21
* **DNS (Domain Name System)**:
  + Used to convert domain names (like google.com) into IP addresses.
  + **Port**: 53

#### **4. Ports**

* **Definition**: Ports are like “doors” through which data enters or exits a device.
* **Example**: Port 80 for web browsing (HTTP), Port 25 for email (SMTP).

#### **5. Why Are Networks and Protocols Important?**

* **Networking** connects devices to share resources.
* **Protocols** ensure devices communicate in an organized way to avoid errors.