🚀 TechSaksham Training - Day 6 Notes

🔹 Encapsulation in Python

Encapsulation is one of the core principles of Object-Oriented Programming (OOP) that restricts direct access to certain details of an object and only allows controlled interactions through methods.

Access Specifiers in Python

Python provides three types of access modifiers:

1. **Public (`var`)** – Accessible from anywhere.
2. **Protected (`_var`)** – Can be accessed within the class and its subclasses (convention-based, not enforced).
3. **Private (`__var`)** – Cannot be accessed directly outside the class (uses name-mangling `_ClassName__var`).

Working with Encapsulation

- **Public Variable Example:** Can be accessed anywhere.
- **Protected Variable Example:** Suggests restricted access but still accessible.
- **Private Variable Example:** Meant for internal class use.

```
class Parent:
    def __init__(self):
        self.__private_var = 100  # Private variable

    def get_private_var(self):
        return self.__private_var  # Public method to access private variable

p = Parent()
print(p.get_private_var())  # ✅ Best practice! Uses encapsulation principles.
```

🔹 Polymorphism in Python

Polymorphism allows methods to have **different behaviors** based on the object calling them.

Types of Polymorphism:

1. **Method Overriding (Runtime Polymorphism)**
2. **Method Overloading (Achieved using default arguments in Python)**
3. **Operator Overloading (Custom behavior for operators like +, -, * etc.)**

Example: Method Overriding

```
class Parent:
    def show(self):
        print("This is the Parent class")

class Child(Parent):
    def show(self):
        print("This is the Child class")

c = Child()
c.show()  # Output: This is the Child class
```

Example: Operator Overloading

```
class Book:
    def __init__(self, pages):
        self.pages = pages

    def __add__(self, other):  # Overloading + operator
        return Book(self.pages + other.pages)

b1 = Book(100)
b2 = Book(150)
b3 = b1 + b2  # Calls __add__()
print(b3.pages)  # Output: 250
```

🔹 Abstraction in Python

Abstraction hides complex details and only exposes essential features.

Using Abstract Classes

- Abstract classes **cannot be instantiated**.
- They contain **abstract methods** that must be implemented by subclasses.

```
from abc import ABC, abstractmethod
```

```
class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Car engine started!")

c = Car()
c.start()  # Output: Car engine started!
```

🔹 Exception Handling in Python

Exception handling ensures that programs continue running **even if an error occurs**.

Basic Exception Handling Syntax

```
try:
    x = 10 / 0  # ZeroDivisionError
except ZeroDivisionError as e:
    print("Error:", e)
finally:
    print("Execution Completed!")
```

Common Python Exceptions:

- `ZeroDivisionError`: Division by zero.
- `ValueError`: Invalid type of value.
- `TypeError`: Incompatible data types.
- `IndexError`: Accessing invalid index.
- `FileNotFoundError`: File does not exist.

🔹 File Handling in Python

File handling enables reading, writing, and manipulating files.

File Modes

1. `'r'` – Read mode (file must exist).
2. `'w'` – Write mode (overwrites if file exists).
3. `'a'` – Append mode (adds data at the end).
4. `'x'` – Create mode (fails if file exists).

Example: Writing to a File

```
with open("example.txt", "w") as file:
    file.write("Hello, World!")
```

Example: Reading from a File

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

🔹 Classroom Activity: Code Presentations

📌 **I presented:** Finding unique characters in a string.

```
def unique_character(s):
    u_char = [char for char in s if s.count(char) == 1]
    return ",".join(u_char) if u_char else "No unique character"

word = "swiss"
print(unique_character(word))  # Output: "w"
```

📌 **Other Presentations:**

- **Password Generator** – Randomly generates secure passwords.
- **Password Strength Checker** – Validates password complexity.
- **Simple Calculator** – Performs arithmetic operations.
- **More exciting mini-projects!**

💡 Summary & Key Takeaways

✅ **Encapsulation** ensures data protection with public, protected, and private members.
✅ **Polymorphism** allows methods and operators to work differently based on context.
✅ **Abstraction** simplifies implementation by hiding unnecessary details.

✅ **Exception Handling** prevents program crashes by managing errors.
✅ **File Handling** enables efficient data storage and retrieval.
✅ **Classroom Activity** helped reinforce key concepts through hands-on presentations.

📜 **Looking forward to more learning in the upcoming sessions!** 🚀