Class #09: Object-Oriented Programming (OOP) – Classes, Objects & Inheritance

1. Understanding classes and objects.
2. Methods vs functions, and magic (dunder) methods.
3. Exploring inheritance in Python.
4. Understanding polymorphism in Python.
5. Working with constructors and destructors in Python. Assignment #9:

3. Exploring inheritance in Python.

Inheritance (উত্তরাধিকার) প্রোগ্রামিং ভাষায় একটি গুরুত্বপূর্ণ বৈশিষ্ট্য, বিশেষত অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং (OOP) তে। এটি এক ক্লাসের বৈশিষ্ট্য এবং আচরণ (properties and methods) অন্য একটি ক্লাসে হস্তান্তর করার প্রক্রিয়া। এই বৈশিষ্ট্যটির মাধ্যমে আমরা কোড পুনরায় ব্যবহার (code reuse) করতে পারি এবং নতুন ক্লাস তৈরি করতে পারি যা পূর্ববর্তী ক্লাসের বৈশিষ্ট্যগুলো উত্তরাধিকার হিসেবে পায়।

প্রোগ্রামিংয়ে কিভাবে কাজ করে:

Inheritance এর বাংলা অর্থ হচ্ছে উত্তরাধিকার। ধরুন, একটি ক্লাসের নাম SheikhMujib, যা বেস ক্লাস (superclass)। আরেকটি ক্লাসের নাম SheikhHasina, যা সাবক্লাস (subclass)। একটি বেস ক্লাস (superclass) থেকে সাবক্লাস (subclass)-এ বৈশিষ্ট্য এবং আচরণ (methods) উত্তরাধিকার (inherit) করে।

এখন, আপনি চাইলে SheikhHasina ক্লাসে SheikhMujib ক্লাসের সব বৈশিষ্ট্য এবং আচরণ উত্তরাধিকার হিসেবে পেতে পারেন, এবং পাশাপাশি নতুন কিছু বৈশিষ্ট্য যোগ করতে পারেন। এর মাধ্যমে আপনি কোডের পুনঃব্যবহারযোগ্যতা নিশ্চিত করতে পারবেন, এবং পুনরায় কোড লেখার প্রয়োজন পড়বে না। এতে আপনার কোড আরও সুশৃঙ্খল এবং আরও কার্যকরী হবে।

উদাহরণস্বরূপ, SheikhMujib ক্লাসের কিছু বৈশিষ্ট্য ও আচরণ SheikhHasina ক্লাসে সহজেই উত্তরাধিকার হিসেবে পাওয়া যায় এবং আপনার প্রয়োজনীয় নতুন বৈশিষ্ট্য যোগ করতে পারেন, যেমন SheikhHasina সম্পর্কিত কিছু বিশেষ আচরণ বা তথ্য।

এইভাবে, inheritance আমাদের কোড লেখার সময় কাজকে সহজ ও কার্যকরী করে তোলে, এবং কোডের পুনঃব্যবহারযোগ্যতা বাড়ায়।

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

```
    def surname(self):
        print(f"{self.name} is a member of the Sheikh family.")

    def politician(self):
        print(f"{self.name} is an Awami leader.")

class SheikhHasina(SheikhMujib):
    def pain(self):
        print(f"আপনজন হারানোর বেদনা যে কত কষ্ট, {self.name} চেয়ে আর কেউ বেশি জানে
না’")

# Creating an instance of SheikhHasina class
n_hasina = SheikhHasina("Sheikh Hasina Wazed", 78)

# Calling methods
n_hasina.surname()   # Method from SheikhMujib class
n_hasina.politician()   # Method from SheikhMujib class
n_hasina.pain()   # Method from SheikhHasina class
```

```
Sheikh Hasina Wazed is a member of the Sheikh family.
Sheikh Hasina Wazed is an Awami leader.
আপনজন হারানোর বেদনা যে কত কষ্ট, Sheikh Hasina Wazed চেয়ে আর কেউ বেশি জানে না’
```

এখানে SheikhHasina ক্লাসের একটি ইনস্ট্যান্স `n_hasina` তৈরি করা হচ্ছে, যেখানে name হচ্ছে "Sheikh Hasina Wazed" এবং age হচ্ছে 78। এটি SheikhMujib ক্লাসের কনস্ট্রাক্টরকেও কল করে, তাই নাম এবং বয়স সেট হবে।

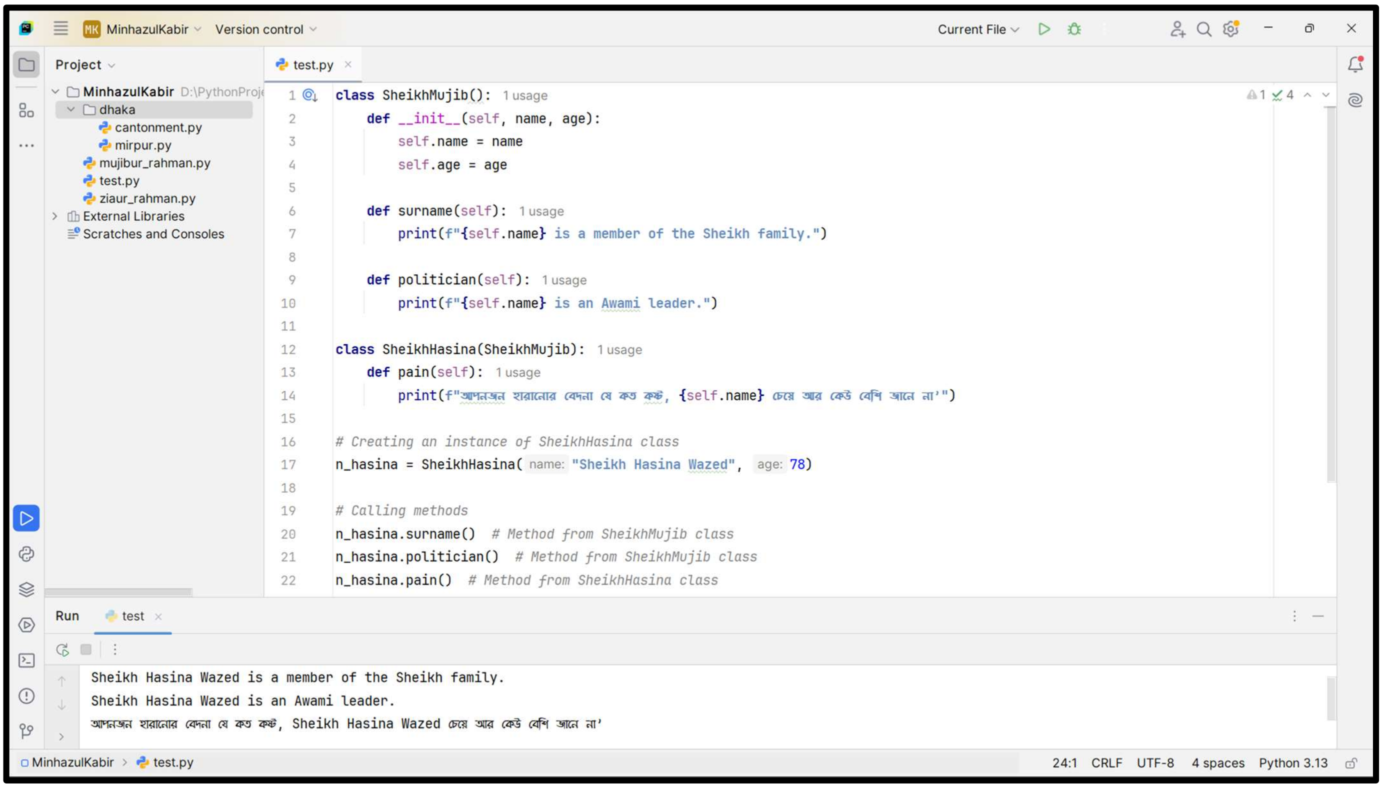
ব্যাখ্যা:

1. প্রথমে n_hasina.surname() কল করলে "Sheikh Hasina Wazed is a member of the Sheikh family." আউটপুট আসে, কারণ surname() মেথডটি SheikhMujib ক্লাস থেকে এসেছে।
2. তারপর n_hasina.politician() কল করলে "Sheikh Hasina Wazed is an Awami leader." আউটপুট আসে, এটি আবার SheikhMujib ক্লাস থেকে।
3. শেষের n_hasina.pain() কল করার ফলে "**আপনজন হারানোর বেদনা যে কত কষ্ট,** Sheikh Hasina Wazed **চেয়ে আর কেউ বেশি জানে না**" আউটপুট আসে, যা SheikhHasina ক্লাসের নিজস্ব মেথড।

সারাংশ:

এই কোডটি Inheritance এর মাধ্যমে SheikhHasina ক্লাসকে SheikhMujib ক্লাসের বৈশিষ্ট্য এবং আচরণ উত্তরাধিকার দিতে সাহায্য করেছে, যার ফলে কোড আরও সহজ, পুনঃব্যবহারযোগ্য এবং সুশৃঙ্খল হয়েছে।

এখানে, n_hasina.surname() এবং n_hasina.politician() মেথডগুলি SheikhMujib ক্লাস থেকে আসছে, কারণ SheikhHasina ক্লাসটি SheikhMujib ক্লাস থেকে উত্তরাধিকার পায়। n_hasina.pain() মেথডটি SheikhHasina ক্লাসের নিজস্ব মেথড, যা শেখ হাসিনার বেদনা বর্ণনা করে।



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

    def surname(self):
        print(f"{self.name} is a member of the Sheikh family.")

    def politician(self):
        print(f"{self.name} is an Awami leader.")

class SheikhHasina(SheikhMujib):
    def pain(self):
        print(f"আপনজন হারানোর বেদনা যে কত কষ্ট, {self.name} চেয়ে আর কেউ বেশি জানে না'")

# Creating an instance of SheikhHasina class
n_hasina = SheikhHasina( name: "Sheikh Hasina Wazed",  age: 78)

# Calling methods
n_hasina.surname()  # Method from SheikhMujib class
n_hasina.politician()  # Method from SheikhMujib class
n_hasina.pain()  # Method from SheikhHasina class
```

```
Sheikh Hasina Wazed is a member of the Sheikh family.
Sheikh Hasina Wazed is an Awami leader.
আপনজন হারানোর বেদনা যে কত কষ্ট, Sheikh Hasina Wazed চেয়ে আর কেউ বেশি জানে না'
```

Multilevel Inheritance

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

    def accommodation(self):
        print(f"{self.name} lived in the cantonment.")

    def politician(self):
        print(f"{self.name} is a Bangladesh Nationalist Party
leader.")

class KhaledaZia(ZiaurRahman):
    def money(self):
        print(f"{self.name} achieved a lot of money.")

class TariqueZia(KhaledaZia):
    def friend(self):
        print(f"{self.name} is the best friend of Sajeeb Wazed
Joy.")

# Creating an instance of TariqueZia class
p_tarique = TariqueZia("Tarique Rahman Zia", 57)

# Calling methods
```

```
p_tarique.accommodation()  # Method from ZiaurRahman class
p_tarique.politician()  # Method from ZiaurRahman class
p_tarique.money()  # Method from KhaledaZia class
p_tarique.friend()  # Method from TariqueZia class
```

```
Tarique Rahman Zia lived in the cantonment.
Tarique Rahman Zia is a Bangladesh Nationalist Party leader.
Tarique Rahman Zia achieved a lot of money.
Tarique Rahman Zia is the best friend of Sajeeb Wazed Joy.
```

কোডের বিশ্লেষণ:

1. **ZiaurRahman** ক্লাসটি বেস ক্লাস হিসেবে কাজ করছে, যেটি দুটি মেথড (accommodation এবং politician) ডিফাইন করেছে।
2. **KhaledaZia** ক্লাসটি ZiaurRahman ক্লাস থেকে উত্তরাধিকারিত (inherit) হয়েছে এবং একটি অতিরিক্ত মেথড (money) যোগ করেছে।
3. **TariqueZia** ক্লাসটি KhaledaZia ক্লাস থেকে উত্তরাধিকারিত (inherit) হয়েছে এবং একটি অতিরিক্ত মেথড (friend) যোগ করেছে।

এই তিনটি ক্লাসের মাধ্যমে, **Multilevel Inheritance** তৈরি হয়েছে, যেখানে:

TariqueZia → KhaledaZia → ZiaurRahman (এটি একটি তিন স্তরের উত্তরাধিকার)

এখানে TariqueZia ক্লাসটি **KhaledaZia** থেকে ইনহেরিট করেছে, এবং **KhaledaZia** ক্লাসটি আবার **ZiaurRahman** ক্লাস থেকে ইনহেরিট করেছে। সুতরাং, TariqueZia ক্লাসটি তিনটি স্তরের ইনহেরিটেন্সের মাধ্যমে সকল মেথড এবং বৈশিষ্ট্য পেয়ে থাকে।

Multiple Inheritance এবং Multilevel Inheritance এর মধ্যে পার্থক্য একটি টেবিলের মাধ্যমে তুলে ধরা হলো:

বৈশিষ্ট্য	Multiple Inheritance	Multilevel Inheritance
সংজ্ঞা	একাধিক সুপারক্লাস থেকে একটি সাবক্লাস বৈশিষ্ট্য এবং আচরণ উত্তরাধিকার নেয়।	একটি ক্লাস থেকে আরেকটি ক্লাস উত্তরাধিকার নেয়ার পর, সেই ক্লাস আবার অন্য একটি ক্লাস থেকে উত্তরাধিকার নেয়।
উদাহরণ	যদি একটি `Chicken` ক্লাস Bird এবং Animal ক্লাস থেকে উত্তরাধিকার নেয়।	Class A -> Class B -> Class C (এখানে Class C হল Class A এর উত্তরাধিকারী)
সংযুক্ত ক্লাসের সংখ্যা	একাধিক সুপারক্লাসের সাথে যুক্ত থাকে।	একটাই সুপারক্লাস থেকে একাধিক স্তরের উত্তরাধিকার সৃষ্টি হয়।
অবস্থান	একাধিক সুপারক্লাস একই স্তরে থাকতে পারে।	একাধিক স্তর একে অপরের উপর ভিত্তি করে থাকে।
এটি কীভাবে কাজ করে?	একটি সাবক্লাস একাধিক সুপারক্লাস থেকে বৈশিষ্ট্য এবং আচরণ গ্রহণ করে।	একটি ক্লাস (সুপারক্লাস) থেকে উত্তরাধিকার নিয়ে, পরবর্তী ক্লাসে সেই উত্তরাধিকার পাওয়া যায়।

কোডের জটিলতা	কোড জটিল হতে পারে, বিশেষত Diamond Problem সৃষ্টি হতে পারে।	কোড সাধারণত সহজ এবং ক্লাসের মধ্যে সম্পর্ক স্পষ্ট থাকে।
উদাহরণ (Python)	class Chicken(Bird, Animal):	class C(B): (যেখানে B -> A)

Multiple Inheritance

```
class Bird():
    def fly(self):
        print("Flying")

class Animal()s:
    def eat(self):
        print("Eating")

class Chicken(Bird, Animal):
    pass

# Creating an instance of Chicken class
obj_fly = Chicken()

# Calling methods
obj_fly.fly()    # Method from Bird class
obj_fly.eat()    # Method from Animal class
```

```
Flying
Eating
```

আমাকে output লিখে দাও ।

```
# Vehicle class - Base class
class Vehicle():
    def __init__(self, brand, model):
        self.brand = brand   # Brand of the vehicle
        self.model = model   # Model of the vehicle

class Car(Vehicle): # Car class - Inherits from Vehicle
    pass

# Creating an instance of the Car class
p1 = Car("Hi", 9)
# Printing the brand and model of the car object
print(p1.brand, p1.model)
```

4. Understanding polymorphism in Python.

Method Overriding (মেথড ওভাররাইডিং): এই ক্ষেত্রে, একটি সাবক্লাস একটি মেথডকে override (অথবা পুনঃসংজ্ঞায়িত) করে, যার ফলে সেই মেথডটি সুপারক্লাসের মেথডের চেয়ে ভিন্নভাবে কাজ করে।

কেন Polymorphism ব্যবহার করা হয়?

1. Code Reusability (কোড পুনঃব্যবহারযোগ্যতা): একই মেথড বা ফাংশনকে ভিন্ন ধরনের অবজেক্টে ব্যবহার করা যায়, যা কোডের পুনঃব্যবহার বৃদ্ধি করে।
2. Flexibility (নমনীয়তা): একাধিক ক্লাসের মধ্যে একই নামের মেথড ব্যবহার করা সম্ভব হওয়ায়, প্রোগ্রামটি আরও নমনীয় এবং পরিবর্তনশীল হয়ে ওঠে।
3. Readability (পাঠযোগ্যতা): একই মেথড নাম দিয়ে বিভিন্ন ধরনের আচরণ সংজ্ঞায়িত করা সহজে বুঝতে সহায়তা করে।

```python
# Base class
class ZiaurRahman():
    def designation(self):
        print("Ziaur Rahman was a Bangladeshi military officer and politician.")

# Subclass KhaledaZia inheriting from ZiaurRahman
class KhaledaZia(ZiaurRahman):
    def designation(self):
        print("Khaleda Zia served as the Prime Minister of Bangladesh.")

# Subclass TariqueZia inheriting from ZiaurRahman
class TariqueZia(ZiaurRahman):
    def designation(self):
        print("Tarique Zia is the acting chairman of Bangladesh Nationalist Party.")

# Creating objects of the subclasses
khaleda = KhaledaZia()
tarique = TariqueZia()

# Calling the same method, but different behaviors based on the object
khaleda.designation()  # Outputs: Khaleda Zia served as the Prime Minister of Bangladesh.
tarique.designation()  # Outputs: Tarique Zia is the acting chairman of Bangladesh Nationalist Party.
```

ব্যাখ্যা:

1. **ক্লাস** ZiaurRahman:

   এটি একটি **বেস ক্লাস**। এখানে একটি designation() মেথড রয়েছে যা Ziaur Rahman সম্পর্কে একটি সাধারণ বিবরণ প্রিন্ট করে।

2. **ক্লাস** KhaledaZia:

   এটি ZiaurRahman ক্লাসের একটি **সাবক্লাস**। এই ক্লাসে designation() মেথডটি **অভিযোজন** (override) করা হয়েছে, যাতে এটি Khaleda Zia সম্পর্কে নির্দিষ্ট তথ্য প্রদান করে।

3. **ক্লাস** TariqueZia:

   এটি ZiaurRahman ক্লাসের আরেকটি **সাবক্লাস**। এখানে designation() মেথডটি আবার **অভিযোজন** (override) করা হয়েছে, যাতে এটি Tarique Zia সম্পর্কে নির্দিষ্ট তথ্য প্রদর্শন করে।

4. **অবজেক্ট তৈরি:**

   khaleda হল KhaledaZia ক্লাসের একটি অবজেক্ট, এবং tarique হল TariqueZia ক্লাসের একটি অবজেক্ট।

5. designation() **মেথড কল করা:**

   - যখন khaleda.designation() কল করা হয়, তখন এটি KhaledaZia ক্লাসের designation() মেথডটি কল করে এবং Khaleda Zia সম্পর্কে তথ্য প্রদর্শন করে।
   - যখন tarique.designation() কল করা হয়, তখন এটি TariqueZia ক্লাসের designation() মেথডটি কল করে এবং Tarique Zia সম্পর্কে তথ্য প্রদর্শন করে।

99. Abstraction এবং Encapsulation:

প্রোগ্রামিং এর দুটি গুরুত্বপূর্ণ ধারণা, বিশেষ করে Object-Oriented Programming (OOP) এর অংশ। এগুলো আপনার কোডের স্বচ্ছতা, কোড পুনরাবৃত্তি, এবং ডাটা সুরক্ষা নিশ্চিত করতে সাহায্য করে। চলুন, পাইথনে এগুলো কীভাবে কাজ করে তা ব্যাখ্যা করি।

Abstract Class এবং Abstract Method এর বাস্তব উদাহরণ:

ধরা যাক, একটি ক্লাসে **স্যার** সবাইকে একটি **এসাইনমেন্ট** জমা দিতে বলেছেন। এসাইনমেন্টের **টপিক** হলো "Python Programming" এবং **জমা দেওয়ার সর্বশেষ তারিখ** হলো ৫ আগস্ট। স্যার কেবল **নির্দেশনা** দিলেন যে, "এই টপিকের উপর এসাইনমেন্ট জমা করতে হবে"। কিন্তু, এসাইনমেন্টের **প্রধান কাজ** এবং **বাস্তবায়ন** কিন্তু **শিক্ষার্থীদেরই** করতে হবে।

এটি একটি বাস্তব উদাহরণ হিসেবে Abstract Class এবং Abstract Method-এর ধারণা ব্যাখ্যা করতে সাহায্য করে। এখানে, স্যারের দেওয়া **নির্দেশনা** হলো Abstract Method, যা **শিক্ষার্থীদের** জন্য একটি **ব্লুপ্রিন্ট** হিসেবে কাজ করে। তবে, নির্দেশনা অনুযায়ী কাজ করার বাস্তবায়ন বা বিস্তারিত **শিক্ষার্থীদেরই** করতে হবে, যা Concrete Class দ্বারা সম্পন্ন হয়।

উদাহরণ ব্যাখ্যা:

1. **স্যার** (Abstract Class): স্যার নিজে এসাইনমেন্টটি করবেন না, তবে তিনি **নির্দেশনা** দিয়েছেন। স্যার জানিয়ে দিয়েছেন যে এসাইনমেন্টটি "Python Programming"-এ হতে হবে এবং তার **জমা দেওয়ার তারিখ** ৫ আগস্ট। তবে, স্যার এই কাজের **বিস্তারিত বাস্তবায়ন** করবেন না, তিনি শুধু একটি **ব্লুপ্রিন্ট** তৈরি করেছেন যা শিক্ষার্থীদের অনুসরণ করতে হবে। এটি Abstract Class এর মতো, যেখানে **এমন কিছু নির্দেশনা** দেওয়া হয় যেগুলোর বাস্তবায়ন **সাবক্লাস** (শিক্ষার্থীদের) করতে হয়।
2. **এসাইনমেন্টের নির্দেশনা** (Abstract Method): স্যারের দেওয়া নির্দেশনা, যেমন "Python Programming"-এ এসাইনমেন্ট লিখতে হবে, তা একটি Abstract Method এর মতো কাজ করে। এটি কেবল জানিয়ে দেয় **কি** করতে হবে, কিন্তু এর বিস্তারিত বাস্তবায়ন কখনোই দেওয়া হয়নি। এটি শুধুমাত্র একটি **সিগনেচার** (signature) বা রূপরেখা দেয়, যা শিক্ষার্থীদের নিজে করে বাস্তবায়ন করতে হবে।
3. **শিক্ষার্থীরা** (Concrete Class): শিক্ষার্থীরা তাদের নিজ নিজ পদ্ধতিতে এসাইনমেন্টটি সম্পন্ন করবে। যেমন, একজন শিক্ষার্থী Python Programming এর উপর একটি প্রোগ্রাম লিখতে পারে, অন্যজন হয়তো অন্য কোনো বিষয়ে মনোযোগী হবে। এটি Concrete Class এর মতো, যেখানে Abstract Method কে বাস্তবায়ন করে **প্রোগ্রামাররা** (শিক্ষার্থীরা) কার্যকরভাবে কাজটি সম্পন্ন করে।

উপসংহার:

এভাবে, Abstract Class এবং Abstract Method যেমন নির্দেশনা দেয়, সেগুলোর **বাস্তবায়ন** কিন্তু Concrete Class (শিক্ষার্থী) করে। এটি কোডের স্ট্রাকচারকে পরিষ্কারভাবে নির্ধারণ করে, যেখানে কোন কাজ কীভাবে করতে হবে তা সাবক্লাস বা শিক্ষার্থীরা সিদ্ধান্ত নেয় এবং বাস্তবায়ন করে।

Abstraction (অ্যাবস্ট্র্যাকশন):

Abstraction হলো **অপ্রয়োজনীয় তথ্য গোপন** করে **গুরুত্বপূর্ণ বৈশিষ্ট্য বা আচরণ** প্রকাশ করার প্রক্রিয়া। এর মাধ্যমে, আপনি ব্যবহারকারীদের শুধুমাত্র সেই তথ্য বা ফিচার সরবরাহ করেন যা তাদের প্রয়োজন এবং বাকি কমপ্লেক্সিটি লুকিয়ে রাখেন।

পাইথনে অ্যাবস্ট্র্যাকশন সাধিত হতে পারে:

1. Abstract Classes এবং Abstract Methods এর মাধ্যমে।
2. Method Overriding ব্যবহার করে।

Python-এ **Abstract Classes** এবং **Abstract Methods** হল Object-Oriented Programming (OOP) এর একটি গুরুত্বপূর্ণ ধারণা, যা ক্লাস এবং মেথডের মধ্যে একটি বেসিক কাঠামো তৈরি করতে সাহায্য করে। এগুলির উদ্দেশ্য হল, সাবক্লাস (derived class) গুলিকে নির্দিষ্ট নিয়ম বা ইন্টারফেস অনুসরণ করতে বাধ্য করা, যাতে কোড আরও পরিস্কার, সুসংগত এবং পুনঃব্যবহারযোগ্য হয়।

Abstract Class:

একটি **Abstract Class** হল এমন একটি ক্লাস, যার মধ্যে কমপক্ষে একটি **abstract method** থাকে এবং সেটি নিজে থেকে ইনস্ট্যানশিয়েট করা যায় না। অর্থাৎ, Abstract Class এর সরাসরি অবজেক্ট তৈরি করা সম্ভব নয়। এটি সাধারণত বেস ক্লাস (base class) হিসেবে ব্যবহার করা হয়, যাতে তা অন্য ক্লাসের জন্য একটি ব্লুপ্রিন্ট হিসেবে কাজ করে।

Abstract Method:

একটি **Abstract Method** হল এমন একটি মেথড যা Abstract Class-এ ডিফাইন করা থাকে, কিন্তু এটি ক্লাসের মধ্যে কোন বাস্তবায়ন (implementation) থাকে না। এটি শুধু একটি সিগনেচার (signature) প্রদান করে এবং এর বাস্তবায়ন সাবক্লাসে করতে হয়।

Python-এ Abstract Class এবং Abstract Method কীভাবে তৈরি করা হয়?

Python-এ abc (Abstract Base Class) মডিউল ব্যবহার করে Abstract Class এবং Abstract Methods তৈরি করা হয়। এখানে ABC এবং abstractmethod ডেকোরেটর ব্যবহার করা হয়।

উদাহরণ:

```
from abc import ABC, abstractmethod

# Abstract class
class Animal(ABC):
    # Abstract method
    @abstractmethod
    def sound(self):
        pass

# Concrete class that inherits from Animal class
class Dog(Animal):
    # Implementation of the abstract method
    def sound(self):
        return "Bark"

class Cat(Animal):
    # Implementation of the abstract method
```

```
    def sound(self):
        return "Meow"


# Now, we cannot create an object of the Animal class
# animal = Animal()   # This will give an error

# We can create objects of the Dog and Cat classes
dog = Dog()
print(dog.sound())  # Output: Bark

cat = Cat()
print(cat.sound())  # Output: Meow
```

ব্যাখ্যা:

1. **Animal ক্লাস** একটি abstract class, কারণ এটি ABC থেকে ইনহেরিট করা হয়েছে এবং এর মধ্যে sound মেথডটি abstract method হিসাবে ডিফাইন করা আছে।
2. Dog এবং Cat ক্লাসগুলি **Concrete Classes**, কারণ সেগুলো Animal ক্লাস থেকে ইনহেরিট করেছে এবং sound মেথডের বাস্তবায়ন (implementation) দিয়েছে।
3. **sound মেথড** প্রত্যেক concrete class-এ বাস্তবায়ন করতে হবে, অন্যথায় Python ত্রুটি দেখাবে।

উপকারিতা:

- **Code Reusability:** Abstract Classes এবং Abstract Methods দিয়ে পুনঃব্যবহারযোগ্য কোড তৈরি করা সম্ভব।
- **Structure and Design:** এটি কোডের স্ট্রাকচার নির্ধারণে সাহায্য করে, এবং সাবক্লাসগুলি নির্দিষ্ট নিয়ম মেনে কাজ করে।
- **Enforcing Implementation:** একটি abstract method ঘোষণা করে, আমরা নিশ্চিত হতে পারি যে, তার বাস্তবায়ন সাবক্লাসে করা হবে।

উপসংহার:

Abstract Classes এবং Abstract Methods OOP ডিজাইনে একটি গুরুত্বপূর্ণ ভূমিকা পালন করে, যা ডেভেলপারদের নির্দিষ্ট কাঠামো অনুসরণ করতে সাহায্য করে এবং কোডের গুণগত মান উন্নত করতে সহায়ক।

Encapsulation (এনক্যাপসুলেশন):

Encapsulation হলো একটি কৌশল যার মাধ্যমে ডেটা এবং তার আচরণকে (ফাংশন) একত্রে একটি ইউনিটে বেঁধে রাখা হয়, এবং ডেটার সরাসরি অ্যাক্সেস সীমিত করা হয়। এটি সাধারণত **private** বা **protected** ভ্যারিয়েবল এবং মেথড ব্যবহার করে ডেটাকে সুরক্ষিত করার জন্য প্রয়োগ করা হয়।

- Private (__) - ডাটা সম্পূর্ণভাবে বাইরের কোড থেকে হিডেন এবং শুধুমাত্র ক্লাসের ভিতরে অ্যাক্সেসযোগ্য।
- Protected (_) - বাইরের কোড থেকে অ্যাক্সেস সীমাবদ্ধ, তবে ইনহেরিটেড ক্লাসে অ্যাক্সেসযোগ্য।

এনক্যাপসুলেশন এর মূল উদ্দেশ্য হল:

1. ডেটার **অবাঞ্ছিত অ্যাক্সেস** বন্ধ করা।
2. **ডেটার সুরক্ষা** নিশ্চিত করা।

পাইথনে এনক্যাপসুলেশন বাস্তবায়ন করা হয় private বা _protected ভ্যারিয়েবল ব্যবহার করে ।

এই প্রোগ্রামটি একটি ক্লাসের ধারণা, ইনহেরিটেন্স এবং প্রাইভেট, প্রটেক্টেড এবং পাবলিক অ্যাট্রিবিউটগুলো কিভাবে কাজ করে তা প্রদর্শন করে।

```
class ZiaurRahman:
    def __init__(self, nationality):
        self.nationality = nationality # Public attribute
        self._surname = "Zia" # Protected attribute (by convention)
        self.__life_style = "poor" # Private attribute
    def show_info(self):
        # This method prints out the information
        print(f"Name: {self.nationality}")
        print(f"Family of: {self._surname}")
        print(f"Age: {self.__life_style}")

class TariqueZia(ZiaurRahman):
    def access_private_protected(self):
        print(f"Sub Class Name: {self.nationality}")  # Accessing the public
attributes
        print(f"Sub Class Family: {self._surname}") # Accessing the protected
attributes
        # Note: Cannot access private attribute directly from the subclass
        # print(f"Sub Class Age: {self.__life_style}")  # This will cause an error

# Creating an object of TariqueZia
n = TariqueZia("Tarique Rahman Zia")
# Calling method to show info
n.show_info()
# Accessing protected attribute (allowed by convention) in the subclass
n.access_private_protected()
```

```
Name: Tarique Rahman Zia
Family of: Zia
Age: poor
Sub Class Name: Tarique Rahman Zia
Sub Class Family: Zia
```

১. ZiaurRahman ক্লাস

এটি একটি সাধারণ ক্লাস যা তিনটি অ্যাট্রিবিউট (বৈশিষ্ট্য) ধারণ করে:

- nationality: এটি একটি public অ্যাট্রিবিউট, যেটি ক্লাসের বাইরে থেকেও সরাসরি অ্যাক্সেস করা যায়।
- _surname: এটি একটি protected অ্যাট্রিবিউট, যেটি সাধারনত ক্লাসের বাইরের কোডে সরাসরি অ্যাক্সেস করা উচিত নয়, তবে ইনহেরিট করা সাব-ক্লাসে অ্যাক্সেস করা যেতে পারে।
- __life_style: এটি একটি private অ্যাট্রিবিউট, যেটি শুধুমাত্র ওই ক্লাসের ভিতরে ব্যবহৃত হতে পারে এবং বাইরের কোড বা সাব-ক্লাস থেকে সরাসরি অ্যাক্সেস করা যাবে না।

২. show_info মেথড

এই মেথডটি ZiaurRahman ক্লাসের সমস্ত বৈশিষ্ট্য (অ্যাট্রিবিউট) প্রদর্শন করে। এতে পাবলিক, প্রটেক্টেড এবং প্রাইভেট অ্যাট্রিবিউটের মান প্রদর্শিত হবে। তবে, প্রাইভেট অ্যাট্রিবিউট শুধুমাত্র সেই ক্লাসের ভিতরে অ্যাক্সেসযোগ্য, তাই আপনি এটি বাইরে থেকে দেখতে পাবেন না।

৩. TariqueZia সাব-ক্লাস

TariqueZia হল ZiaurRahman ক্লাসের একটি সাব-ক্লাস, অর্থাৎ এটি ZiaurRahman ক্লাস থেকে কিছু বৈশিষ্ট্য (অ্যাট্রিবিউট) এবং মেথড উত্তরাধিকারসূত্রে পেয়েছে।

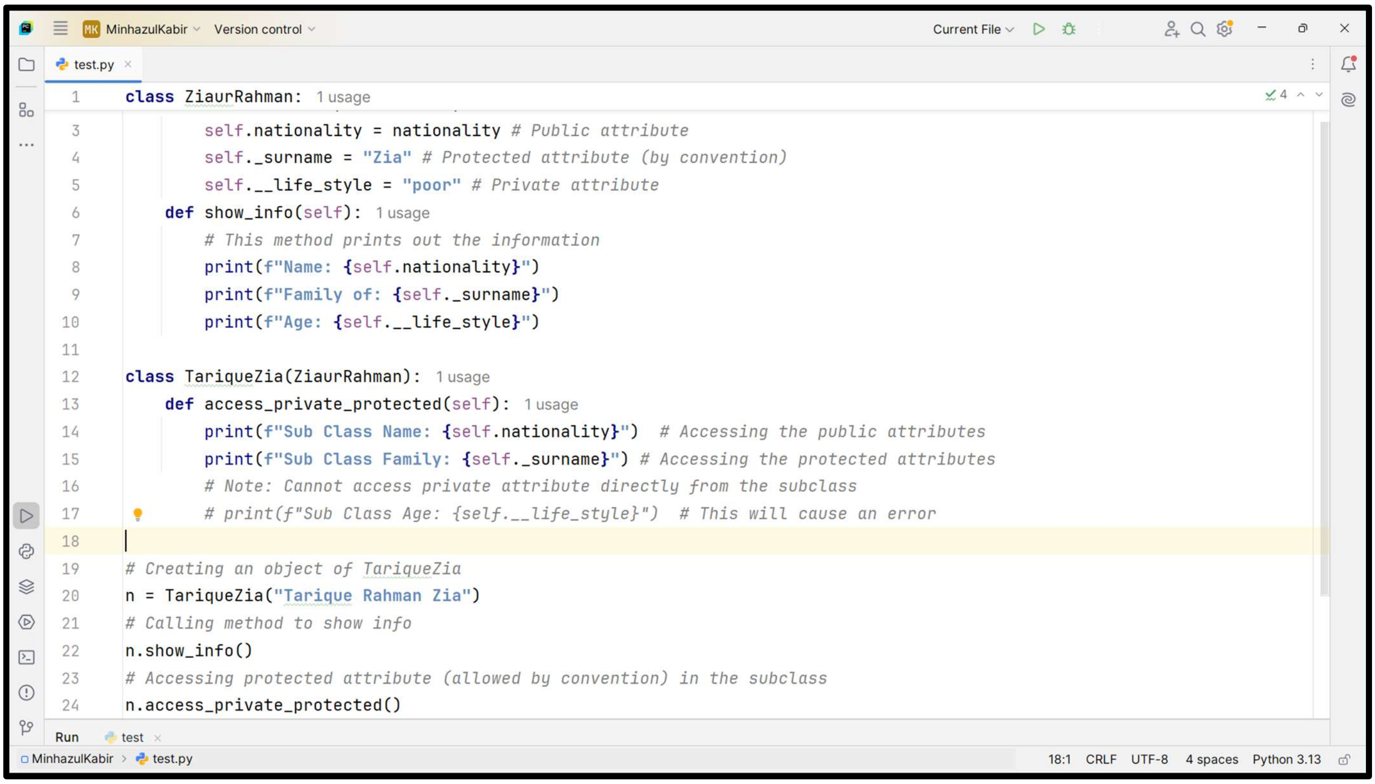
- access_private_protected মেথডে, এটি পাবলিক অ্যাট্রিবিউট nationality এবং প্রটেক্টেড অ্যাট্রিবিউট _surname অ্যাক্সেস করছে। তবে, প্রাইভেট অ্যাট্রিবিউট __life_style সরাসরি অ্যাক্সেস করা সম্ভব নয়, তাই কোডে সেটি মন্তব্য আকারে রাখা হয়েছে।

৪. অবজেক্ট তৈরি

- TariqueZia ক্লাসের একটি অবজেক্ট n তৈরি করা হচ্ছে, যার nationality "Tarique Rahman Zia" দেওয়া হচ্ছে।
- এরপর, n.show_info() মেথড কল করলে এটি ZiaurRahman ক্লাসের তথ্য প্রিন্ট করবে।
- পরে n.access_private_protected() মেথড কল করলে এটি পাবলিক এবং প্রটেক্টেড অ্যাট্রিবিউট প্রদর্শন করবে, তবে প্রাইভেট অ্যাট্রিবিউট প্রদর্শন করা যাবে না।

সারাংশ:

- **পাবলিক অ্যাট্রিবিউট:** ক্লাসের বাহিরে এবং সাব-ক্লাসে সরাসরি অ্যাক্সেস করা যায়।
- **প্রটেক্টেড অ্যাট্রিবিউট:** ক্লাসের বাইরে সরাসরি অ্যাক্সেস করা উচিত নয়, তবে সাব-ক্লাসে অ্যাক্সেস করা যায়।
- **প্রাইভেট অ্যাট্রিবিউট:** শুধুমাত্র ওই ক্লাসের ভিতরে অ্যাক্সেসযোগ্য, বাইরের কোড বা সাব-ক্লাসে সরাসরি অ্যাক্সেস করা যায় না।



```python
class ZiaurRahman:
        self.nationality = nationality # Public attribute
        self._surname = "Zia" # Protected attribute (by convention)
        self.__life_style = "poor" # Private attribute
    def show_info(self):
        # This method prints out the information
        print(f"Name: {self.nationality}")
        print(f"Family of: {self._surname}")
        print(f"Age: {self.__life_style}")

class TariqueZia(ZiaurRahman):
    def access_private_protected(self):
        print(f"Sub Class Name: {self.nationality}")  # Accessing the public attributes
        print(f"Sub Class Family: {self._surname}") # Accessing the protected attributes
        # Note: Cannot access private attribute directly from the subclass
        # print(f"Sub Class Age: {self.__life_style}")  # This will cause an error

# Creating an object of TariqueZia
n = TariqueZia("Tarique Rahman Zia")
# Calling method to show info
n.show_info()
# Accessing protected attribute (allowed by convention) in the subclass
n.access_private_protected()
```

End of Python