

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:

1. Understanding classes and objects.

Python-এ **Classes** এবং **Objects** হল Object-Oriented Programming (OOP) এর দুটি গুরুত্বপূর্ণ ধারণা। সহজভাবে ব্যাখ্যা করা যাক:

1. ক্লাস (Class): ক্লাস একটি **ব্লুপ্রিন্ট** বা **টেমপ্লেট**। ধরুন, আপনি একটি সাবান তৈরি করতে চান। একটি **ক্লাস** হল সেই সাবানের ডিজাইন, যা আপনাকে দেখাবে কিভাবে সাবানটি দেখতে হবে, তার গন্ধ কেমন হবে, আকার কেমন হবে, এবং তার ব্র্যান্ড কী হবে।

একটি ক্লাসে আপনি নির্দিষ্ট বৈশিষ্ট্য (properties) এবং কাজ (functions) সংজ্ঞায়িত করতে পারেন।

উদাহরণ: ধরা যাক, **সাবান** একটি ক্লাস। এতে থাকবে বৈশিষ্ট্য যেমন: সাবানের রঙ, গন্ধ, আকার, ব্র্যান্ড। এবং কাজ যেমন: পরিষ্কার করা, স্নান করা।

2. অবজেক্ট (Object): অবজেক্ট হল সেই ক্লাসের একটি বাস্তব (real) উদাহরণ। আপনি যে ডিজাইন তৈরি করেছেন (ক্লাস), তার ভিত্তিতে একটি আসল সাবান তৈরি করবেন, যেটি আসলে একটি **অবজেক্ট**।

ধরা যাক, আপনি "Dove সাবান" তৈরি করেছেন। এটি হল সেই ক্লাসের একটি **অবজেক্ট**। এক ক্লাসের একাধিক অবজেক্ট হতে পারে। যেমন, আপনি "Dove সাবান" তৈরি করেছেন, আর কেউ হয়তো "Lifeboy সাবান" তৈরি করেছে, কিন্তু দুটোই একই **সাবান** ক্লাসের অবজেক্ট।

সহজ উদাহরণ:

1. ক্লাস: "সাবান" (Soap)

🌈 বৈশিষ্ট্য: রঙ, গন্ধ, আকার, ব্র্যান্ড

🔧 কাজ: পরিষ্কার করা, স্নান করা

2. অবজেক্ট:

✚ Dove সাবান (এটি "সাবান" ক্লাসের একটি অবজেক্ট)

✚ Lifeboy সাবান (এটি "সাবান" ক্লাসের একটি আরেকটি অবজেক্ট)

এখন, "সাবান" ক্লাসের ভেতরে যা যা বৈশিষ্ট্য এবং কাজ আছে, সেইগুলিকে "Dove" এবং "Lifeboy"-এর মতো আলাদা আলাদা বাস্তব সাবানে প্রয়োগ করা হয়েছে।

সারাংশ:

- ক্লাস হল একটি টেমপ্লেট বা ডিজাইন, যেটা আপনাকে কোন ধরনের অবজেক্ট তৈরি করতে হবে সেটা দেখায়।
- অবজেক্ট হল সেই ক্লাসের বাস্তব উদাহরণ যা আমরা তৈরি করি এবং ব্যবহার করি।

আরেকটি সহজ উদাহরণ: যেমন, "সাবান" ক্লাস (ক্লাস) দিয়ে আপনি "Dove", "Lifeboy", "Lux" (অবজেক্ট) তৈরি করতে পারেন।

প্রশ্নঃ Dove এবং Lifeboy সাবানের “বৈশিষ্ট্য: রঙ, গন্ধ, আকার, ব্র্যান্ড, কাজ: পরিষ্কার করা, স্নান করা” এইগুলো বিস্তারিত আমাকে লিখে জানাও। at last 50+50=100 word

একজন আমাকে বলেছেন যে, <https://www.cursor.com/> একটি অসাধারণ প্ল্যাটফর্ম, যেখানে AI-এর মাধ্যমে অটোমেটিকভাবে কোড লেখা যায়। এই সাইটটি কোডিং প্রক্রিয়াকে সহজ ও দ্রুত করে তোলে, যা ডেভেলপারদের কাজকে আরও কার্যকরী এবং সুবিধাজনক করে।

How to make a class in python:

পাইথনে একটি ক্লাস তৈরি করতে হলে, নিচের ধাপগুলো অনুসরণ করতে হবে:

1. ক্লাসের নাম নির্বাচন: প্রথমে ক্লাসের নাম নির্বাচন করতে হবে। নামটি সাধারণত প্যাসক্যাল কেস (PascalCase) ফরম্যাটে লিখতে হয়, যেখানে প্রতিটি শব্দের প্রথম অক্ষর বড় হয়।
2. ক্লাসের নামের পরে parentheses () এবং colon : দিতে হবে।
3. ক্লাস ডিফাইন করা: class শব্দটি ক্লাসের নামের আগে লিখতে হবে।

গঠন: class YourClassName():

যদি আমরা MinhazulKabir নামে ক্লাস বানাতে চাই, তবে সঠিকভাবে লিখতে হবে: class MinhazulKabir():

```
class MinhazulKabir():
    roll = 5
print(MinhazulKabir)

<class '__main__.MinhazulKabir'>
```

```

test.py x
1 class MinhazulKabir(): 1 usage
2     roll = 5
3     print(MinhazulKabir)

Run test x
"C:\Users\Minhazul Kabir\AppData\Local\F
<class '__main__.MinhazulKabir'>

```

class ব্যবহার করাঃ

```

# Defining the MinhazulKabir class
class MinhazulKabir():
    # Creating an attribute name 'roll'
    roll = 27
# Defining the ShahjalalUniversity class
class ShahjalalUniversity():
    # Creating an attribute name 'area'
    area = 321

# Creating an object 'n' from the MinhazulKabir class
n = MinhazulKabir()
# Printing the 'roll' attribute of the 'n' object
print(n.roll)

```

ব্যাখ্যা:

1. **MinhazulKabir ক্লাস:** এই ক্লাসে একটি অ্যাট্রিবিউট/ভ্যারিয়েবল roll রয়েছে যার মান 27।
2. **ShahjalalUniversity ক্লাস:** এই ক্লাসে একটি অ্যাট্রিবিউট/ভ্যারিয়েবল area রয়েছে যার মান 321।
3. **অবজেক্ট তৈরি:** n = MinhazulKabir() এর মাধ্যমে MinhazulKabir ক্লাসের একটি অবজেক্ট তৈরি করা হয়।
4. **প্রিন্ট করা:** print(n.roll) এর মাধ্যমে n অবজেক্টের roll অ্যাট্রিবিউটের মান (যা 27) প্রিন্ট করা হয়।

```

1 # Defining the MinhazulKabir class
2 class MinhazulKabir():
3     # Creating an attribute name 'roll'
4     roll = 27
5 # Defining the ShahjalalUniversity class
6 class ShahjalalUniversity():
7     # Creating an attribute name 'area'
8     area = 321
9
10 # Creating an object 'n' from the MinhazulKabir class
11 n = MinhazulKabir()
12 # Printing the 'roll' attribute of the 'n' object
13 print(n.roll)

```

Run test

"C:\Users\Minhazul Kabir\AppData\Local\Programs\Python\Python313\python.exe" D:\PythonProject\MinhazulK
27

এখানে, roll এবং area হলো member variable। ক্লাসের মধ্যে কোনো variable লিখলে তাকে member variable বলা হয়। অর্থাৎ, variable এবং member variable একই জিনিস।

ফাংশন হলে, তাকে শুধু কল দিলেই হয়। অর্থাৎ, function_name() লিখলেই কাজ হয়ে যায়। তবে, ক্লাসের জন্য তাকে আগে variable এর মধ্যে রাখা লাগে। ক্লাসকে যে variable এর মধ্যে রাখা হয়, তাকে object বলা হয়। পরবর্তীতে, ওই variable/object এর পরে ডট (.) দিয়ে member variable লিখতে হয়, যাতে ওই class এর member variable অ্যাক্সেস করা যায়।

Function & Class:

- ✚ **Function:** একটি function কল করতে হলে তার নামের সাথে parentheses (brackets) ব্যবহার করতে হয়। এখানে function এর কাজ কল করা হয়, এবং প্রয়োজন হলে argument পাস করা হয়।
- ✚ **Class:** একটি class এর object তৈরি করতে হয়। class এর মাধ্যমে object তৈরি করা হয় এবং object তৈরি করার সময় class এর নামের সাথে parentheses ব্যবহার করা হয়

2. Methods vs functions, and magic (dunder) methods.

Function এবং Method এর পার্থক্য:

Definition:

- ✚ **Function:** একটি কোডের ব্লক যা কোনো নির্দিষ্ট কাজ সম্পাদন করে এবং এটি ক্লাসের বাইরে থাকে।
- ✚ **Method:** যদি কোনো function একটি ক্লাসের মধ্যে থাকে এবং এটি ক্লাসের অবজেক্ট বা ইনস্ট্যান্স এর সাথে সম্পর্কিত হয়।

Usage:

- ✚ **Function:** এটি ক্লাস বা অবজেক্টের সাথে সম্পর্কিত না সেজন্য শুধুমাত্র কল দিলেই হয়।
- ✚ **Method:** এটি ক্লাসের অবজেক্ট বা ইনস্ট্যান্স এর সাথে সম্পর্কিত এবং অবজেক্ট তৈরি করার পরেই কল করা হয়।

Access:

- ✚ **Function:** সরাসরি ফাংশনের নাম দিয়ে কল করা যায়।
- ✚ **Method:** অবজেক্ট বা ইনস্ট্যান্স এর মাধ্যমে কল করতে হয় (যেমন: object.method())।

Self Parameter:

- ✚ **Function:** ফাংশনে সাধারণত self প্যারামিটার থাকে না।
- ✚ **Method:** মেথডে সাধারণত প্যারামিটার হিসেবে self থাকে, যা অবজেক্ট বা ইনস্ট্যান্সকে নির্দেশ করে।

ফাংশন এবং মেথড একই ধরনের কোডের ব্লক যা একটি নির্দিষ্ট কাজ সম্পাদন করে। তবে, class এর মধ্যে থাকা function-কে method বলা হয়।

ভেরিয়েবল এবং অবজেক্ট এক নয়। যখন কোনো ভেরিয়েবলে class রাখা হয়, তখন সেই ভেরিয়েবলটি object হয়ে যায়।

নিচের কোডের প্রতিটি অংশ বাংলায় ব্যাখ্যা করা হলো:

```
# Function define
def minhaz_f():
    print("function Minhaz")
```

```

# Class define
class MinhazulKabir():
    # Method define
    def minhaz_m(self):
        print("method Minhaz")
    # Class attribute 'm' initialized to 27
    m = 27

# Calling the function minhaz_f
minhaz_f()

# Creating an instance (object) of the MinhazulKabir class
a = MinhazulKabir()
# Calling the minhaz_m method on the created object 'a'
a.minhaz_m()

# Accessing the class attribute 'm' and adding 30 to it, then
printing the result
print(a.m + 30)

function Minhaz
method Minhaz
57

```

গুরুত্বপূর্ণ ব্যাখ্যা:

1. ফাংশন ডিফিনেশন (minhaz_f):

- এই ফাংশনটি যখন কল করা হয়, তখন এটি একটি মেসেজ প্রিন্ট করবে: "function Minhaz"।

2. ক্লাস ডিফিনেশন (MinhazulKabir):

- MinhazulKabir একটি ক্লাস যার মধ্যে একটি মেথড যা (minhaz_m) নামে। একটি ক্লাস অ্যাট্রিবিউট/ভ্যারিয়েবল যা (m) নামে রয়েছে।
- minhaz_m মেথডটি যখন কল করা হবে, এটি "method Minhaz" মেসেজটি প্রিন্ট করবে।
- ক্লাস অ্যাট্রিবিউট/ভ্যারিয়েবল m এর মান ২৭ সেট করা হয়েছে।

3. মেথড কল:

- a.minhaz_m() এই লাইনটি ক্লাসের অবজেক্ট a এর মাধ্যমে minhaz_m মেথড কল করে, যা "method Minhaz" প্রিন্ট করবে।

4. অ্যাট্রিবিউট/ভ্যারিয়েবল অ্যাক্সেস:

- a.m + 30 এর মাধ্যমে ক্লাস MinhazulKabir এর অ্যাট্রিবিউট/ভ্যারিয়েবল m এর মান (২৭) ৩০ এর সাথে যোগ করা হয়েছে এবং এর ফলাফল প্রিন্ট করা হয়েছে, যা হবে ৫৭।

```

1  # Function define
2  def minhaz_f(): 1 usage
3      print("function Minhaz")
4
5  # Class define
6  class MinhazulKabir(): 1 usage
7      # Method define
8      def minhaz_m(self): 1 usage
9          print("method Minhaz")
10         # Class attribute 'm' initialized to 27
11         m = 27
12
13     # Calling the function minhaz_f
14     minhaz_f()
15
16     # Creating an instance (object) of the MinhazulKabir class
17     a = MinhazulKabir()
18     # Calling the minhaz_m method on the created object 'a'
19     a.minhaz_m()
20
21     # Accessing the class attribute 'm' and adding 30 to it, then printing the result
22     print(a.m + 30)

```

মেথডে সাধারণত প্যারামিটার হিসেবে self থাকে, যা অবজেক্ট বা ইনস্ট্যান্সকে নির্দেশ করে।

পাইথনে class, member, এবং object কীভাবে কাজ করে তা কোড দিয়ে দেখানো হলো:

1. **Class (ক্লাস):** ক্লাস একটি ব্লুপ্রিন্ট যা অবজেক্ট তৈরি করতে ব্যবহৃত হয়। এটি ডাটা এবং ফাংশন ধারণ করে।
2. **Member (মেম্বর):** ক্লাসের ভিতরে থাকা বৈশিষ্ট্য (attributes/variable) এবং ফাংশন (methods) গুলো মেম্বর হিসেবে পরিচিত।
3. **Object (অবজেক্ট):** অবজেক্ট হলো ক্লাসের একটি বাস্তব উদাহরণ, যা ক্লাসের সমস্ত মেম্বরকে ব্যবহার করতে পারে। শুধুমাত্র object নামক ভ্যারিয়েবল ক্লাসের সকল কিছু ব্যবহার করতে পারে। object ছাড়া কোনো ক্লাস ব্যবহার করা যায় না।

কোড দিয়ে উদাহরণ:

```

# ক্লাস ডিফাইন করা হচ্ছে
class Car():
    # মেম্বর ভ্যারিয়েবল (attributes)
    brand = "Toyota" # এটা একটি member variable
    model = "Corolla" # এটা আরেকটি member variable

    # মেম্বর ফাংশন (method)
    def start(self): # এটা একটি method

```

```
print(f"{self.brand} {self.model} is starting.")
```

অবজেক্ট তৈরি করা হচ্ছে

```
my_car = Car() # 'my_car' হলো Car ক্লাসের একটি অবজেক্ট
```

অবজেক্টের মেম্বার ভ্যারিয়েবল ব্যবহার করা হচ্ছে

```
print(my_car.brand) # "Toyota" প্রিন্ট হবে
```

অবজেক্টের মেম্বার ফাংশন কল করা হচ্ছে

```
my_car.start() # "Toyota Corolla is starting." প্রিন্ট হবে
```

Toyota

Toyota Corolla is starting.

ব্যাখ্যা:

1. Class (ক্লাস):

class Car: এখানে Car একটি ক্লাস যা গাড়ির বৈশিষ্ট্য এবং কাজ ধারণ করে।

2. Member (মেম্বার):

Member Variables (অ্যাট্রিবিউট/ভ্যারিয়েবল): brand এবং model ক্লাসের মেম্বার ভ্যারিয়েবল বা অ্যাট্রিবিউট। এগুলি গাড়ির ব্র্যান্ড এবং মডেল সংরক্ষণ করে।

Member Function (মেথড): start(self) একটি মেম্বার মেথড, যা গাড়ির স্টার্ট করার কাজ করে।

3. Object (অবজেক্ট):

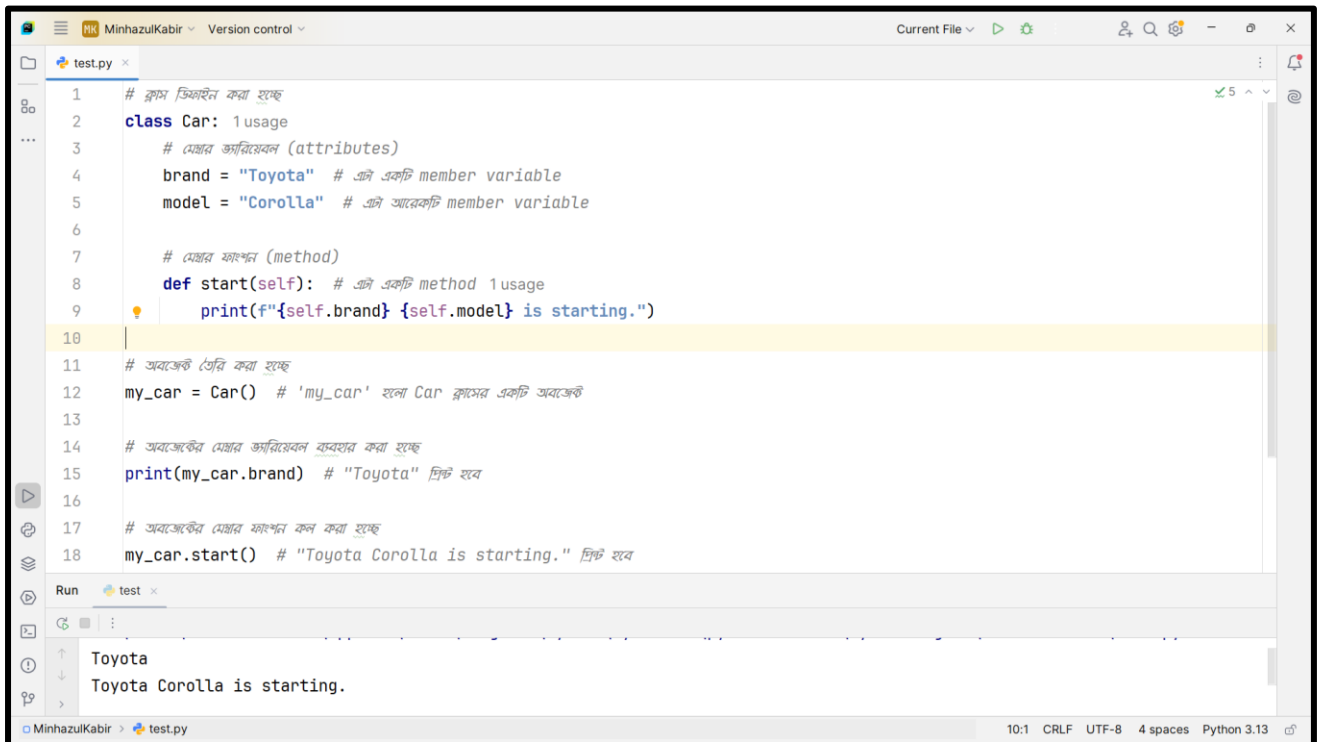
my_car = Car() এখানে my_car হলো Car ক্লাসের একটি অবজেক্ট, যা Car ক্লাসের সমস্ত বৈশিষ্ট্য (যেমন brand, model) এবং মেথড (যেমন start()) ব্যবহার করতে পারে।

আউটপুট:

Toyota

Toyota Corolla is starting.

এভাবে পাইথনে class, member, এবং object কাজ করে।



```

1  # ক্লাস ডিফাইন করা হচ্ছে
2  class Car: 1 usage
3      # মেম্বার ভ্যারিয়েবল (attributes)
4      brand = "Toyota" # এটি একটি member variable
5      model = "Corolla" # এটি আরেকটি member variable
6
7      # মেম্বার ফাংশন (method)
8      def start(self): # এটি একটি method 1 usage
9          print(f"{self.brand} {self.model} is starting.")
10
11 # অবজেক্ট তৈরি করা হচ্ছে
12 my_car = Car() # 'my_car' হলো Car ক্লাসের একটি অবজেক্ট
13
14 # অবজেক্টের মেম্বার ভ্যারিয়েবল ব্যবহার করা হচ্ছে
15 print(my_car.brand) # "Toyota" প্রিন্ট হবে
16
17 # অবজেক্টের মেম্বার ফাংশন কল করা হচ্ছে
18 my_car.start() # "Toyota Corolla is starting." প্রিন্ট হবে

```

Run test

Toyota
Toyota Corolla is starting.

পাইথনে **Magic Methods** বা **Dunder Methods** (Double Underscore Methods) হলো বিশেষ ধরনের মেথড যা ক্লাসের কিছু বিল্ট-ইন কার্যকারিতা কাস্টমাইজ করতে ব্যবহৃত হয়। এগুলোর নাম সাধারণত দুটি আন্ডারস্কোর দিয়ে শুরু এবং শেষ হয় (যেমন `__init__`, `__str__`, ইত্যাদি)। এই মেথডগুলো পাইথনের বিভিন্ন অপারেশন যেমন যোগফল, তুলনা, স্ট্রিং রূপান্তর ইত্যাদি করতে ব্যবহৃত হয়।

কিছু গুরুত্বপূর্ণ Magic Methods (Dunder Methods)

1. **`__init__(self, ...)`**: এটি ক্লাসের কনস্ট্রাক্টর বা ইনিশিয়ালাইজার মেথড, যা ক্লাসের একটি নতুন অবজেক্ট তৈরি হওয়ার সময় কল হয়। এটি অবজেক্টের প্রাথমিক অবস্থা সেট করতে ব্যবহৃত হয়।

```

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

```

2. **`__str__(self)`**: এই মেথডটি অবজেক্টের প্রিন্ট করা বা স্ট্রিং রূপে কনভার্ট করার সময় কল হয়। এটি সাধারণত অবজেক্টের একটি পাঠযোগ্য স্ট্রিং রিটার্ন করে।

```

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

    def __str__(self):
        return f"{self.name}, {self.age} years old"

```

3. **__repr__(self):** এটি সাধারণত ডিবাগিং বা রেপ্রেজেন্টেশন অ্যাক্সেসের জন্য ব্যবহৃত হয়। যখন আপনি একটি অবজেক্টের `repr()` কল করেন বা সরাসরি তাকে প্রিন্ট করার চেষ্টা করেন, তখন এটি কল হয়। এটি অবজেক্টের আরও আনুষ্ঠানিক বা সার্বজনীন স্ট্রিং রিটার্ন করে।

```
class Person():
    def __repr__(self):
        return f"Person(name={self.name}, age={self.age})"
```

4. **__add__(self, other):** এটি দুটি অবজেক্টের মধ্যে যোগফল নির্ধারণ করতে ব্যবহৃত হয় (যেমন + অপারেটর)। আপনি এটি ব্যবহার করে আপনার ক্লাসের অবজেক্টে কাস্টম যোগফল নির্ধারণ করতে পারেন।

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
```

5. **__eq__(self, other):** এটি দুটি অবজেক্টের সমতা পরীক্ষা করতে ব্যবহৃত হয় (যেমন == অপারেটর)। এটি দুইটি অবজেক্টের মধ্যে সমতা চেক করে, এবং True অথবা False রিটার্ন করে।

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

    def __eq__(self, other):
        return self.name == other.name and self.age == other.age
```

6. **__len__(self):** এই মেথডটি অবজেক্টের দৈর্ঘ্য বা সাইজ ফিরে দেয়। এটি বিশেষ করে তখন ব্যবহৃত হয় যখন `len()` ফাংশন কল করা হয়।

```
class MyList():
    def __init__(self, items):
        self.items = items

    def __len__(self):
        return len(self.items)
```

7. **__getitem__(self, key):** এটি একটি অবজেক্টের মধ্যে ইন্ডেক্সিং বা কী দিয়ে অ্যাক্সেস করার সময় কল হয় (যেমন [] অপারেটর)। এটি বিশেষভাবে ডিকশনারি বা লিস্টের মতো কন্টেইনারের জন্য ব্যবহৃত হয়।

```
class MyDict():
    def __init__(self):
        self.data = {'a': 1, 'b': 2}

    def __getitem__(self, key):
        return self.data[key]
```

8. **__setitem__(self, key, value):** এটি একটি অবজেক্টের মধ্যে ইন্ডেক্সিং বা কী দিয়ে মান সেট করার সময় কল হয় (যেমন [] অপারেটর দিয়ে মান সেট করা)।

```
class MyDict():
    def __init__(self):
        self.data = {}

    def __setitem__(self, key, value):
        self.data[key] = value
```

9. **__del__(self):** এটি একটি অবজেক্ট ডিলিট হওয়ার সময় কল হয়। সাধারণত অবজেক্টের মেমরি ক্লিনআপ বা রিসোর্স রিলিজ করার জন্য ব্যবহার হয়।

```
class MyClass():
    def __del__(self):
        print("Object is being deleted")
```

10. **__call__(self, ...):** এই মেথডটি ক্লাসের একটি অবজেক্টকে ফাংশনের মতো কল করার জন্য ব্যবহৃত হয়।

```
class MyClass():
    def __call__(self):
        print("Object is called like a function")

obj = MyClass()
obj() # This will call the __call__ method
```

উপসংহার:

Magic methods বা dunder (double underscore) methods পাইথনের একটি গুরুত্বপূর্ণ অংশ যা ক্লাসগুলোর আচরণ কাস্টমাইজ করতে সাহায্য করে। এগুলি সাধারণত অটোমেটিক্যালি কল হয় যখন কোনো নির্দিষ্ট অপারেশন বা কার্যক্রম সম্পাদিত হয় (যেমন যোগফল, সমতা, রেপ্রেজেন্টেশন ইত্যাদি)।

5. Working with constructors and destructors in Python.

Constructor হলো একটি বিশেষ মেথড যা Python এ ক্লাসের অবজেক্ট তৈরি হওয়ার সময় স্বয়ংক্রিয়ভাবে কল হয়। আপনাকে constructor কে আলাদা করে কল করতে হয় না, এটি নিজে থেকেই কাজ শুরু করে দেয়। এর মূল কাজ হলো ক্লাসের অবজেক্ট (যা সাধারণত variable হিসেবে পরিচিত) ইনিশিয়ালাইজ করা, অর্থাৎ অবজেক্ট তৈরির সময় তার প্রাথমিক মান বা ভ্যালু সেট করা। `person1 = Person("Alice", 30)` আগের কোনো অবজেক্ট বা ভ্যারিয়েবলে আমরা ক্লাসের মধ্যে মান দেই নাই। সেগুলো ফাকা ছিল।

Python এ constructor সাধারণত `__init__()` মেথডের মাধ্যমে তৈরি হয়। এটি ক্লাসের অবজেক্ট তৈরির পর, অবজেক্টের ভ্যালুগুলো ইনিশিয়ালাইজ করে। constructor এর মাধ্যমে আপনি যখন নতুন অবজেক্ট তৈরি করবেন, তখন সেই অবজেক্টের মান স্বয়ংক্রিয়ভাবে ইনিশিয়ালাইজ হয়ে যাবে।

এটি যেমন দেখানো হয়েছে:

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

person1 = Person("Alice", 30)
print(person1.name) # Output: Alice
print(person1.age)  # Output: 30
```

এখানে:

- ✚ `__init__` হল constructor মেথড।
- ✚ `self.name` এবং `self.age` হলো অবজেক্টের ভ্যালু বা প্রপার্টি, যেগুলো constructor এর মাধ্যমে ইনিশিয়ালাইজ করা হচ্ছে।
- ✚ `name` এবং `age` হল parameters, যেগুলো constructor এর মাধ্যমে অবজেক্ট তৈরির সময় দেওয়া হবে।

এখানে, `person1` অবজেক্ট তৈরি করার সময় "Alice" এবং 30 মানগুলি constructor এর মাধ্যমে `name` এবং `age` ভ্যারিয়েবল হিসেবে সেট হয়ে গেছে।

Python এ কেনো Constructor বানাতে হয়?

1. **এবং অবজেক্ট ইনিশিয়ালাইজেশন:** যখন আপনি কোনো ক্লাসের অবজেক্ট তৈরি করেন, তখন আপনি চাইবেন যে অবজেক্টটি কিছু ডিফল্ট বা প্রাথমিক মান নিয়ে তৈরি হোক। constructor এর মাধ্যমে আপনি সেই কাজটা করতে পারেন। যেমন, ধরুন আপনি একটা Car ক্লাস তৈরি করেছেন, তখন গাড়ির মডেল, রঙ বা বছরের মতো কিছু মান প্রাথমিকভাবে সেট করা উচিত। constructor আপনাকে সেই কাজগুলো সহজে করতে সাহায্য করবে।

2. **এফিসিয়েন্সি:** Constructor ব্যবহার করলে আপনি ক্লাসের মধ্যে যে কোন ইনিশিয়াল ডাটা সেট করতে পারবেন। ফলে, বারবার আলাদা করে কিছু সেট করার প্রয়োজন পড়বে না। এতে কোড লেখাও সহজ হবে এবং সময়ও বাঁচবে।

প্রয়োজনীয়তা:

- **স্বয়ংক্রিয় ডাটা সেটিং:** অবজেক্ট তৈরি হওয়ার সাথে সাথে তার প্রাথমিক মান বা ডাটা স্বয়ংক্রিয়ভাবে সেট হয়ে যাবে। আপনাকে আলাদা করে সেট করার দরকার পড়বে না।
- **কোডের সাচ্ছন্দ্য:** constructor দিয়ে কোডটা বেশি clean এবং manageable হয়। যখন আপনি অবজেক্ট তৈরি করবেন, তখন সব ডাটা ইনিশিয়ালাইজ হয়ে যাবে, যা কোডিংয়ের সময় অনেক সুবিধা দেয়।

বাস্তব উদাহরণ:

ধরি, আপনি ব্যাংকে নতুন একটি একাউন্ট খুলতে চান। ব্যাংক কর্তৃপক্ষ আগে থেকেই ঠিক করে দিয়েছে যে, নতুন একাউন্টে ন্যূনতম ২০০০ বা ৩০০০ টাকা থাকতে হবে, যা স্বয়ংক্রিয়ভাবে আপনার একাউন্টে যোগ হয়ে যাবে। এছাড়া, একাউন্ট নম্বরের একটি নির্দিষ্ট কাঠামো আছে, যেমন: প্রথমে ব্যাংক কোড, তারপর ব্রাঞ্চ কোড, এবং শেষে একটি সিরিয়াল নম্বর থাকে।

এটি ঠিক **constructor** এর মতো, যেটি একাউন্ট তৈরি হওয়ার সময় সব কিছু স্বয়ংক্রিয়ভাবে সেট করে দেয়। যখন আপনি একাউন্ট খুলবেন, তখন **constructor** স্বয়ংক্রিয়ভাবে একাউন্ট নম্বর এবং প্রাথমিক ব্যালেন্স ইনিশিয়ালাইজ করে দেয়, যেমন: ব্যাংকের কোড, ব্রাঞ্চ কোড এবং একাউন্টে প্রাথমিক টাকা (২০০০ বা ৩০০০ টাকা)।

এভাবেই, **constructor** ব্যাংকের আগে থেকে নির্ধারিত নিয়মাবলী অনুসারে কাজ করে, যাতে আপনার একাউন্ট সহজেই তৈরি হয়।

```
class BankAccount():
    # Initialize account details
    def __init__(self, bank_code, branch_code, balance):
        self.bank_code = bank_code
        self.branch_code = branch_code
        self.balance = balance
        self.account_number = f"{self.bank_code}-{self.branch_code}-..."
    # Display account information
    def display_account_info(self):
        print(f"Account Number: {self.account_number}")
        print(f"Initial Balance: {self.balance} BDT")

# Create and display account 1 info
account1 = BankAccount("XYZ", "001", 3000)
account1.display_account_info()

# Create and display account 2 info
account2 = BankAccount("XYZ", "002", 2000)
account2.display_account_info()
```

```

1 class BankAccount(): 2 usages
2     # Initialize account details
3     def __init__(self, bank_code, branch_code, balance):
4         self.bank_code = bank_code
5         self.branch_code = branch_code
6         self.balance = balance
7         self.account_number = f"{self.bank_code}-{self.branch_code}-..."
8     # Display account information
9     def display_account_info(self): 2 usages
10        print(f"Account Number: {self.account_number}")
11        print(f"Initial Balance: {self.balance} BDT")
12
13    # Create and display account 1 info
14    account1 = BankAccount( bank_code: "XYZ", branch_code: "001", balance: 3000)
15    account1.display_account_info()
16
17    # Create and display account 2 info
18    account2 = BankAccount( bank_code: "XYZ", branch_code: "002", balance: 2000)
19    account2.display_account_info()

```

Run test x

```

Account Number: XYZ-001-...
Initial Balance: 3000 BDT
Account Number: XYZ-002-...
Initial Balance: 2000 BDT

```

MinhazulKabir test.py 20:1 CRLF UTF-8 4 spaces Python 3.13

#Error

```

class man:
    def abcd(self, n, p): # Method
        print(n + p)

    def __init__(self, x, z): # Constructor
        print(x + z)

# Error: Constructor requires two arguments, but none are provided
n = man() # an error because the constructor expects arguments
n.abcd(10, 20) # Calling the abcd method with arguments 10 and 20

```

#No Error

```

class man:
    def abcd(self, n, p): # Method
        print(n + p)

    def __init__(self, x, z): # Constructor
        print(x + z)

# Creating an instance of the 'man' class with arguments 40 and 30
# for the constructor
n = man(40, 30) # Output: 70 (sum of 40 and 30)
n.abcd(10, 20) # Calling the abcd method with arguments 10 and 20

```

কনস্ট্রাক্টর (Constructor) এবং ডেস্ট্রাক্টর (Destructor) হলো দুটি বিশেষ ধরনের ফাংশন যা ক্লাসে ব্যবহৃত হয়:

1. কনস্ট্রাক্টর (Constructor):

- ✚ **কাজ:** কনস্ট্রাক্টর একটি বিশেষ ফাংশন যা যখন একটি অবজেক্ট তৈরি হয়, তখন স্বয়ংক্রিয়ভাবে কল হয়। এটি অবজেক্টের প্রাথমিক মান (values) সেট করার জন্য ব্যবহৃত হয়।
- ✚ **মুখ্য উদ্দেশ্য:** অবজেক্ট তৈরি হওয়ার সময় তার প্রপার্টি বা স্টেট ইনিশিয়ালাইজ (set) করা।

2. ডেস্ট্রাক্টর (Destructor):

- ✚ **কাজ:** ডেস্ট্রাক্টর একটি বিশেষ ফাংশন যা অবজেক্ট ধ্বংস (destroy) হওয়ার সময় স্বয়ংক্রিয়ভাবে কল হয়। এটি অবজেক্টটি মুছে ফেলার আগে কোন ক্লিনআপ বা ফাইনাল কাজ করার জন্য ব্যবহৃত হয়।
- ✚ **মুখ্য উদ্দেশ্য:** অবজেক্ট ধ্বংসের পর অবশিষ্ট কাজ বা রিসোর্স মুক্ত করা।

সারাংশ:

- ✚ কনস্ট্রাক্টর অবজেক্ট তৈরি হওয়ার সময় কাজ করে।
- ✚ ডেস্ট্রাক্টর অবজেক্ট ধ্বংস হওয়ার সময় কাজ করে।

```
class Example:
    def __init__(self, name): # Constructor:
        self.name = name
        print(f"Constructor: {self.name}")

    def __del__(self):
        print(f"Destructor: {self.name}") # Destructor:

# Creating an object
obj = Example("Hi")

# The destructor will be called automatically when the object's
work is finished (when the object is destroyed)
```

সংক্ষিপ্ত ব্যাখ্যা:

- ✚ `__init__(self, name)`: এই কনস্ট্রাক্টর মেথড অবজেক্ট তৈরি হওয়ার সময় চালু হয় এবং একটি name গ্রহণ করে সেটি সেট করে।
- ✚ `__del__(self)`: এটি ডেস্ট্রাক্টর মেথড, যা অবজেক্ট ধ্বংস হওয়ার সময় কল হয়। তবে, কখন এটি কল হবে তা নির্দিষ্ট করা কঠিন, কারণ পাইথনের গার্বেজ কালেকশন (garbage collection) ম্যানেজমেন্ট স্বয়ংক্রিয়।

পাইথনের **বিল্ট-ইন ডেকোরেটর** হল কিছু বিশেষ ডেকোরেটর যা ফাংশন বা মেথডের আচরণ পরিবর্তন করতে সাহায্য করে। এগুলি সাধারণত কোডের কার্যকারিতা বাড়াতে এবং পুনঃব্যবহারযোগ্যতা উন্নত করতে ব্যবহৃত হয়। নিচে কিছু সাধারণ **বিল্ট-ইন ডেকোরেটর** দেওয়া হলো এবং সহজভাবে তাদের ব্যাখ্যা করা হলো:

১. @staticmethod

- ✚ **ব্যাখ্যা:** এই ডেকোরেটরটি একটি মেথডকে স্ট্যাটিক (অথবা ক্লাসের সাথে সম্পর্কহীন) মেথডে পরিণত করে। এটি ক্লাসের ইনস্ট্যান্স বা অন্যান্য ভেরিয়েবল ব্যবহার না করে সাধারণভাবে কাজ করে।
- ✚ **যেমন:** ক্লাসের অবজেক্ট তৈরি না করেও এই মেথডটি কল করা যায়। **self নাই এখানে।**

```
class MyClass:
    @staticmethod
    def greet(name):
        # This static method does not require an instance or class
        # reference
        print(f"Hello, {name}!")

# You can call this method directly through the class, without
# creating an instance
MyClass.greet("Alice")
```

২. @classmethod

- ✚ **ব্যাখ্যা:** এটি একটি ক্লাস মেথড তৈরি করে। @staticmethod এর মতো, তবে এই মেথডটি প্রথম প্যারামিটার হিসেবে **ক্লাস (cls)** গ্রহণ করে, এবং ক্লাসের ভেরিয়েবল বা মেথডে অ্যাক্সেস করতে সাহায্য করে।
- ✚ **যেমন:** এটি ক্লাসের সাথে সম্পর্কিত কোনো কাজ করার জন্য ব্যবহৃত হয়। **increment(cls):** method এর বাইরে count থাকার পরেও তাকে ব্যবহার করা গিয়েছে।

```
class MyClass:
    count = 0

    @classmethod
    def increment(cls):
        # This method modifies class-level variables
        cls.count += 1
        print(f"Count: {cls.count}")

# The class method is called through the class itself
MyClass.increment()
```

৩. @property

- ✚ **ব্যাখ্যা:** এই ডেকোরেটরটি একটি মেথডকে অ্যট্রিবিউট হিসেবে তৈরি করে। এটি সাধারণত গেটার (getter) হিসেবে কাজ করে, যাতে আপনি সরাসরি একটি ভেরিয়েবল অ্যাক্সেস করার মতো ফাংশনটি ব্যবহার করতে পারেন।
- ✚ **যেমন:** কোনো ভেরিয়েবলকে মেথডের মাধ্যমে অ্যাক্সেস করা।


```

class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        # This method returns the value of the private attribute
        _radius
        return self._radius

    @property
    def area(self):
        # This method calculates and returns the area of the
        circle
        return 3.14 * (self._radius ** 2)

# Accessing the properties as if they are attributes
circle = Circle(5)
print(circle.radius)    # Output: 5
print(circle.area)      # Output: 78.5

```

8. @abstractmethod

🌟 **ব্যাখ্যা:** এটি অ্যাবস্ট্রাক্ট মেথড তৈরি করতে ব্যবহৃত হয়। যখন আপনি একটি ক্লাসে এটি ব্যবহার করেন, তখন ওই মেথডটি সাবক্লাসে ইমপ্লিমেন্ট (অথবা লিখিত) করতে হবে। এটি মূলত **Abstract Base Class (ABC)** মডিউলের অংশ।

🌟 **যেমন:** কোনো মেথডের ব্যবহার নির্ধারণ করতে, যা সাবক্লাসে পূর্ণ করতে হবে।

```

from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        # Abstract method to be implemented in subclasses
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        # Implementation of the abstract method
        return 3.14 * (self.radius ** 2)

circle = Circle(5)
print(circle.area())    # Output: 78.5

```

৫. @functools.lru_cache

🌟 **ব্যাখ্যা:** এই ডেকোরেটরটি ফাংশনের ফলাফল ক্যাশে করে রাখে, অর্থাৎ যদি কোনো ফাংশন একাধিকবার একই আর্গুমেন্টের সাথে কল হয়, তবে প্রথম কলের ফলাফল সংরক্ষণ করে রাখে এবং পরবর্তী কলগুলোর জন্য তা ব্যবহার করে, যাতে কার্যকারিতা বৃদ্ধি পায়।

🌟 **যেমন:** এটি সাধারণত পুনরাবৃত্তিমূলক কাজের জন্য ব্যবহার হয় (যেমন Fibonacci সিরিজ).

```
from functools import lru_cache

@lru_cache(maxsize=None)
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# Using caching to speed up repeated calls
print(fibonacci(10)) # Output: 55
```

৬. @setter (গেটার-সেটার মেথডের জন্য)

🌟 **ব্যাখ্যা:** যখন আপনি @property ব্যবহার করে একটি অ্যাট্রিবিউট তৈরি করেন, তখন আপনি @setter ব্যবহার করে সেটি পরিবর্তন করতে পারেন। এটি মূলত একটি মেথডের মাধ্যমে অ্যাট্রিবিউট সেট (অথবা পরিবর্তন) করার অনুমতি দেয়।

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

    @property
    def name(self):
        # Getter method to retrieve the name
        return self._name

    @name.setter
    def name(self, value):
        # Setter method to change the name
        self._name = value

person = Person("Alice")
print(person.name) # Output: Alice (getter)
person.name = "Bob" # Changes the name (setter)
print(person.name) # Output: Bob
```

উপসংহার:

পাইথনে বিল্ট-ইন ডেকোরেটর আপনার কোডকে আরও বেশি শক্তিশালী ও সহজবোধ্য করে তুলতে সাহায্য করে।
এগুলি ব্যবহারের মাধ্যমে আপনি ফাংশন বা মেথডের আচরণ পরিবর্তন বা বাড়াতে পারেন, যাতে কোড আরও পরিষ্কার
ও কার্যকরী হয়।