# ویژگی کلاس ها

## ویژگی کلاس ها

در برنامه‌نویسی شی‌گرا، کلاس‌ها ساختارهای بنیادینی برای تعریف اشیاء و مدیریت داده‌ها هستند. در این درسنامه، مفاهیم پایه‌ای کلاس در زبان پایتون و نحوه‌ی پیاده‌سازی آن‌ها را بررسی می‌کنیم.

## کلاس (Class) در پایتون

تابع `type` نوع اشیاء را مشخص می‌کند. مثال‌های زیر خروجی این تابع را نشان می‌دهند:

```
print(type("Python"))  # <class 'str'>
print(type(42))        # <class 'int'>
print(type([3, 6, 9]))  # <class 'list'>
print(type((1, 4)))     # <class 'tuple'>
print(type({'key': 'value'}))  # <class 'dict'>
```

همان‌طور که مشاهده می‌کنید، هر شیء متعلق به یک کلاس است. اکنون نحوه‌ی تعریف کلاس جدید را بررسی می‌کنیم:

```
class Vehicle:
    pass

car = Vehicle()
print(type(car))  # <class '__main__.Vehicle'>
```

در این مثال، کلاس `Vehicle` تعریف شده و یک شیء از آن ساخته‌ایم.

## نحوه ساخت کلاس در پایتون:

در پایتون، برای ساخت یک کلاس، از کلمه کلیدی `class` استفاده می‌کنیم و سپس نام کلاس را می‌نویسیم. نام کلاس معمولاً با حرف بزرگ شروع می‌شود تا تفاوت آن با سایر متغیرها یا توابع که معمولاً با حروف کوچک شروع

می‌شوند، مشخص شود. کلاس‌ها در واقع الگوهایی هستند که برای ایجاد اشیاء (Objects) از آن‌ها استفاده می‌کنیم.

### ۱. ساختار اولیه یک کلاس:

```
class ClassName:
    def __init__(self, parameter1, parameter2):
        self.attribute1 = parameter1
        self.attribute2 = parameter2
```

- **`class ClassName:`** : این خط کلاس جدیدی به نام `ClassName` تعریف می‌کند.
- **`def __init__(self, ...)`** : این تابع به نام `__init__` شناخته می‌شود و تابع سازنده (Constructor) برای کلاس است. هر زمان که شیء جدیدی از این کلاس ایجاد می‌شود، این تابع به طور خودکار فراخوانی می‌شود. هدف این تابع مقداردهی اولیه به ویژگی‌های شیء است.
- **`self.attribute = ...`** : در داخل `__init__` ، می‌توانیم ویژگی‌های شیء را با استفاده از `self` مقداردهی اولیه کنیم. `self` به شیء فعلی اشاره دارد و به این وسیله می‌توانیم به ویژگی‌ها و متدهای شیء دسترسی پیدا کنیم.

### ۲. مثال کامل از تعریف کلاس:

فرض کنید می‌خواهیم کلاسی به نام `Car` بسازیم که نمایانگر یک خودرو باشد. هر خودرو ویژگی‌هایی مانند مدل و رنگ دارد. ما همچنین می‌خواهیم یک متد برای روشن کردن موتور خودرو تعریف کنیم.

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

    def start(self):
        print(f"The {self.color} {self.model} is starting.")
```

- **`__init__(self, model, color)`** : تابع سازنده که هنگام ایجاد شیء از کلاس فراخوانی می‌شود. پارامترهای `model` و `color` برای مقداردهی اولیه به ویژگی‌های `self.model` و `self.color` استفاده می‌شوند.
- **`self.model = model`** : ویژگی `model` به شیء اختصاص داده می‌شود.
- **`self.color = color`** : ویژگی `color` به شیء اختصاص داده می‌شود.
- **`start(self)`** : متدی که وقتی فراخوانی می‌شود، پیامی به صورت قالبی با استفاده از ویژگی‌های `model` و `color` چاپ می‌کند.

### ۳. نحوه استفاده از کلاس:

برای استفاده از یک کلاس، باید ابتدا یک شیء از آن ایجاد کنیم. برای ایجاد شیء از کلاس، نام کلاس را صدا می‌زنیم و پارامترهای مورد نیاز برای `__init__` را به آن می‌دهیم. پس از ایجاد شیء، می‌توانیم متدهای آن را فراخوانی کنیم.

```
my_car = Car("Toyota", "red")  #Creating an object from the Car class
my_car.start()  # Calling the start method
```

۱. **`my_car = Car("Toyota", "red")`** : این خط یک شیء از کلاس `Car` می‌سازد و مقادیر `"Toyota"` و `"red"` را به پارامترهای `model` و `color` در تابع `__init__` می‌دهد. در نتیجه، ویژگی‌های `model` و `color` شیء `my_car` مقداردهی می‌شوند.

۲. **`my_car.start()`** : این خط متد `start()` شیء `my_car` را فراخوانی می‌کند. این متد پیامی به صورت قالبی چاپ می‌کند که در آن رنگ و مدل خودرو درج شده است.

۴. نتیجه:

```
The red Toyota is starting.
```

برای داشتن متغیرهایی که ویژگی‌های کلاس را نمایندگی کنند، از ویژگی‌های **instance** یا **کلاس** استفاده می‌کنیم.

## ویژگی‌های شیء (Instance Variables):

ویژگی‌های شیء به متغیرهایی گفته می‌شود که مختص هر شیء از کلاس هستند. این ویژگی‌ها با استفاده از `self` در داخل متدها تعریف و دسترسی پیدا می‌کنند. به این صورت که هر شیء از کلاس ویژگی‌های خود را دارد که به‌طور مجزا از سایر اشیاء عمل می‌کند.

**مثال:**

در کلاس `Book` ، ویژگی‌هایی همچون عنوان و نویسنده به هر شیء از این کلاس اختصاص داده می‌شوند:

```
class Book:
    def __init__(self, title, author):
        self.title = title  # Book title attribute
        self.author = author  # Book author attribute
```

در اینجا، `title` و `author` ویژگی‌های شیء هستند. هر شیء از کلاس `Book` ویژگی‌های خودش را خواهد داشت و می‌توان از طریق `self` به آن‌ها دسترسی داشت.

## ویژگی‌های کلاس (Class Variables):

ویژگی‌های کلاس ویژگی‌هایی هستند که در سطح کلاس تعریف می‌شوند و برای تمام اشیاء کلاس به‌صورت مشترک اعمال می‌شوند. برخلاف ویژگی‌های شیء که به هر شیء اختصاص دارند، این ویژگی‌ها برای همه اشیاء کلاس یکسان هستند و نیازی به استفاده از `self` برای دسترسی به آن‌ها نیست.

**مثال:**

در اینجا، شمارش تعداد کتاب‌هایی که از کلاس `Book` ساخته شده‌اند را به‌عنوان یک ویژگی کلاس در نظر می‌گیریم:

```
class Book:
    total_books = 0  # Class attribute that holds the total number of boo

    def __init__(self, title, author):
        self.title = title
```

```
        self.author = author
        Book.total_books += 1 # Increase the number of books each time a
```

در این مثال، `total_books` یک ویژگی کلاس است که به تمام اشیاء کلاس `Book` مشترک است. هر بار که شیء جدیدی از این کلاس ساخته می‌شود، مقدار `total_books` افزایش می‌یابد.

## نحوه دسترسی به ویژگی‌ها:

- **ویژگی‌های کلاس** می‌توانند هم از طریق شیء و هم از طریق خود کلاس دسترسی پیدا کنند. با این حال، معمولاً برای دسترسی به ویژگی‌های کلاس از خود کلاس استفاده می‌شود.

مثال استفاده:

```
# Creating objects from the Book class
book1 = Book("Python Programming", "John Doe")
book2 = Book("Learning AI", "Jane Smith")

#Accessing object attributes
print(book1.title)  # Python Programming
print(book2.author)  # Jane Smith

# Accessing class attributes
print(Book.total_books)  # 2
```

## تعریف ویژگی برای اشیاء (Property):

در پایتون می‌توان ویژگی‌هایی برای اشیاء به‌صورت داینامیک (در زمان اجرا) تعریف کرد. این ویژگی‌ها به‌طور معمول فقط برای شیء خاصی از کلاس قابل استفاده هستند و سایر اشیاء این ویژگی‌ها را نخواهند داشت.

مثال:

```
class Person:
    pass
```

```
4
5  p1 = Person()
6  p2 = Person()
7  p1.name = "Ali"
8  p1.age = 25
9
10 print(p1.name, p1.age)  # Ali 25
   print(hasattr(p2, 'name'))  # False
```

در این مثال، ویژگی‌های `name` و `age` تنها برای شیء `p1` تعریف شده‌اند و شیء `p2` آن‌ها را ندارد.

## متدها (Methods) در کلاس‌ها:

متدها توابعی هستند که در داخل کلاس‌ها تعریف می‌شوند و برای انجام عملیات مختلف روی اشیاء آن کلاس استفاده می‌شوند. متدها همیشه باید `self` را به‌عنوان پارامتر ورودی دریافت کنند تا به ویژگی‌ها و رفتارهای شیء دسترسی داشته باشند.

مثال:

```
1  class BankAccount:
2      def set_balance(self, amount):
3          self.balance = amount  ## Setting the account balance
4      def get_balance(self):
5          return self.balance  # Getting the account balance
```

در اینجا، متد `set_balance` برای تنظیم موجودی حساب استفاده می‌شود و متد `get_balance` برای دریافت موجودی حساب.

## تابع سازنده (Constructor) در پایتون:

تابع `__init__` در پایتون یک تابع ویژه است که هنگام ساخت هر شیء از کلاس فراخوانی می‌شود و مسئول مقداردهی اولیه به ویژگی‌های شیء است. این تابع می‌تواند پارامترهایی برای تعیین مقادیر پیش‌فرض ویژگی‌ها دریافت کند.

مثال:

```python
class Product:
    def __init__(self, name, price=100):
        self.name = name
        self.price = price

p1 = Product("Laptop", 1500)
p2 = Product("Mouse")

print(p1.name, p1.price)  # Laptop 1500
print(p2.name, p2.price)  # Mouse 100
```

در این مثال، مقدار پیش‌فرض برای `price` برابر 100 است. اگر در زمان ساخت شیء، مقداری برای `price` مشخص نشود، از مقدار پیش‌فرض استفاده خواهد شد.

## متغیرهای ایستا (Static Variables):

متغیرهای ایستا به متغیرهایی گفته می‌شود که به تمام اشیاء یک کلاس تعلق دارند. این ویژگی‌ها برای تمام اشیاء کلاس یکسان هستند و مقدار آن‌ها در سطح کلاس ذخیره می‌شود.

مثال:

```python
class Company:
    company_name = "Tech Corp"

c1 = Company()
c2 = Company()
print(c1.company_name)  # Tech Corp

Company.company_name = "New Tech"
print(c2.company_name)  # New Tech
```

در این مثال، تغییر مقدار `company_name` در سطح کلاس، تمام اشیاء کلاس را تحت تأثیر قرار داده است.

## متدهای ایستا (Static Methods):

متدهای ایستا به‌گونه‌ای طراحی شده‌اند که نیازی به استفاده از `self` ندارند و به‌طور مستقیم روی کلاس اجرا می‌شوند. این متدها می‌توانند بدون ایجاد شیء از کلاس فراخوانی شوند.

مثال:

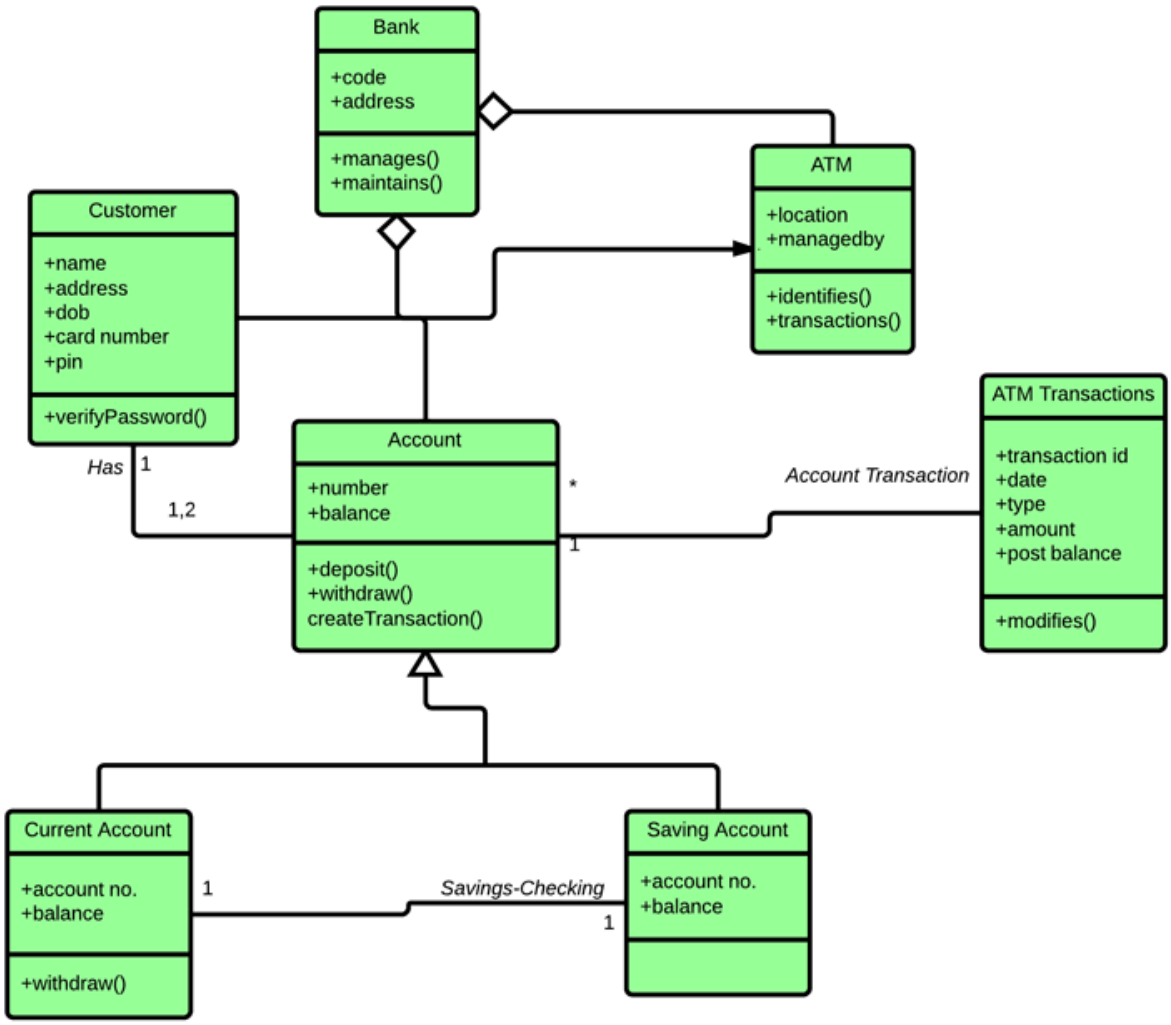
```python
class MathOperations:
    @staticmethod
    def add(x, y):
        return x + y

print(MathOperations.add(10, 20))  # 30
```

در اینجا، متد `add` به‌عنوان یک متد ایستا تعریف شده است و می‌توان آن را بدون ایجاد شیء از کلاس `MathOperations` فراخوانی کرد.

## جمع‌بندی:

- **ویژگی‌های شیء** اطلاعات مختص هر شیء را نگه می‌دارند.
- **ویژگی‌های کلاس** اطلاعات مشترک برای تمام اشیاء کلاس را ذخیره می‌کنند.
- **متدهای کلاس** عملیات مختلفی روی اشیاء انجام می‌دهند و برای دسترسی به ویژگی‌ها و رفتارهای شیء به `self` نیاز دارند.
- **متغیرهای ایستا** و **متدهای ایستا** برای اشتراک‌گذاری اطلاعات و عملیات بین تمام اشیاء کلاس استفاده می‌شوند.
- تابع `__init__` برای مقداردهی اولیه ویژگی‌های شیء به کار می‌رود.


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance