کپسوله‌سازی (Encapsulation)

کپسوله‌سازی یکی از مفاهیم اصلی شی‌گرایی است که به معنای **پنهان‌سازی جزئیات داخلی** یک شیء و **نمایش فقط رابط‌های عمومی** برای تعامل با آن شیء است. هدف اصلی کپسوله‌سازی این است که جزئیات پیاده‌سازی داخلی یک شیء از دسترس کاربران خارج شده و تنها قابلیت‌های ضروری آن در دسترس قرار گیرد.

این فرآیند باعث می‌شود که:

- کاربر فقط با رابط‌های عمومی (public) تعامل داشته باشد.
- از تغییرات در پیاده‌سازی داخلی جلوگیری شود.
- می‌توان پیاده‌سازی داخلی را تغییر داد بدون اینکه تاثیری بر بخش‌های دیگر کد داشته باشد.

استفاده از ویژگی‌ها و متدهای خصوصی و عمومی

در پایتون، به طور پیش‌فرض همه ویژگی‌ها و متدهای یک کلاس **عمومی** (public) هستند و می‌توان به آنها به راحتی از خارج از کلاس دسترسی داشت. اما در بسیاری از مواقع، نیاز است که برخی ویژگی‌ها و متدها به صورت خصوصی یا محافظت‌شده (protected) باشند.

برای **پنهان‌سازی ویژگی‌ها و متدهای کلاس**، می‌توان از قوانین خاص پایتون استفاده کرد:

1. **ویژگی‌ها و متدهای عمومی:**
   به طور معمول بدون هیچ پیشوند خاصی تعریف می‌شوند.
2. **ویژگی‌ها و متدهای خصوصی (Private):**
   ویژگی‌ها و متدهایی که نمی‌خواهیم از خارج از کلاس به آنها دسترسی داشته باشیم با پیشوند `__` (دو خط تیره) تعریف می‌شوند.
3. **ویژگی‌ها و متدهای محافظت‌شده (Protected):**
   ویژگی‌ها و متدهایی که می‌خواهیم فقط در داخل کلاس و زیرکلاس‌ها قابل دسترسی باشند، با پیشوند `_` (یک خط تیره) تعریف می‌شوند.

دسترسی به ویژگی‌ها و متدهای خصوصی با استفاده از قوانین `private` و `protected`

در پایتون، هیچ مکانیزم سخت‌گیرانه‌ای برای دسترسی به ویژگی‌ها و متدهای خصوصی وجود ندارد، اما با استفاده از پیشوندهای `__` و `_`، می‌توان این ویژگی‌ها را پنهان کرد تا از دسترسی‌های غیرمجاز جلوگیری شود.

ویژگی‌های خصوصی (Private):

- ویژگی‌هایی که با `__` شروع می‌کنند، به طور خصوصی در نظر گرفته می‌شوند. این ویژگی‌ها از دسترسی مستقیم از خارج از کلاس جلوگیری می‌کنند.

ویژگی‌های محافظت‌شده (Protected):

- ویژگی‌هایی که با `_` شروع می‌شوند، معمولاً در کلاس‌های فرزند قابل دسترسی هستند، ولی نباید به طور مستقیم از خارج از کلاس مورد استفاده قرار گیرند.

استفاده از ویژگی‌های خصوصی با استفاده از دکوریتور `@property` و `@setter`

در پایتون، برای **کنترل دسترسی به ویژگی‌های خصوصی**، می‌توان از دکوریتورهای `@property` و `@setter` استفاده کرد.

- `@property`: این دکوریتور برای تبدیل یک متد به یک ویژگی (property) استفاده می‌شود. به این ترتیب، می‌توان به متدها مانند ویژگی‌ها دسترسی داشت.
- `@setter`: این دکوریتور برای تنظیم مقدار ویژگی‌های خصوصی استفاده می‌شود.

این دکوریتورها اجازه می‌دهند که ویژگی‌های خصوصی به صورت امن تغییر کنند و کنترل بیشتری بر دسترسی به آنها داشته باشیم.

مثال‌هایی از کپسوله‌سازی با استفاده از `private` و `@property`

در این مثال، یک کلاس `Person` ایجاد می‌کنیم که ویژگی `age` را به صورت خصوصی نگهداری می‌کند. از دکوریتور `@property` برای دسترسی به این ویژگی استفاده می‌کنیم و از `@setter` برای تغییر مقدار آن استفاده می‌کنیم.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age  # ویژگی خصوصی

    # استفاده از @property برای دسترسی به ویژگی خصوصی
    @property
    def age(self):
        return self.__age

    # استفاده از @setter برای تنظیم ویژگی خصوصی
    @age.setter
    def age(self, value):
        if value < 0:
            print("سن نمی‌تواند منفی باشد!")
        else:
            self.__age = value

# ایجاد شیء از کلاس Person
person = Person("Ali", 25)

# دسترسی به ویژگی private از طریق @property
print(person.age)  # خروجی: 25

# تغییر ویژگی private از طریق @setter
person.age = 30
print(person.age)  # خروجی: 30

# تلاش برای تنظیم مقدار نادرست
person.age = -5  # خروجی: سن نمی‌تواند منفی باشد!
```

توضیحات:

- ویژگی `__age` در کلاس `Person` به صورت خصوصی تعریف شده است.

- با استفاده از دکوریتور `@property`، امکان دسترسی به این ویژگی بدون اینکه به صورت مستقیم از خارج از کلاس مورد دسترسی قرار گیرد، فراهم می‌شود.
- با استفاده از دکوریتور `@age.setter`، می‌توان مقدار جدیدی برای `age` تنظیم کرد و در عین حال اعتبارسنجی انجام داد.

مزایای کپسوله‌سازی

1. **پنهان‌سازی جزئیات داخلی:**
   کپسوله‌سازی کمک می‌کند که جزئیات پیاده‌سازی داخلی از دسترس کاربر پنهان شود، به این ترتیب، تغییرات در پیاده‌سازی داخلی تاثیر زیادی بر کدهای خارجی نخواهند داشت.
2. **کنترل دقیق بر دسترسی به داده‌ها:**
   با استفاده از `@property` و `@setter`، می‌توان به طور دقیق کنترل کرد که داده‌ها چگونه دسترسی و تغییر می‌کنند.
3. **کاهش پیچیدگی:**
   کپسوله‌سازی کد را ساده‌تر و قابل فهم‌تر می‌کند، چرا که کاربر فقط با رابط‌های عمومی درگیر است و جزئیات پیچیده پنهان می‌شوند.
4. **امنیت بیشتر:**
   با پنهان کردن ویژگی‌ها و متدهای داخلی، امنیت داده‌ها افزایش می‌یابد و از دسترسی‌های غیرمجاز به داده‌ها جلوگیری می‌شود.

📝 تمرین برای شما:

یک کلاس `BankAccount` بسازید که شامل ویژگی‌های خصوصی مانند `balance` و متدهایی برای واریز و برداشت باشد. از دکوریتور `@property` برای نمایش موجودی حساب استفاده کنید و از `@setter` برای واریز یا برداشت مبلغ استفاده کنید. به یاد داشته باشید که موجودی حساب نباید منفی شود.

کپسوله‌سازی به شما کمک می‌کند تا از **دسترسی‌های غیرمجاز** به ویژگی‌ها جلوگیری کرده و کنترل بیشتری بر نحوه دسترسی و تغییر داده‌ها داشته باشید.