📌 دستگاه‌های مدیریت ویژگی‌ها (Property Methods)

در پایتون، دکوریتورهای `property@` و `setter@` برای تبدیل متدها به ویژگی‌ها و مدیریت تغییرات ویژگی‌های خصوصی استفاده می‌شوند. این ویژگی‌ها جزء ابزارهای اصلی در **کپسوله‌سازی** به شمار می‌روند و به شما این امکان را می‌دهند که دسترسی به ویژگی‌ها را کنترل کرده و تغییرات آنها را اعتبارسنجی کنید.

✅ استفاده از دکوریتور `property@` برای تبدیل متدها به ویژگی‌ها

دکوریتور `property@` به شما این امکان را می‌دهد که یک متد را به گونه‌ای تعریف کنید که به عنوان یک ویژگی (property) عمل کند. در این صورت، وقتی به متد دسترسی پیدا می‌کنید، دیگر نیازی به فراخوانی آن به صورت تابع نخواهید داشت و می‌توانید به آن مانند یک ویژگی دسترسی پیدا کنید.

مزایای استفاده از `property@`:

- اجازه می‌دهد که متدها به شکل طبیعی‌تری استفاده شوند، به طوری که به نظر می‌رسد ویژگی‌ها هستند.
- می‌توانید بر خروجی متدها کنترل داشته باشید و محاسباتی را انجام دهید بدون اینکه از کاربر خواسته شود آن را مانند یک متد فراخوانی کند.

مثال استفاده از `property@`:

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

    # استفاده از @property برای تبدیل متد به ویژگی
    @property
    def radius(self):
        return self.__radius

    # استفاده از @property برای محاسبه مساحت
    @property
    def area(self):
        return 3.14 * self.__radius * self.__radius

# ایجاد شیء از کلاس Circle
circle = Circle(5)

# دسترسی به ویژگی radius
print(circle.radius)  # خروجی: 5

# دسترسی به ویژگی area (که به صورت متد محاسبه شده)
print(circle.area)  # خروجی: 78.5
```

در این مثال:

- `radius` به صورت یک ویژگی قابل دسترسی است، به طوری که نیازی به فراخوانی آن به عنوان یک متد نیست.
- `area` هم به عنوان یک ویژگی محاسبه می‌شود و از آن به صورت طبیعی (نه متد) استفاده می‌شود.

✅ استفاده از `setter@` برای تغییر مقادیر ویژگی‌های خصوصی

دکوریتور `setter@` به شما این امکان را می‌دهد که تغییرات در ویژگی‌های خصوصی را کنترل کنید و از اعتبارسنجی برای اطمینان از صحت داده‌ها قبل از اعمال تغییرات استفاده کنید. به عبارت دیگر، زمانی که بخواهید ویژگی‌های خصوصی را تغییر دهید، می‌توانید منطق خاصی برای آن در نظر بگیرید (مثلاً جلوگیری از وارد کردن مقادیر نامعتبر).

مزایای استفاده از `setter@`:

- از اعتبارسنجی داده‌ها هنگام تغییر ویژگی‌ها حمایت می‌کند.
- می‌توانید تغییرات را در داده‌ها کنترل کرده و از بروز مشکلات جلوگیری کنید.

مثال استفاده از `setter@`:

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

    # استفاده از @property برای تبدیل متد به ویژگی
    @property
    def radius(self):
        return self.__radius

    # استفاده از @setter برای تغییر مقدار radius
    @radius.setter
    def radius(self, value):
        if value < 0:
            print("خطا: شعاع نمی‌تواند منفی باشد.")
        else:
            self.__radius = value

# ایجاد شیء از کلاس Circle
circle = Circle(5)

# تغییر شعاع با استفاده از setter
circle.radius = 10  # تغییر شعاع به 10
print(circle.radius)  # خروجی: 10

# تلاش برای تغییر شعاع به مقدار منفی
circle.radius = -3  # خروجی: خطا: شعاع نمی‌تواند منفی باشد.
```

در این مثال:

- با استفاده از `radius.setter@`، اجازه می‌دهیم که شعاع دایره تغییر کند.
- اگر مقدار وارد شده منفی باشد، پیامی به کاربر نشان داده می‌شود که از تغییرات نامعتبر جلوگیری می‌کند.

✅ مزایا و کاربردهای استفاده از `@property` و `@setter` در کپسوله‌سازی

1. **کنترل دسترسی به ویژگی‌ها:** با استفاده از `@property`، می‌توانید ویژگی‌های خصوصی را پنهان کرده و اجازه دهید که به صورت کنترل‌شده به آنها دسترسی پیدا شود. به این ترتیب، نیازی به نوشتن متدهایی برای دسترسی و تغییر ویژگی‌ها نخواهید داشت و همه چیز به صورت طبیعی و به کمک دکوریتورها انجام می‌شود.
2. **اعتبارسنجی داده‌ها:** با استفاده از `@setter`، می‌توانید هرگونه تغییرات در ویژگی‌های خصوصی را کنترل کرده و از اعتبارسنجی داده‌ها اطمینان حاصل کنید. این کار به خصوص در زمان‌هایی که ویژگی‌های حساس دارید (مثل سن، قیمت و غیره) بسیار مفید است.
3. **ساده‌سازی کد:** با استفاده از این دکوریتورها، کد ساده‌تر و قابل فهم‌تر می‌شود. به جای اینکه نیاز به نوشتن متدهای متعدد برای دسترسی به ویژگی‌ها یا تغییر آنها باشد، همه چیز به صورت طبیعی و با استفاده از ویژگی‌ها انجام می‌شود.
4. **پنهان‌سازی پیچیدگی‌ها:** با استفاده از `@property`، می‌توانید محاسبات پیچیده‌ای که نیاز به انجام در پس‌زمینه دارند را پنهان کرده و فقط نتیجه نهایی را به کاربر نشان دهید.

📝 تمرین برای شما:

یک کلاس `BankAccount` بسازید که ویژگی `balance` را به صورت خصوصی نگهداری کند. از دکوریتور `@property` برای نمایش موجودی حساب استفاده کنید و از `@setter` برای واریز و برداشت مبلغ استفاده کنید. از اعتبارسنجی داده‌ها برای جلوگیری از برداشت بیش از موجودی حساب استفاده کنید.

استفاده از **دستگاه‌های مدیریت ویژگی‌ها** (Property Methods) به شما این امکان را می‌دهد که دسترسی به ویژگی‌ها را به شکل طبیعی و امن کنترل کرده و از اعتبارسنجی داده‌ها حمایت کنید، که این کار کمک شایانی به **کپسوله‌سازی** و **بهبود امنیت داده‌ها** می‌کند.