جزوه برنامه نویسی پیشرفته

امیرحسین قلی زاده

دانشگاه علوم و فنون آریان ترم دوم 1403 – 1404

جزوه آموزشی – جلسه سوم: decorator ها

استاد: امیرحسین قلی زاده | **دانشگاه:** علوم و فنون آریان | **نیمسال تحصیلی:** نیمسال دوم 1403 - 1404

جزوه آموزشی: Decorator ها در پایتون

تعریف ابتدایی

دکوراتورها (Decorators) توابعی هستند که برای **تغییر رفتار توابع یا کلاس‌های دیگر** بدون تغییر در کد آن‌ها استفاده می‌شوند. آن‌ها از مفاهیم **توابع مرتبه بالا (higher-order functions)** و **توابع درون‌تو در پایتون** بهره می‌برند.

کاربرد اصلی

- افزودن ویژگی به یک تابع یا کلاس بدون دستکاری مستقیم آن
- پیاده‌سازی اصول (Don't Repeat Yourself) **DRY**
- استفاده در **لاگ‌گیری، احراز هویت، بررسی زمان اجرا، مدیریت دسترسی** و...

نحوه ساخت یکdecorator

گام 1: تعریف یک تابع ساده

```
1. def say_hello():
2.     print("Hello!")
3.
```

گام 2: تعریف یک decorator ساده

```
def my_decorator(func):
    def wrapper():
        print("قبل از اجرای تابع")
        func()
        print("بعد از اجرای تابع")
    return wrapper

```

گام 3: استفاده از decorator

```
decorated_func = my_decorator(say_hello)
decorated_func()

```

خروجی:

```
   قبل از اجرای تابع
Hello!
   بعد از اجرای تابع

```

استفاده از سینتکس@decorator

پایتون یک سینتکس ساده‌تر برای اعمال دکوراتور فراهم کرده است:

```
@my_decorator
def say_hello():
    print("Hello!")

```

این دقیقاً معادل است با:

```
say_hello = my_decorator(say_hello)

```

دکوراتور با آرگومان

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print("قبل از تابع")
        result = func(*args, **kwargs)
```

```
        print("بعد از تابع")
        return result
    return wrapper

@my_decorator
def greet(name):
    print(f"سلام، {name}!")

```

خروجی:

```
قبل از تابع
سلام، علی!
بعد از تابع

```

مثال: محاسبه زمان اجرای تابع با decorator

```
import time

def timer(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"زمان اجرا: {end - start} ثانیه")
        return result
    return wrapper

@timer
def slow_function():
    time.sleep(2)
    print("تابع اجرا شد.")

slow_function()

```

استفاده از functools.wraps برای حفظ متادیتا

بدون استفاده از functools.wraps، اطلاعات تابع اصلی مانند نام و داک‌استرینگ از بین می‌رود.

```
from functools import wraps

def my_decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        return func(*args, **kwargs)
    return wrapper
```

```
8.
```

دکوراتور با پارامتر (سطح پیشرفته‌تر)

```python
def repeat(n):
    def decorator(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                func(*args, **kwargs)
        return wrapper
    return decorator

@repeat(3)
def say_hi():
    print("سلام!")

say_hi()

```

خروجی:

```
سلام!
سلام!
سلام!

```

مثال واقعی: بررسی احراز هویت

```python
def authenticated(func):
    def wrapper(user, *args, **kwargs):
        if not user.get("is_authenticated"):
            print("دسترسی غیرمجاز!")
            return
        return func(user, *args, **kwargs)
    return wrapper

@authenticated
def view_dashboard(user):
    print(f"خوش آمدید، {user['name']}!")

user1 = {"name": "Amir", "is_authenticated": True}
user2 = {"name": "Ali", "is_authenticated": False}

view_dashboard(user1)  # مجاز
view_dashboard(user2)  # غیرمجاز

```

جمع‌بندی

مفاهیم کلیدی	توضیح
دکوراتور چیست؟	تابعی که رفتاری را به تابع دیگر اضافه می‌کند
استفاده اصلی	لاگ‌گیری، احراز هویت، کش، مدیریت دسترسی
سینتکس	@decorator_name قبل از تعریف تابع
مزیت	تمیز بودن کد، قابلیت استفاده مجدد، رعایتDRY