مفهوم دکوراتورها در پایتون

پیش از شروع بحث در مورد دکوراتورها، ابتدا بیایید مفاهیم پایه‌ای زبان پایتون را مرور کنیم.

توابع به‌عنوان اشیاء

در پایتون، همه‌چیز، از جمله توابع، به‌عنوان شیء در نظر گرفته می‌شود. این یعنی توابع می‌توانند به‌عنوان ورودی به توابع دیگر داده شوند یا به‌عنوان خروجی از توابع برگشت داده شوند.

مثال:

```
def f(g):
    g(10)
    return g

def g(a):
    print(a)

f(g)(100)
```

خروجی:

```
10
100
```

در این مثال، ابتدا تابع `f` با تابع `g` فراخوانی می‌شود و عدد ۱۰ چاپ می‌شود. سپس تابع `g` به‌عنوان خروجی تابع `f` برگشت داده شده و عدد ۱۰۰ را چاپ می‌کند.

توابع در هر محدوده‌ای

در پایتون، توابع می‌توانند در هر محدوده‌ای (scope) تعریف شوند. این ویژگی به شما این امکان را می‌دهد که توابع را به‌طور دینامیک و در هر جایی تعریف کنید.

مثال:

```
def poly_builder(coeffs):
    def p(x):
        return sum(c * (x ** i) for i, c in enumerate(coeffs))
    return p

f = poly_builder([1, 1])  # x + 1
print(f(-1))
f = poly_builder([-2, 0, 1])  # x^2 - 2
print(f(5))
```

خروجی:

```
0
23
```

در اینجا تابع `poly_builder` یک چندجمله‌ای جدید می‌سازد که قادر است برای هر مقدار x، مقدار آن را محاسبه کند.

استفاده از دکوراتورها

دکوراتورها در پایتون ابزارهایی هستند که امکان افزودن رفتارهای اضافی به توابع یا متدها بدون تغییر در کد اصلی آن‌ها را فراهم می‌کنند.

تعریف دکوراتور

در پایتون، دکوراتور تابعی است که یک تابع دیگر را به‌عنوان ورودی دریافت می‌کند و تابعی جدید بر اساس آن می‌سازد.

مثال:

```
def check_int(f):
    def wrapper(x):
```

```
        if not isinstance(x, int):
            raise ValueError("The argument must be an integer.")
        return f(x)
    return wrapper

@check_int
def add(a):
    return a + a

print(add(4))  # Output: 8
print(add("hello"))  # Throw Exception: The argument must be an integer.
```

در اینجا دکوراتور `check_int` بررسی می‌کند که آیا ورودی تابع از نوع `int` است یا خیر.

کاربرد دکوراتورها

کنترل نوع آرگومان‌ها

دکوراتورها به شما این امکان را می‌دهند که قبل از اجرای تابع اصلی، نوع ورودی‌ها را بررسی کنید و از ارسال مقادیر نامناسب جلوگیری کنید.

مثال:

```
def type_check(check_class):
    def decorator(func):
        def wrapper(x):
            if not isinstance(x, check_class):
                return f"Argument must be {check_class.__name__}"
            return func(x)
        return wrapper
    return decorator

@type_check(int)
def square(x):
    return x * x
```

```
print(square(5))  # Output: 25
print(square("5"))  # Output: Argument must be int
```

در اینجا دکوراتور `type_check` بررسی می‌کند که ورودی تابع از نوع `int` باشد.

شمارش تعداد فراخوانی‌ها

دکوراتورها می‌توانند تعداد دفعات فراخوانی یک تابع را شمارش کنند و این داده‌ها را ذخیره کنند.

مثال:

```
call_counter = {}

def count_calls(func):
    def wrapper(*args, **kwargs):
        call_counter[func.__name__] = call_counter.get(func.__name__, 0)
        return func(*args, **kwargs)
    return wrapper

@count_calls
def f(x):
    return x

@count_calls
def g(x, y):
    return x + y

f(1)
g(1, 2)

print(call_counter)  # Output: {'f': 1, 'g': 1}
```

نکات پیشرفته‌تر در دکوراتورها

دکوراتورهای با ورودی‌های متغیر

گاهی ممکن است بخواهید دکوراتورهایی بسازید که با توجه به ورودی‌های خاص تغییر کنند.

مثال:

```python
def decorator_builder(check_class):
    def decorator(func):
        def wrapper(x):
            if not isinstance(x, check_class):
                return f"Argument must be {check_class.__name__}"
            return func(x)
        return wrapper
    return decorator

@decorator_builder(int)
def double(x):
    return x * 2

print(double(4))  # Output: 8
print(double("4"))  # Output: Argument must be int
```

استفاده از کلاس به‌عنوان دکوراتور

کلاس‌ها نیز می‌توانند به‌عنوان دکوراتور استفاده شوند. این به شما این امکان را می‌دهد که رفتار دکوراتورها را به‌طور پویا تنظیم کنید.

مثال:

```python
class MyDecorator:
    def __init__(self, message):
        self.message = message

    def __call__(self, func):
        def wrapper():
            print(self.message)
            return func()
        return wrapper

```

```
@MyDecorator("Hello, World!")
def my_func():
    print("Function is running!")

my_func()
```

خروجی:

```
Hello, World!
Function is running!
```

دکوراتورها ابزار قدرتمندی در پایتون هستند که به شما امکان می‌دهند رفتار توابع را به‌طور دینامیک تغییر دهید، کنترل‌های اضافی اعمال کنید و کد خود را تمیزتر و قابل نگهداری‌تر کنید.