PYTHON BASICS
装饰器

2019-05-12

STEVEN WANG

上节总结

类型	创建方法	用处
生成器	用生成函数 + **yield** 用生成表达式 + **()**	一次循环
可迭代对象	用解析表达式	多次循环
迭代器	用类实现 **iter()** 和 **next()** 用生成器	一次循环
专用迭代器	使用 **itertools** 包	花式循环

装饰器

- 装饰器 101
- 装饰器知识点
- 装饰器案例

装饰器


```
def meat(food='鸡肉饼'):
    print(food)
```

```
burger = meat
burger()
```

```
def vegetable(func):
    def wrapper():
        print('西红柿')
        func()
        print('沙拉菜')
```

```
burger = vegetable(meat)
burger()
```

```
def bread(func):
    def wrapper():
        print('<面包>')
        func()
        print('<面包>')
    return wrapper
```

```
burger = bread(vegetable(meat))
burger()
```

复习函数

函数赋值给变量

```
def f(args):
```

```
f_var = f
```

函数里返回函数

```
def f_outer(args):
    def f_inner():
        do sth on args
    return f_inner
```

```
f_var = f_outer(args)
```

函数储存到容器

```
fn(args)
```

```
f2 =lambda args:expr
def f1(args):
```

```
f_list = [f1, f2, …, fn]
```

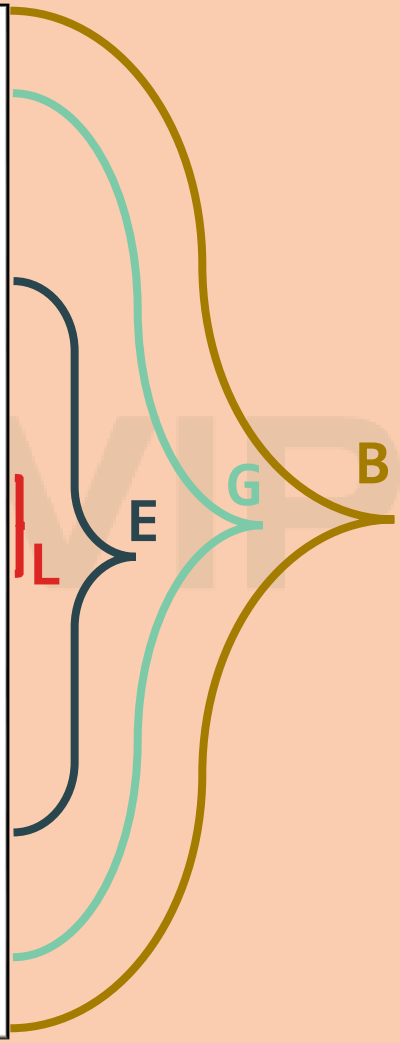
函数传递给函数

```
def f(args):
def F(func):
```

```
f_var = f
F(f_var)
```

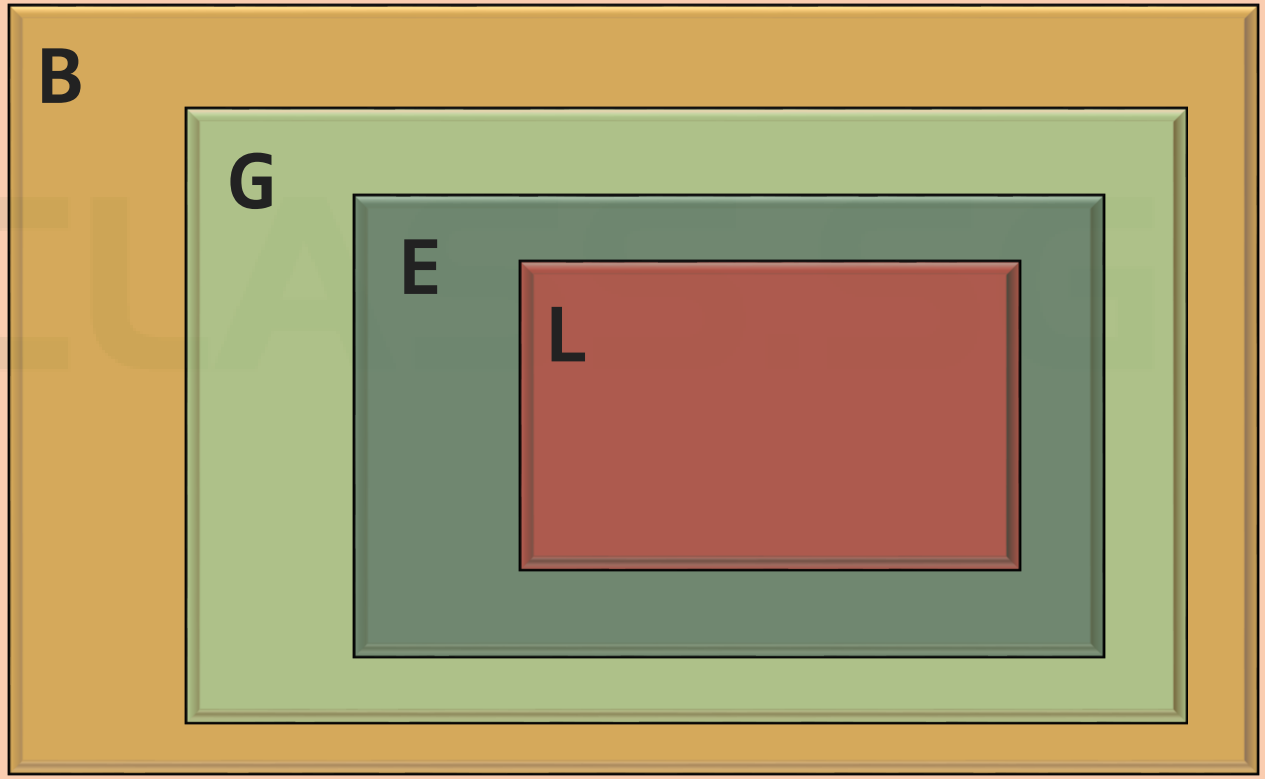
复习 LEGB 作用域


```
x = 'global x'

def outer():
    x = 'enclosed x'

    def inner():
        x = 'local x'
        print(x)

    inner()
    print(x)

outer()
print(x)
```



Local：本地 ← LEGB → Global：全局

Enclosed：内嵌 ← LEGB → Built-in：自带



BGEL = 倍感饿了

闭包到装饰器

1. 闭包通常是嵌套函数 (nested function) 的结构。
2. 该结构由外函数 (outer function) 嵌套内函数 (inner function)。
3. 内函数必须引用非本地 (non-local) 变量。
4. 外函数必须返回内函数。

闭包

```
def outer(msg):
    def inner():
        print(msg)
    return inner
```

几乎装饰器

```
def decorator(msg):
    def wrapper():
        print(msg)
    return wrapper
```

装饰器

```
def decorator(func):
    def wrapper():
        return func()
    return wrapper
```

装饰器就是一个高阶函数，其输入是函数，其输出也是函数。

decorator　　func　　wrapper

第一个装饰器

装饰器 (decorator) 在不改变原函数的条件下，增加额外功能。它可看做是给原函数做了重新包装 (wrapper)。

原函数

```
def greet():
    print('Hello!')
```

装饰器

```
def decorator(func):
    def wrapper():
        print(f"Executed before function '{func.__name__}'")
        func()
        print(f"Executed after function '{func.__name__}'")
    return wrapper
```

```
greet = decorator(greet)
```

等价

```
@decorator
def greet():
    print('Hello!')
```

```
greet()
```

```
Executed before function 'greet'
Hello!
Executed after function 'greet'
```

装饰器

- 装饰器 101
- 装饰器知识点
- 装饰器案例

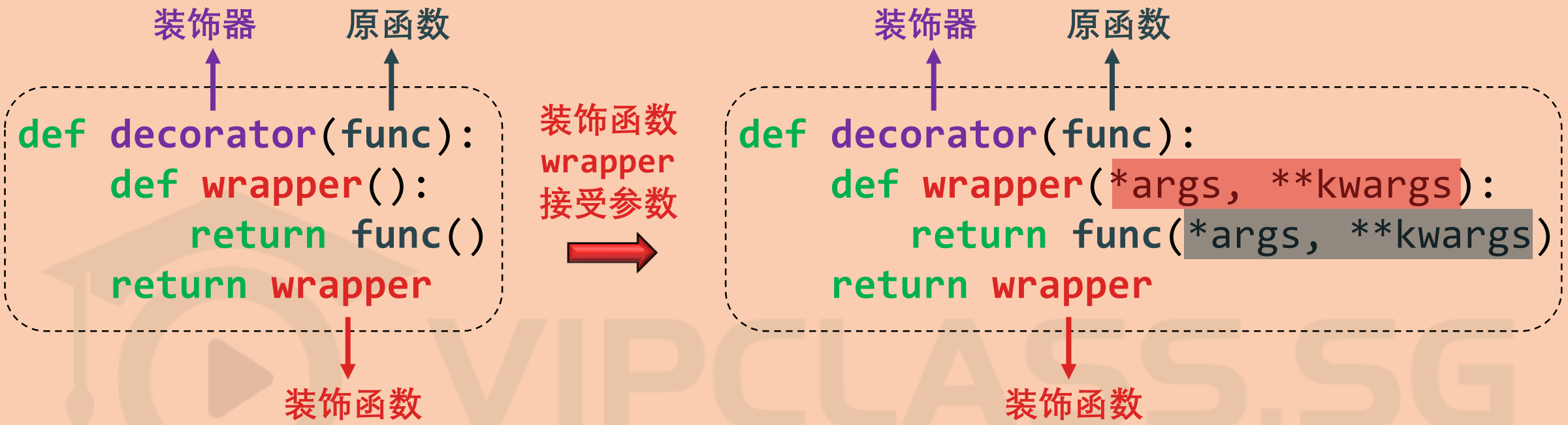
多个装饰器

从多个方面“修饰”原函数需要多个装饰器，而装饰器是有序的。

```
@bread
@vegetable
def meat(food='鸡肉饼'):
    print(food)
```

```
@vegetable
@bread
def meat(food='鸡肉饼'):
    print(food)
```

装饰函数接受参数



在装饰函数 wrapper 的定义中，*args 和 **kwargs 代表收集所有的位置参数和关键字参数，并存储在 args 和 kwargs 中。

在函数 func 被调用时，*args 和 **kwargs 代表将所有位置参数（打包成元组）和关键字参数（打包成字典）打散。

重新找回身份


```
def greet():
    print('Hello!')
```

```
def decorator(func):
    def wrapper():
        func()
    return wrapper

@decorator
def greet():
    print('Hello!')
```

```
def decorator(func):
    @functools.wraps(func)
    def wrapper():
        func()
    return wrapper

@decorator
def greet():
    print('Hello!')
```

greet.__name__
greet

greet.__name__
wrapper

greet.__name__
greet

在自己编写的所有装饰器中都使用 functools.wraps

装饰器接受参数

保留小数点两位

装饰函数接受参数

```
def money_format(func):
    def wrapper(*args, **kwargs):
        return f'{func(*args, **kwargs):.2f}'
    return wrapper
```

装饰器接受参数

添加货币单位

```
def currency_unit(curr):
    def money_format(func):
        def wrapper(*args, **kwargs):
            return f'{func(*args, **kwargs):.2f}' + ' ' + curr
        return wrapper
    return money_format
```

装饰器

- 装饰器 101
- 装饰器知识点
- 装饰器案例

装饰器实例


```
import functools

def decorator(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        # do something before
        value = func(*args, **kwargs)
        # do something after
        return value
    return wrapper
```

- 查错装饰器
- 计时装饰器
- 日志装饰器

总结

装饰器	解释
定义	输入为函数输出为函数的函数
用处	为原函数添加额外功能，重新包装
功能	多个装饰器，装饰函数接收参数，装饰器接收参数
实例	查错、计时、日志

用 `functools.wraps`

完结!


VIPCLASS.SG
small class, world class
终身学习
快乐学习
王圣元 Steven Wang
微信公众号：王的机器