函数装饰器 **decorators**

1. 定义：在不改变原函数的调用以及内部代码情况下，为其添加新功能的函数。
2. 语法

```
def 函数装饰器名称(func):
    def 内嵌函数(*args, **kwargs):
        需要添加的新功能
        return func(*args, **kwargs)
    return wrapper

@ 函数装饰器名称
def 原函数名称(参数):
        函数体

原函数(参数)
```

3. 本质：使用“@函数装饰器名称”修饰原函数，等同于创建与原函数名称相同的变量，关联内嵌函数；故调用原函数时执行内嵌函数。

   原函数名称 = 函数装饰器名称（原函数名称）

4. 装饰器链：

   一个函数可以被多个装饰器修饰，执行顺序为从近到远。

"""

```
    装饰器
"""
# 需求变化:在以下两个函数中,增加新功能(在控制台中打印函数名称).
"""
def say_hello():
    print("hello")
def say_goodbye():
    print("goodbye")
say_hello()
say_goodbye()
"""
# 缺点:新增加的功能,定义了多次.
"""
```

```
def say_hello():
    print("say_hello")
    print("hello")
def say_goodbye():
    print("say_goodbye")
    print("goodbye")
say_hello()
say_goodbye()
"""
#行为相同,数据不同.
#提取相同的行为
#缺点:增加新功能,修改原有函数内部(代码可读性不高).
"""
def print_func_name(func):
    print(func.__name__)# 函数 --> 名称
def say_hello():
    print_func_name(say_hello)
    print("hello")
def say_goodbye():
    print_func_name(say_goodbye)
    print("goodbye")
"""
"""
# 新功能
def print_func_name(func):
    print(func.__name__)
# 旧功能
def say_hello():
    print("hello")
```

```
def say_goodbye():
    print("goodbye")
# 旧功能 = 新功能 + 旧功能
say_hello = print_func_name + say_hello

say_hello()
say_goodbye()
"""
# 缺点:定义完旧功能,需要在旧功能下面用内部函数覆盖.
"""
def print_func_name(func): # func --> say_hello
    def wrapper():
        # 定义新功能
        print(func.__name__)
        # 调用旧功能
        func()
    return wrapper
# 旧功能
def say_hello():
    print("hello")
def say_goodbye():
    print("goodbye")
# 旧功能 = 新功能 + 旧功能
# 旧功能 = 新功能(旧功能) # 返回值是内部函数(新功能 + 旧功能)
say_hello = print_func_name(say_hello)
say_goodbye = print_func_name(say_goodbye)
say_hello()# 调用内部函数
say_goodbye()
"""
```

```
#缺点:如果参数不同，会异常·
"""
def print_func_name(func): # func --> say_hello
    def wrapper():
        # 定义新功能
        print(func.__name__)
        # 调用旧功能
        func()
    return wrapper
# 旧功能
@print_func_name # say_hello =
print_func_name(say_hello)
def say_hello():
    print("hello")
@print_func_name # say_goodbye =
print_func_name(say_goodbye)
def say_goodbye():
    print("goodbye")
say_hello()# 调用内部函数
say_goodbye()
"""
#缺点：旧功能的返回值丢失了
"""
def print_func_name(func): # func --> say_hello
    def wrapper(*args):# 星号元组形参
        # 定义新功能
        print(func.__name__)
        # 调用旧功能
        func(*args)# 序列实参
```

```
    return wrapper
# 旧功能
@print_func_name# say_hello =
print_func_name(say_hello)
def say_hello():
    print("hello")
    return 1
@print_func_name# say_goodbye =
print_func_name(say_goodbye)
def say_goodbye(name):
    print(name,"goodbye")
    return 2
print(say_hello())# 调用内部函数
print(say_goodbye("qtx"))
"""
def print_func_name(func): # func --> say_hello
    def wrapper(*args,**kwargs):
        # 定义新功能
        print(func.__name__)
        # 调用旧功能
        return func(*args,**kwargs)
    return wrapper
# 旧功能
@print_func_name# say_hello = print_func_name(say_hello)
def say_hello():
    print("hello")
    return 1
@print_func_name# say_goodbye =
print_func_name(say_goodbye)
```

```
def say_goodbye(name):
    print(name,"goodbye")
    return 2
print(say_hello())# 调用内部函数
print(say_goodbye("qtx"))
```

pycharm 常用快捷键

1、Ctrl + Enter：在下方新建行但不移动光标；

2、Shift + Enter：在下方新建行并移到新行行首；

3、Ctrl + /：注释(取消注释)选择的行；

4、Ctrl + Alt + L：格式化代码(与 QQ 锁定热键冲突，关闭 QQ 的热键)；

5、Ctrl + Shift + +：展开所有的代码块；

6、Ctrl + Shift + -：收缩所有的代码块；

7、Ctrl + Alt + I：自动缩进行；

8、Alt + Enter：优化代码，提示信息实现自动导包；

9、Ctrl + Shift + F：高级查找；

10、Alt + Shift + Q：更新代码到远程服务器；

11、Ctrl + N 查找所有的类的名称

12、Ctrl + Shift + N 查找项目中的任何文件