day13

今日内容

- 作业题（21题） 【看自己 学号+1 作业的22题+评论】
- 装饰器
- 推导式
- 模块【可选】

内容回顾

1. 函数

- 参数
  - def (a1,a2):pass
  - def (a1,a2=None):pass 默认参数推荐用不可变类型，慎用可变类型。
  - def(*args,**kwargs):pass
  - 注意：位置参数 > 关键字参数
  - 面试题
    - 函数可以做参数【知识点】。

```
def func(arg):
    arg()
def show():
    pass
func(show)
```

    - 函数的参数传递的是什么？【内存地址=引用 or 值】

```
v = [11,2,33,4]

def func(arg):
    print(id(arg)) # 列表内存地址

print(id(v)) # 列表内存地址
func(v)

# 传递的是内存地址。
```

    - *args和**kwargs的作用
- 返回值
  - 常见数据类型可以返回

- 函数也可以返回

```
def func():
    def inner():
        pass
    return inner

v = func()
```

- 特殊
  - 默认没返回就是None
  - return 1，2，3 等价于 return (1,2,3,4,)

- 执行函数
  - 函数不被调用，内部代码永远不执行。

```
def func():
    return i
func_list = []
for i in range(10):
    func_list.append(func)

print(i) # 9
v1 = func_list[4]()
v2 = func_list[0]()
```

```
func_list = []
for i in range(10):
    # func_list.append(lambda :x) # 函数不被调用，内部永远不执行（不知道是什么。）
    func_list.append(lambda :i) # 函数不被调用，内部永远不执行（不知道是什么。）

print(func_list)

func_list[2]()=
```

  - 执行函数时，会新创建一块内存保存自己函数执行的信息 => 闭包

```
def base():
    return i

def func(arg):
    def inner():
        return arg
    return inner

base_list = [] # [base,base,]
func_list = [] # [由第一次执行func函数的内存地址，内部arg=0 创建的inner函数，有arg=1的
inner函数 ]
for i in range(10): # i = 0 , 1
    base_list.append(base)
    func_list.append(func(i))
```

```
# 1. base_list 和 func_list中分别保存的是什么?
"""
base_list中存储都是base函数。
func_list中存储的是inner函数，特别要说的是每个inner是在不同的地址创建。
"""
# 2. 如果循环打印什么?
for item in base_list:
    v = item() # 执行base函数
    print(v) # 都是9
for data in func_list:
    v = data()
    print(v) # 0 1 2 3 4
```

总结:

- 传参：位置参数 > 关键字参数
- 函数不被调用，内部代码永远不执行。
- 每次调用函数时，都会为此次调用开辟一块内存，内存可以保存自己以后想要用的值。
- 函数是作用域，如果自己作用域中没有，则往上级作用域找。

2. 内置和匿名函数（精英）

- 内置函数
- 匿名函数

3. 模块

- getpass
- random
- hashlib

内容详细

1. 作业题讲解

2.装饰器

```
v = 1
v = 2
# ########################
def func():
    pass
v = 10
v = func

# ##########################
def base():
    print(1)

def bar():
```

```
    print(2)

bar = base
bar()
```

```
def func():
    def inner():
        pass
    return inner

v = func()
print(v) # inner函数
# ########################################
def func(arg):
    def inner():
        print(arg)
    return inner

v1 = func(1)
v2 = func(2)

# ########################################
def func(arg):
    def inner():
        arg()
    return inner

def f1():
    print(123)

v1 = func(f1)
v1()
# ########################################
def func(arg):
    def inner():
        arg()
    return inner

def f1():
    print(123)
    return 666

v1 = func(f1)
result = v1() # 执行inner函数 / f1含函数 -> 123
print(result) # None
# ########################################
def func(arg):
    def inner():
        return arg()
    return inner

def f1():
    print(123)
```

```
    return 666

v1 = func(f1)
result = v1() # 执行inner函数 / f1含函数 -> 123
print(result) # 666
```

```
def func():
    print(1)

v1 = func
func = 666
```

==========装饰器=========

```
def func(arg):
    def inner():
        print('before')
        v = arg()
        print('after')
        return v
    return inner

def index():
    print('123')
    return '666'


# 示例一
"""
v1 = index() # 执行index函数，打印123并返回666赋值给v1.
"""
# 示例二
"""
v2 = func(index) # v2是inner函数，arg=index函数
index = 666
v3 = v2()
"""
# 示例三
"""
v4 = func(index)
index = v4  # index ==> inner
index()
"""

# 示例四
index = func(index)
index()
```

```
def func(arg):
    def inner():
```

```
        v = arg()
        return v
    return inner

# 第一步：执行func函数并将下面的函数参数传递，相当于：func(index)
# 第二步：将func的返回值重新赋值给下面的函数名。 index = func(index)
@func
def index():
    print(123)
    return 666

print(index)
```

装饰器：在不改变原函数内部代码的基础上，在函数执行之前和之后自动执行某个功能。 应用：

```
# 计算函数执行时间

def wrapper(func):
    def inner():
        start_time = time.time()
        v = func()
        end_time = time.time()
        print(end_time-start_time)
        return v
    return inner

@wrapper
def func1():
    time.sleep(2)
    print(123)
@wrapper
def func2():
    time.sleep(1)
    print(123)

def func3():
    time.sleep(1.5)
    print(123)

func1()
```

总结

- 目的：在不改变原函数的基础上，再函数执行前后自定义功能。
- 编写装饰器 和应用

```
# 装饰器的编写
def x(func):
    def y():
        # 前
        ret = func()
        # 后
```

```
        return ret
    return y

# 装饰器的应用
@x
def index():
    return 10

@x
def manage():
    pass

# 执行函数，自动触发装饰器了
v = index()
print(v)
```

- 应用场景：想要为函数扩展功能时，可以选择用装饰器。
- 记住：
  - 装饰器编写格式

```
def 外层函数(参数):
    def 内层函数(*args,**kwargs):
        return 参数(*args,**kwargs)
    return 内层函数
```

  - 装饰器应用格式

```
@外层函数
def index():
    pass

index()
```

  - 问题：为什么要加 *args, **kwargs

3.推导式

- 列表推导式
  - 基本格式

```
"""
目的：方便的生成一个列表。
格式：
    v1 = [i for i in 可迭代对象 ]
    v2 = [i for i in 可迭代对象 if 条件 ] # 条件为true才进行append
"""
v1 = [ i for i in 'alex' ]
v2 = [i+100 for i in range(10)]
v3 = [99 if i>5 else 66  for i in range(10)]
```

```
def func():
    return 100
v4 = [func for i in range(10)]

v5 = [lambda : 100 for i in range(10)]
result = v5[9]()

def func():
    return i
v6 = [func for i in range(10)]
result = v6[5]()

v7 = [lambda :i for i in range(10)]
result = v7[5]()


v8 = [lambda x:x*i for i in range(10)] # 新浪微博面试题
# 1.请问 v8 是什么?
# 2.请问 v8[0](2) 的结果是什么?

# 面试题
def num():
    return [lambda x:i*x for i in range(4)]
# num() -> [函数,函数,函数,函数]
print([ m(2) for m in num() ]) # [6,6,6,6]

# ###################### 筛选 ##########################
v9 = [i for i in range(10) if i > 5]
```

- 集合推导式

```
v1 = { i for i in 'alex' }
```

- 字典的推导式

```
v1 = { 'k'+str(i):i for i in range(10) }
```

今日总结

- 装饰器 (6**)
  - 编写格式：双层嵌套函数
  - 应用格式：@外层函数
  - 理解：
    - 变量赋值

```
def func():
    print(1)

v1 = func
func = 666
```

        - 看看到底return的是什么?
        - 自己 > 上级作用域
    - 背会:

```
@xx  # index = xx(index)
def index():
    pass
index()
```

- 推导式 (3*)
- 模块

```
import time

v = time.time() # 获取当前时间
time.sleep(2) # 睡2秒
```