

# Day06 课堂笔记

---

## 课程之前

---

## 复习和反馈

形参 是属于局部变量的.

## 作业

## 今日内容

- 函数部分
  - 不定长参数的补充扩展
  - 匿名函数 `lambda`
- 面向对象

## 函数

---

# 不定长参数补充-函数调用时的拆包

```
def my_sum(*args, **kwargs):  
    num = 0 # 定义变量,保存求和的结果  
    for i in args:  
        num += i  
  
    for j in kwargs.values():  
        num += j  
  
    print(num)  
  
# 需求, my_list = [1, 2, 3, 4] 字典 my_dict = {'a':  
1, 'b': 2, 'c': 3, 'd': 4}  
my_list = [1, 2, 3, 4]  
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}  
  
# 将字典和列表中的数据使用 my_sum 函数进行求和, 改如何传  
参的问题  
# my_sum(1, 2, 3, 4)  
# my_sum(a=1, b=2, c=3, d=4)  
# 想要将列表(元组)中的数据 分别作为位置参数,进行传参,需要  
对列表进行拆包操作  
# my_sum(*my_list) # my_sum(1, 2, 3, 4)
```

```
# 想要将字典中的数据，作为关键字传参，，需要使用 **  
对字典进行拆包  
my_sum(**my_dict) # my_sum(a=1, b=2, c=3, d=4)
```

## 匿名函数

匿名函数：就是使用 `lambda` 关键字定义的函数  
一般称为使用 `def` 关键字定义的函数为，标准函数

匿名函数只能书写一行代码

匿名函数的返回值不需要 `return`，一行代码（表达式）的结果就是返回值

使用场景：作为函数的参数，这个函数比较简单，值使用一次，没有必要使用 `def` 定义

- 语法

**lambda** 参数：一行代码 # 这一行代码,称为是表达式

# 匿名函数一般不需要我们主动的调用，一般作为函数的参数使用的

# 我们在学习阶段为了查看匿名函数定义的是否正确,可以调用

# 1, 在定义的时候 ,将匿名函数的引用保存到一个变量中

变量 = **lambda** 参数：一行代码

# 2. 使用变量进行调用

变量()

```
7 # lambda : print('hello lambda') # 匿名函数的定义
8 func11 = lambda: print('hello lambda')
9 fun
10 # 2
11 # 3. 有参无返回值
12 # 4. 有参有返回值
13
14
```

PEP 8: do not assign a lambda expression, use a def

Reformat file More actions...

不要对 lambda 表达式赋值, 使用 def

## • 代码

# 1. 无参无返回值

```
def func1():
    print('hello world')
```

func1()

# lambda : print('hello lambda') # 匿名函数的定义

```
func11 = lambda: print('hello lambda')  
func11()
```

## # 2. 无参有返回值

```
def func2():  
    return 10
```

```
print(func2())  
func22 = lambda: 10  
print(func22())
```

## # 3. 有参无返回值

```
def my_sum(a, b):  
    print(a + b)
```

```
my_sum(1, 2)  
my_sum11 = lambda a, b: print(a+b)  
my_sum11(10, 20)
```

## # 4. 有参有返回值

```
def func4(a, b):  
    return a + b
```

```
print(func4(1, 2))    # num = func4(1, 2)
print(num)
func44 = lambda a, b: a+b

print(func44(10, 20))
```

## ● 练习

1. 定义一个匿名函数可以求两个数的乘积
2. 定义一个匿名函数，参数为字典，返回字典中键为 `age` 的值

# 1. 定义一个匿名函数可以求两个数的乘积（参数需要两个，）

```
func1 = lambda a, b: a * b
```

# 2. 定义一个匿名函数，参数为字典，返回字典中键为 `age` 的值

# 参数只是一个占位的作用，定义的时候没有具体的数据值，形参的值是在调用的时候进行传递，此时，形参才有数据值形参的类型就是由实参来决定的，在函数定义的时候，参数只是一个符号，写什么都可以，想让其是字典类型，只需要保证

# 实参是字典即可

```
func2 = lambda x: x.get('age')
```

```
func3 = lambda x: x['age']
```

```
print(func1(1, 2))
```

```
print(func1(3, 2))
```

```
my_dict = {'name': '张三', 'age': 18}
```

```
print(func2(my_dict))
```

```
print(func3(my_dict))
```

- 匿名函数作为函数的参数 - 列表中的字典排序

```
user_list = [  
{"name": "zhangsan", "age": 18}, {"name": "lisi",  
"age": 19}, {"name": "wangwu", "age": 17}  
]
```

列表排序(列表中的数字):

列表.sort() # 升序

列表.sort(reverse=True) # 降序

列表中的内容都是字典，想要排序？

```
user_list = [  
    {"name": "zhangsan", "age": 18},
```

```
{ "name": "lisi", "age": 19 },
{ "name": "wangwu", "age": 17 }
]

# user_list.sort()
# 列表的排序，默认是对列表中的数据进行比大小的，可以对 数
# 字类型和字符串进行比大小，
# 但是对于字典来说，就不知道该怎么比大小，此时，我们需要使用
# sort 函数中的 key 这个参数，来指定字典比大小的方法
# key 这个参数，需要传递一个函数，一般是匿名函数，字典的
# 排序，其实要指定根据字典的什么 键进行排序，我们只需要使用
# 匿名函数返回字典的这个键对应的值即可
# 列表.sort(key=lambda x: x['键'])
# 根据年龄排序
# user_list.sort(key=lambda x: x['age'])
user_list.sort(key=lambda x: x['age'], reverse=True)
print(user_list)

# user_list.sort(key=lambda x: x['age'])
# 说明：匿名函数中的参数是 列表中的数据，在 sort 函数内
# 部，会调用 key 这个函数(将列表中每个数据作为实参传递给形
# 参)，
# 从列表中的获取函数的返回值，对返回值进行比大小操作(<)

def get_value(x):
    return x['age']
```



```
user_list.sort(key=get_value)
```

- 字符串比大小

字符串比大小,是比较字符对应的 ASCII 码值

A < Z < a < z

ord(字符) # 获取字符对应的 ASCII 的值

chr(ASCII 值) # 获取对应的 字符

字符串比大小:

对应下标位置字符的大小,直到比出大小,如果全部比完了,还没有比出大小,就是相等

```
>>> 'a' < 'A'
False
>>> 'a' > 'A'
True
>>> 'a' > 'z'
True
>>> ord('a')
97
>>> ord('A')
65
>>> chr(97)
'a'
```

```
>>> 'abcde' < 'b'
True
>>> 'abc' < 'abbbbb'
False
```

## 面向对象

# 基本的介绍

面向对象是一个编程思想(写代码的套路)

编程思想:

1. 面向过程

2. 面向对象

以上两种都属于写代码的套路(方法),最终目的都是为了将代码书写出来,只不过过程和思考方法不太一样.

- 面向过程
  - 关注的是 具体步骤的实现,所有的功能都自己书写
  - 亲力亲为
  - 定义一个个函数,最终按照顺序调用函数
- 面向对象
  - 关注的是结果,谁(对象)能帮我做这件事
  - 偷懒
  - 找一个对象(),让对象去做

## 类和对象

面向对象的核心思想是 找一个对象去帮我们处理事情  
在程序代码中 对象是由 类 创建的

类和对象,是 面向对象编程思想中非常重要的两个概念

- 类
  - 抽象的概念, 对 多个 特征和行为相同或者相似事物的统称
  - 泛指(指代多个,而不是具体的一个)
- 对象
  - 具体存在的一个事物, 看得见摸得着的
  - 特指的,(指代一个)

苹果 ---> 类

红苹果 -----> 类

张三嘴里正在吃的那个苹果 ---> 对象

## 类的组成

- 1, 类名 (给这多个事物起一个名字, 在代码中 满足大驼峰命名法(每个单词的首字母大写))
- 2, 属性 (事物的特征, 即有什么, 一般文字中的名词)
- 3, 方法 (事物的行为, 即做什么事, 一般是动词)

# 类的抽象(类的设计)

类的抽象,其实就是找到 类的 类名, 属性 和方法

需求:

- 小明 今年 **18** 岁, 身高 **1.75**, 每天早上 跑 完步, 会去 吃 东西
- 小美 今年 **17** 岁, 身高 **1.65**, 小美不跑步, 小美喜欢 吃 东西

类名: 人类(Person, People)

属性: 名字(name), 年龄(age), 身高(height)

方法: 跑步(run) 吃(eat)

需求:

- 一只 黄颜色 的 狗狗 叫 大黄
- 看见生人 汪汪叫
- 看见家人 摇尾巴

类名: 狗类(Dog)

属性: 颜色(color) , 名字(name)

方法: 汪汪叫 (bark), 摇尾巴(shake)

# 面向代码的步骤

1. 定义类，在定义类之前先设计类
2. 创建对象，使用第一步定义的类创建对象
3. 通过对象调用方法

## 面向对象基本代码的书写

### 1. 定义类

先定义简单的类，不包含属性，在 `python` 中定义类需要使用关键字 `class`

方法：方法的本质是在类中定义的函数，只不过，第一个参数是 `self`

```
class 类名:
    # 在缩进中书写的内容,都是类中的代码
    def 方法名(self):    # 就是一个方法
        pass
```

## 2. 创建对象

创建对象是使用 `类名()` 进行创建,即

```
类名() # 创建一个对象, 这个对象在后续不能使用  
# 创建的对象想要在后续的代码中继续使用, 需要使用一个变量,  
将这个对象保存起来  
变量 = 类名() # 这个变量中保存的是对象的地址, 一般可以  
成为这个变量为对象  
  
# 一个类可以创建多个对象, 只要出现 类名() 就是创建一个对  
象,每个对象的地址是不一样的
```

## 3. 调用方法

对象.方法名()

列表.sort()

列表.append()

## 4. 案例实现

需求:小猫爱吃鱼, 小猫要喝水

类名: 猫类 `Cat`

属性: 暂无

方法: 吃鱼 (`eat`) 喝水 (`drink`)

```
"""
```

需求:小猫爱吃鱼, 小猫要喝水, 定义不带属性的类

```
"""
```

```
class Cat:
```

```
    # 在缩进中书写 方法
```

```
    def eat(self): # self 会自动出现, 暂不管
        print('小猫爱吃鱼...')
```

```
    def drink(self):
        print('小猫要喝水----')
```

```
# 2. 创建对象
```

```
blue_cat = Cat()
```

```
# 3. 通过对象调用类中的方法
```

```
blue_cat.eat()
```

```
blue_cat.drink()
```

```
# 创建对象
```

```
black_cat = Cat()
```

```
black_cat.eat()
```

```
black_cat.drink()
```

```
Cat() # 是
```



```
a = black_cat # 不是
b = Cat      # 不是
```

## self 的说明

```
class Cat:
    # 在缩进中书写 方法
    def eat(self): # self 会自动出现,暂不管
        print('小猫爱吃鱼...')
```

```
black_cat.eat()
```

1. 从函数的语法上讲, `self` 是形参, 就可以是任意的变量名, 只不过我们习惯性将这个形参写作 `self`
2. `self` 是普通的形参, 但是在调用的时候没有传递实参值, 原因是, `Python` 解释器在执行代码的时候, 自动的将调用这个方法的对象 传递给了 `self`, 即 `self` 的本质是对象
3. 验证, 只需要确定 通过哪个对象调用, 对象的引用和 `self` 的引用是一样的
4. `self` 是函数中的局部变量, 直接创建的对象是全局变量

"""

需求:小猫爱吃鱼, 小猫要喝水, 定义不带属性的类

"""

```
class Cat:
```

```
    # 在缩进中书写 方法
```

```
    def eat(self): # self 会自动出现,暂不管
```

```
        print(f'{id(self)}, self')
```

```
        print('小猫爱吃鱼...')
```

```
# 2. 创建对象
```

```
blue_cat = Cat()
```

```
print(f'{id(blue_cat)}, blue_cat')
```

```
# 3. 通过对象调用类中的方法
```

```
blue_cat.eat() # blue_cat 对象调用 eat 方法,解释器就  
会将 blue_cat 对象传给 self
```

```
print('_*_ ' * 30)
```

```
# 创建对象
```

```
black_cat = Cat()
```

```
print(f"{id(black_cat)}, black_cat")
```

```
black_cat.eat() # black_cat 对象调用 eat 方法,解释器  
会将 black_cat 对象传给 self
```

# 对象的属性操作

## 添加属性

```
对象.属性名 = 属性值
```

- 类内部添加

```
在内部方法中, self 是对象,  
self.属性名 = 属性值  
# 在类中添加属性一般写在 __init__ 方法中
```

- 类外部添加

```
对象.属性名 = 属性值 # 一般不使用
```

## 获取属性

```
对象.属性名
```

- 类内部

```
在内部方法中, self 是对象,  
self.属性名
```

- 类外部

```
对象.属性名 # 一般很少使用
```

```
class Cat:
    # 在缩进中书写 方法
    def eat(self): # self 会自动出现,暂不管
        print(f'{id(self)}, self')
        print(f'小猫{self.name} 爱吃鱼...')
```

## # 2. 创建对象

```
blue_cat = Cat()
print(f'{id(blue_cat)}, blue_cat')
```

# 给 蓝猫添加 name 属性

```
blue_cat.name = '蓝猫'
```

## # 3. 通过对象调用类中的方法

blue\_cat.eat() # blue\_cat 对象调用 eat 方法,解释器就会将 blue\_cat 对象传给 self

```
print('_*_ ' * 30)
```

## # # 创建对象

```
black_cat = Cat()
```

```
black_cat.name = '黑猫'
```

```
print(f"{id(black_cat)}, black_cat")
```

black\_cat.eat() # black\_cat 对象调用 eat 方法,解释器就会将 black\_cat 对象传给 self

# 魔法方法

python 中有一类方法，以两个下划线开头，两个下划线结尾，并且在满足某个条件的情况下，会自动调用，这类方法称为 魔法方法

学习：

1. 什么情况下自动调用
2. 有什么用，用在哪
3. 书写的注意事项

## `__init__` 方法 \*\*

1. 什么情况下自动调用
  - > 创建对象之后会自动调用
2. 有什么用，用在哪
  - > 1. 给对象添加属性的，（初始化方法，构造方法） 2. 某些代码，在每次创建对象之后，都要执行，就可以将这行代码写在 `__init__` 方法
3. 书写的注意事项
  - > 1. 不要写错了 2. 如果 `init` 方法中，存在出了 `self` 之外的参数，在创建对象的时候必须传参

```
"""
猫类，属性 name, age , show_info(输出属性信息)
"""
```

```
class Cat:
    # 定义添加属性的方法
    def __init__(self, name, age): # 这个方法是创建
    对象之后调用
        self.name = name # 给对象添加 name 属性
        self.age = age # 给对象添加 age 属性

    # 输出属性信息
    def show_info(self):
        print(f'小猫的名字是: {self.name}, 年龄是:
        {self.age}')
```

# 创建对象,不要在自己类缩进中创建

# Cat() # 创建对象 ,会输出

blue\_cat = Cat('蓝猫', 2)

blue = blue\_cat

blue.show\_info()

# 创建黑猫

black\_cat = Cat('黑猫', 3)

black\_cat.show\_info()

## \_\_str\_\_ 方法 \*

### 1. 什么情况下自动调用

- 使用 `print(对象)` 打印对象的时候 会自动调用

### 2. 有什么用，用在哪

- 在这个方法中一般书写对象的 属性信息的，即打印对象的时候想要查看什么信息，在这个方法中进行定义的
- 如果类中没有定义 `__str__` 方法，`print(对象)`，默认输出对象的引用地址

### 3. 书写的注意事项

- 这个方法必须返回 一个字符串

```
class Cat:
    # 定义添加属性的方法
    def __init__(self, n, age): # 这个方法是创建对象之后调用
        self.name = n # 给对象添加 name 属性
        self.age = age # 给对象添加 age 属性

    def __str__(self):
        # 方法必须返回一个字符串，只要是字符串就行，
        return f'小猫的名字是：{self.name}，年龄是：{self.age}'
```

# 创建对象,不要在自己类缩进中创建

# Cat() # 创建对象 ,会输出

```
blue_cat = Cat('蓝猫', 2)
```

```
print(blue_cat)
```

# 创建黑猫

```
black_cat = Cat('黑猫', 3)
```

```
print(black_cat)
```

## `__del__` 方法 [了解]

`__init__` 方法, 创建对象之后, 会自动调用 (构造方法)

`__del__` 方法, 对象被删除销毁时, 自动调用的(遗言, 处理后事) (析构方法)

1. 调用场景, 程序代码运行结束, 所有对象都被销毁
2. 调用场景, 直接使用 `del` 删除对象(如果对象有多个名字(多个对象引用一个对象), 需要吧所有的对象都删除才行 )

```
class Demo:
```

```
    def __init__(self, name):
```

```
        print('我是 __init__, 我被调用了 ')
```

```
        self.name = name
```



```
def __del__(self):  
    print(f'{self.name} 没了，给他处理后事...')  
  
# Demo('a')  
  
a = Demo('a')  
  
b = Demo('b')  
del a # 删除销毁 对象,  
print('代码运行结束')
```

## 案例1

需求：

1. 小明 体重 75.0 公斤
2. 小明每次 跑步 会减肥 0.5 公斤
3. 小明每次 吃东西 体重增加 1 公斤

类名：人类 Person

属性：姓名 name，体重 weight

方法：跑步 run

吃东西 eat

添加属性 `__init__`

属性信息 `__str__`

```
class Person:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight

    def __str__(self):
        return f"姓名：{self.name}，体重：{self.weight} kg"

    def run(self):
        print(f'{self.name} 跑步 5 km，体重减少了')
        # 减体重，即修改属性
        self.weight -= 0.5

    def eat(self):
        print(f'{self.name} 大餐一顿，体重增加了')
        # 修改体重
        self.weight += 1
```

```
xm = Person('小明', 75.0)
print(xm)
xm.run()
print(xm)
xm.eat()
print(xm)
```

## 案例 2

需求：

1. 房子(**House**) 有 户型、总面积 和 家具名称列表
  - 新房子没有任何的家具
2. 家具(**HouseItem**) 有 名字 和 占地面积，其中
  - 席梦思(**bed**) 占地 4 平米
  - 衣柜(**chest**) 占地 2 平米
  - 餐桌(**table**) 占地 1.5 平米
3. 将以上三件 家具 添加 到 房子 中
4. 打印房子时，要求输出：户型、总面积、剩余面积、家具名称列表

剩余面积

1. 在创建房子对象时，定义一个 剩余面积的属性，初始值和总面积相等
2. 当调用 `add_item` 方法，向房间 添加家具 时，让 剩余面积 -= 家具面积

类名：房子类 House

属性：户型 name，总面积 total\_area，剩余面积 free\_area  
= total\_area

家具名称列表 item\_list = []

方法：\_\_init\_\_，\_\_str\_\_

添加家具方法

```
def add_item(self, item): #item 家具对象
```

先判断房子的剩余面积和总面积的关系

修改房子的剩余面积

修改房子的家具名称列表

类名：家具类 HouseItem

属性：名字 name，占地面积 area

方法：\_\_init\_\_，\_\_str\_\_