实例成员

实例变量

1. 语法
   (1) 定义：对象.变量名
   (2) 调用：对象.变量名

2. 说明
   (1) 首次通过对象赋值为创建，再次赋值为修改.
   w01 = Wife()
   w01.name = “丽丽”
   w01.name = “莉莉”
   (2) 通常在构造函数(_init_)中创建。
   w01 = Wife(“丽丽”,24)
   print(w01.name)
   (3) 每个对象存储一份，通过对象地址访问。

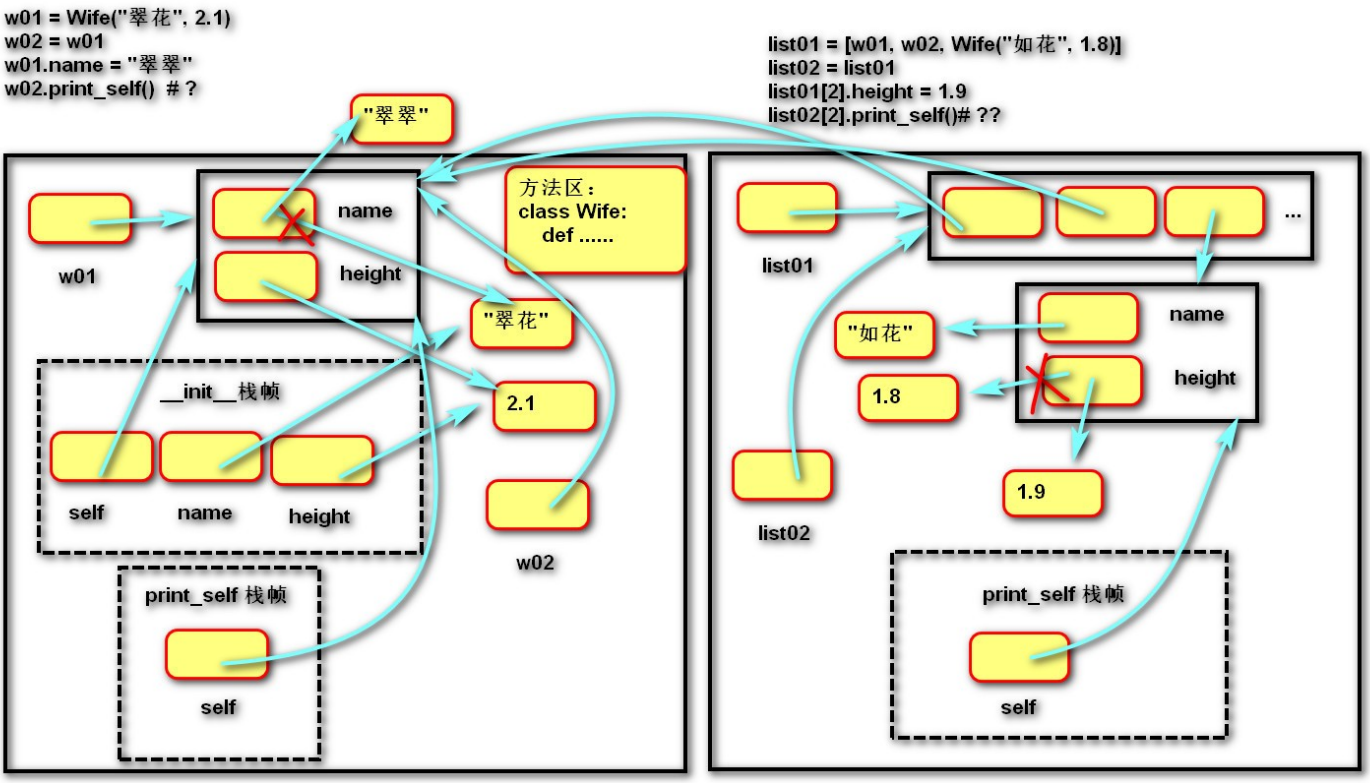
3. 作用：描述所有对象的共有数据。
4. __dict__：对象的属性，用于存储自身实例变量的字典。

实例方法

1. 语法
   (1) 定义： def 方法名称(self, 参数列表):
   方法体
   (2) 调用： 对象地址.实例方法名(参数列表)
   不建议通过类名访问实例方法
2. 说明
   (1) 至少有一个形参，第一个参数绑定调用这个方法的对象,一般命名为"self"。
   (2) 无论创建多少对象，方法只有一份，并且被所有对象共享。
3. 作用：表示对象行为。

```
"""
    实例成员
        记住一句话:实例成员,使用对象地址访问.
    练习:exercise01.py
    练习:exercise02.py
"""
class Wife:
    pass
w01 = Wife()
```

```python
# 定义实例变量: 对象.变量名 = ?
w01.name = "赵敏"
print(w01.name)
w02 = Wife()
# print(w02.name)# 错误.因为w02指向的对象,没有创建过实例变量name
print(w01.__dict__)# 通过__dict__获取当前对象的所有实例变量
print(w02.__dict__)
class Wife2:
    def __init__(self, name, height):
        self.name = name
        self.height = height
    # 实例方法
    def print_self(self):
        print(self.name,self.height)
w01 = Wife2("灭绝",2.3)
w02 = Wife2("金毛狮王",3.3)
# 建议:实例方法,通过对象地址访问.
w01.print_self()
# 不建议:Wife2.print_self()# 没有传递对象地址,实例方法不能正确访问对象数据.
Wife2.print_self(w02)
```



```
    画出内存图
"""
class Wife:
    def __init__(self, name, height):
        self.name = name
        self.height = height
    def print_self(self):
        print(self.name, self.height)
# 构造函数init:将数据传递给创建的对象.
w01 = Wife("翠花", 2.1)  # 调用了构造函数init
# 将老婆对象地址赋值给w02(两个变量指向同一个老婆对象)
w02 = w01
# 通过其中一个变量修改老婆姓名
w01.name = "翠翠"
# 通过另外一个变量访问老婆信息
w02.print_self()  # "翠翠"  2.1
list01 = [w01, w02, Wife("如花", 1.8)]
list02 = list01
list01[2].height = 1.9
list02[2].print_self()# ??
```

4.

类成员

类变量

1. 语法
   (1) 定义：在类中，方法外定义变量。
   class 类名:
   变量名 = 表达式
   (2) 调用：类名.变量名
   不建议通过对象访问类变量
2. 说明
   (1) 存储在类中。
   (2) 只有一份，被所有对象共享。
3. 作用：描述所有对象的共有数据。

"""

```
    练习
"""
class Student:
    def __init__(self,name,sex,score,age):
        self.name = name
        self.sex = sex
        self.score = score
        self.age = age
    def print_self(self):
        print(self.name,self.sex,self.score,self.age)
list01 = [
    Student("无忌","男",86,28),
    Student("赵敏","女",99,26),
    Student("周芷若","女",72,24),
    Student("珠儿","女",90,23),
]
# 练习1:定义在list01中查找"赵敏"同学的函数
def find_student():
    for item in list01:
            if item.name == "赵敏":
                return item
stu1 = find_student()
# 1. 如果函数没有找到,则错误,所以如果不能确定是否找到,需要判断.
if stu1:
    print(stu1.name)
# 练习2:定义在list01中查找所有女同学的函数
def find_girl():
    list_stu2 = []
    for item in list01:
        if item.sex == "女":
            list_stu2.append(item)
    return list_stu2
list_result = find_girl()
for item in list_result:
    item.print_self()
# 练习3:定义在list01中查找所有同学姓名的函数
def find_name():
    list_name = []
    for i in list01:
        list_name.append(i.name)
    return  list_name
```

```
for item in find_name():
    print("%s"%item,end=" ")
print()
#4.定义将 list01 中所有女同学的成绩+10 分的函数
def girl_10():
    for item in list01:
        if item.sex == "女":
            item.score += 10
girl_10()
for item in list01:
    item.print_self()
print()
# 练习 5:定义函数,删除 list01 中所有不及格的学生
def delete_fail():
    for item in range(len(list01) - 1, -1, -1):
        if list01[item].score < 90:
            del list01[item]
delete_fail()
for item in list01:
    item.print_self()
```

类方法

1. 语法
   (1) 定义:
   @classmethod
   def 方法名称(cls,参数列表):
   方法体
   (2) 调用：类名.方法名(参数列表)
   不建议通过对象访问类方法
2. 说明
   (1) 至少有一个形参，第一个形参用于绑定类，一般命名为'cls'
   (2) 使用@classmethod 修饰的目的是调用类方法时可以隐式传递类。
   (3) 类方法中不能访问实例成员，实例方法中可以访问类成员。
3. 作用：操作类变量。

"""

```
类成员
15:25
```

```
"""
class ICBC:
    # 类变量:总行的钱
    total_money = 1000000
    # 类方法
    @classmethod
    def print_total_money(cls):
        print(id(cls), id(ICBC))
        # cls : 存储当前类的地址
        # print("当前总行金额:",ICBC.total_money)
        print("当前总行金额:", cls.total_money)
    def __init__(self, name, money):
        self.name = name
        self.money = money
        # 从总行扣除当前支行的钱
        ICBC.total_money -= money
i01 = ICBC("天坛支行", 100000)
i02 = ICBC("陶然亭支行", 100000)
# 主流:通过类访问类成员
ICBC.print_total_money()
print(ICBC.total_money)
# 非主流:通过对象访问类成员
# print(i02.total_money)
# i02.print_total_money()
```

4.

静态方法

1. 语法
   (1) 定义：
   @staticmethod
   def 方法名称(参数列表):
   方法体
   (2) 调用：类名.方法名(参数列表)
   不建议通过对象访问静态方法
2. 说明
   (1) 使用@ staticmethod 修饰的目的是该方法不需要隐式传参数。
   (2) 静态方法不能访问实例成员和类成员
3. 作用：定义常用的工具函数。

三大特征

封装

数据角度讲

1. 定义：
   将一些基本数据类型复合成一个自定义类型。
2. 优势：
   将数据与对数据的操作相关联。
   代码可读性更高（类是对象的模板）。

行为角度讲

1. 定义：
   类外提供必要的功能，隐藏实现的细节。
2. 优势：
   简化编程，使用者不必了解具体的实现细节，只需要调用对外提供的功能。
3. 私有成员：
   (1) 作用：无需向类外提供的成员，可以通过私有化进行屏蔽。
   (2) 做法：命名使用双下划线开头。
   (3) 本质：障眼法，实际也可以访问。
   私有成员的名称被修改为：_类名__成员名，可以通过_dict_属性或 dir 函数查看。

```
"""
    封装
        数据: 老婆(名字,高度,体重)  学生(姓名,成绩,性别,年龄)
            字典(名称,单价...)
        行为: 必要
"""
class Wife:
    def __init__(self, name, age):
        self.name = name
        # 私有成员:以双下划线开头
        # self.__age = age
        self.set_age(age)
    def get_age(self):
        return self.__age
    def set_age(self, value):
```

```
        if 20 <= value <= 50:
            self.__age = value
        else:
            raise ValueError("我不要")
w01 = Wife("小乔", 25)
# 不能访问私有变量
# print(w01.__age)
# w01.set_age(30)
print(w01.get_age())
```

```
"""
    封装
        行为: property
        练习:exercise05.py
"""
class Wife:
    def __init__(self, name, age):
        self.name = name
        # self.set_age(age)
        self.age = age
    def get_age(self):
        return self.__age
    def set_age(self, value):
        if 20 <= value <= 50:
            self.__age = value
        else:
            raise ValueError("我不要")
    # 类变量
    # property 属性: 在拦截对age的读写操作
    age = property(get_age,set_age)
w01 = Wife("小乔", 25)
# print(w01.get_age())
print(w01.age)
```

4. 属性@property：

公开的实例变量，缺少逻辑验证。私有的实例变量与两个公开的方法相结合，又使调用者的操作略显复杂。而属性可以将两个方法的使用方式像操作变量一样方便。

(1) 定义：

```
@property
def 属性名(self):
    return self.__属性名
@属性名.setter
```

def 属性名(self, value):
self.__属性名= value

(2) 调用:

对象.属性名 = 数据

变量 = 对象.属性名

(3) 说明:

通常两个公开的属性，保护一个私有的变量。

@property 负责读取，@属性名.setter 负责写入

只写：属性名= property(None, 写入方法名)

"""

```
    属性拦截的正确写法
"""
class Enemy:
    def __init__(self, name, hp, attack):
        self.name = name
        self.hp = hp
        self.attack = attack
    @property # 拦截读取,hp = property(读取方法,None)
    def hp(self):
        return self.__hp
    @hp.setter  # 拦截写入,hp.setter(写入方法)
    def hp(self, value):
        if 0 <= value <= 100:
            self.__hp = value
        else:
            raise ValueError("血量不在范围内.")
    #hp = property(get_hp, set_hp)
    @property # 拦截读取,attack = property(读取方法,None)
    def attack(self):
        return self.__attack
    @attack.setter  # 拦截写入,attack.setter(写入方法)
    def attack(self, value):
        if 10 <= value <= 50:
            self.__attack = value
        else:
            raise ValueError("攻击力不在范围内.")
    #attack = property(get_attack, set_attack)
enemy01 = Enemy("吕布", 100, 50)
print(enemy01.hp)
print(enemy01.attack)
```

```
"""
   封装
      行为: 标准属性
      练习:exercise06.py
      练习:exercise07.py
"""
# 读写属性age
class Wife:
   def __init__(self, name, age):
      self.name = name
      self.age = age
   @property# 拦截读取 age = property(读取方法,None)
   def age(self):
      return self.__age
   @age.setter # 拦截写入 age.setter(写入方法)
   def age(self, value):
      if 20 <= value <= 50:
         self.__age = value
      else:
         raise ValueError("我不要")
w01 = Wife("小乔", 25)
# print(w01.get_age())
print(w01.age)
# ----------------------------
# 只读属性age
class Wife2:
   def __init__(self, name):
      self.name = name
      self.__age = 23
   @property# 拦截读取 age = property(读取方法,None)
   def age(self):
      return self.__age
w02 = Wife2("大桥")
# w02.age = 30
#---------------------
# 只写属性age
class Wife3:
   def __init__(self, name):
      self.name = name
      self.age = 23
   def set_age(self, value):
```

```
        if 20 <= value <= 50:
            self.__age = value
        else:
            raise ValueError("我不要")
    age = property(None,set_age)
w03 = Wife3("大桥")
w03.age = 30
# print(w03.age)# 不能读取
```