2-2面向对象

2020-知数堂-Python第二期

章老师

目录

- 类的声明与使用
- 类的私有方法和私有属性
- self是什么
- 类的特殊方法
  - __init__ vs __new__
  - __str__ vs __repr__
  - __setitem__ vs __getitem__
  - __iter__ vs __next__
  - __call__
  - 运算符重载
  - __getattribute__ vs __getattr__ vs __get__ vs __set__

简单声明类与使用

```
class MyClass:
    """一个简单的类实例"""
    i = 12345
    def f(self):
        return 'hello world'

# 实例化类
x = MyClass()
Y = MyClass()

# 访问类的属性和方法
print("MyClass 类的属性 i 为: ", x.i)
print("MyClass 类的方法 f 输出为: ", x.f())
```

类的私有方法和私有属性

- 通过在属性或者方法前面加上两个下划线 __，就可以声明私有属性以及私有方法
- 私有属性和私有方法，在类的外部是不能够被访问的

```
class MyClass:
    """一个简单的类实例"""
    __i = 12345

    def __f(self):
        return 'hello world’

# 实例化类
x = MyClass()

# 访问类的属性和方法
print(“MyClass 类的属性 i 为: ”, x.__i)
#AttributeError: 'MyClass' object has no attribute '__i'
print("MyClass 类的方法 f 输出为: ", x.__f())
```

self是什么

- self代表类的实例，而非类。类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的第一个参数名称，按照惯例它的名称是 self。

```
class Rectangle:
    def __init__(s, width, length):
print("Rectangle innstance in init", id(s))

    def area(self):
        return self.length * self.width

r = Rectangle(13.0, 3.0)
print("Rectangle instance out of init", id(r))
print(r.area())
```

观察这两行的输出，发现对象的内存地址是一样的

类的特殊方法

- __init__ vs __new__
- __str__ vs __repr__
- __setitem__ vs __getitem__
- __iter__ vs __next__
- __call__
- 运算符重载

__init__

- 当一个对象被初始化时，就会调用对象的__init__方法

```
class ZstClass:
    def __init__(self):
        print("zst class init")

# 实例化类
x = ZstClass()
>> zst class init
```

__init__

类似于下面的这种调用尚未赋值的成员变量是合法的，但是确会给人带来烦恼

```
class Rectangle:
        def area(self):
                return self.length * self.width

r = Rectangle()
r.length, r.width = 13, 3
r.area()
```

__init__

我们可以在init方法初始化成员变量

```
class Rectangle:
    def __init__(self, width, length):
        if not isinstance(width, float):
            raise Exception("invalid width")
        if not isinstance(length, float):
            raise Exception("invalid length")
        self.width = width
        self.length = length

    def area(self):
        return self.length * self.width

r = Rectangle(13.0, 3.0)
print(r.area())
```

__new__

我们希望声明一个拥有单位的float类

```
class Float_Fail(float):
    def __init__(self, value, unit):
        super().__init__(value)
        self.unit = unit

f = Float_Fail(1, "kg")

# 报错
#f = Float_Fail(1, "kg")
#TypeError: float() takes at most 1 argument (2 given)
```

__new__

- 对于内置的float类，我们不能够简单的重载init方法，因为float类是不可修改
- 对于这种情况，我们要使用__new__.
- __new__方法中允许创建未初始化的对象，允许我们在init方法前设置对象的属性

__new__

```
class Float_Units(float):
    def __new__(cls, value, unit):
        obj = super().__new__(cls, value)
        obj.unit = unit
        return obj

f = Float_Units(1, "kg")
```

你能总结一下__init__和__new__之间的区别么