day19

今日内容

- 面向对象基本用法
- 好处和应用场景
- 面向对象的三大特性

内容详细

1.面向对象基本格式

```
# ###### 定义类 ######
class 类名:
    def 方法名(self,name):
        print(name)
        return 123
    def 方法名(self,name):
        print(name)
        return 123
    def 方法名(self,name):
        print(name)
        return 123
# ###### 调用类中的方法 ######
# 1.创建该类的对象
obj = 类名()
# 2.通过对象调用方法
result = obj.方法名('alex')
print(result)
```

应用场景：遇到很多函数，需要给函数进行归类和划分。 【封装】

练习题：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

class Db:
    def db_read(self):
        pass

    def db_write(self):
        pass

    def db_delete(self):
        pass
```

```
    def db_update(self):
        pass

class File:
    def file_read(self):
        pass

    def file_write(self):
        pass

    def file_delete(self):
        pass

    def file_update(self):
        pass

class Redis:
    def redis_read(self):
        pass

    def redis_write(self):
        pass

    def redis_delete(self):
        pass

    def redis_update(self):
        pass
```

2.对象的作用

存储一些值，以后方便自己使用。

```
class File:
    def read(self):
        with open(self.xxxxx, mode='r', encoding='utf-8') as f:
            data = f.read()
        return data

    def write(self, content):
        with open(self.xxxxx, mode='a', encoding='utf-8') as f:
            f.write(content)

# # 实例化了一个File类的对象
obj1 = File()
# # 在对象中写了一个xxxxx = 'test.log'
obj1.xxxxx = "test.log"
# # 通过对象调用类中的read方法，read方法中的self就是obj。
# # obj1.read()
obj1.write('alex')


# 实例化了一个File类的对象
```

```
obj2 = File()
# 在对象中写了一个xxxxx = 'test.log'
obj2.xxxxx = "info.txt"
# 通过对象调用类中的read方法，read方法中的self就是obj。
# obj2.read()
obj2.write('alex')
```

```
class Person:
    def show(self):
        temp = "我是%s,年龄：%s,性别：%s " %(self.name,self.age,self.gender,)
        print(temp)


p1 = Person()
p1.name = '李邵奇'
p1.age = 19
p1.gender = '男'
p1.show()

p2 = Person()
p2.name = '利奇航'
p2.age = 19
p2.gender = '男'
p2.show()
```

```
class Person:
    def __init__(self,n,a,g): # 初始化方法（构造方法），给对象的内部做初始化。
        self.name = n
        self.age = a
        self.gender = g

    def show(self):
        temp = "我是%s,年龄：%s,性别：%s " % (self.name, self.age, self.gender,)
        print(temp)

# 类() 实例化对象，自动执行此类中的 __init__方法。
p1 = Person('李兆琪',19,'男')
p1.show()

p2 = Person('利奇航',19,'男')
p2.show()
```

总结：将数据封装到对象，方便使用。

总结

```
"""
如果写代码时，函数比较多比较乱。
1. 可以将函数归类并放到同一个类中。
2. 函数如果有一个反复使用的公共值，则可以放到对象中。
```

```
"""

class File:
    def __init__(self,path):
        self.file_path = path

    def read(self):
        print(self.file_path)

    def write(self,content):
        print(self.file_path)

    def delete(self):
        print(self.file_path)

    def update(self):
        print(self.file_path)

p1 = File('log.txt')
p1.read()

p2 = File('xxxxxx.txt')
p2.read()
```

```
# 1. 循环让用户输入：用户名/密码/邮箱。 输入完成后再进行数据打印。
# ########## 以前的写法
USER_LIST = []
while True:
    user = input('请输入用户名：')
    pwd = input('请输入密码：')
    email = input('请输入邮箱：')
    temp = {'username':user,'password':pwd,'email':email}
    USER_LIST.append(temp)
for item in USER_LIST:
    temp = "我的名字：%s,密码：%s,邮箱%s" %(item['username'],item['password'],item['email'],)
    print(temp)

# ########## 面向对象写法

class Person:
    def __init__(self,user,pwd,email):
        self.username = user
        self.password = pwd
        self.email = email

USER_LIST = [对象(用户/密码/邮箱),对象(用户/密码/邮箱),对象(用户/密码/邮箱)]
while True:
    user = input('请输入用户名：')
    pwd = input('请输入密码：')
    email = input('请输入邮箱：')
    p = Person(user,pwd,email)
    USER_LIST.append(p)
```

```
for item in USER_LIST:
    temp = "我的名字：%s,密码：%s,邮箱%s" %(item.username,item.password,item.email,)
    print(temp)

# ########## 面向对象写法

class Person:
    def __init__(self,user,pwd,email):
        self.username = user
        self.password = pwd
        self.email = email

    def info(self):
        return "我的名字：%s,密码：%s,邮箱%s" %(item.username,item.password,item.email,)

USER_LIST = [对象(用户/密码/邮箱),对象(用户/密码/邮箱),对象(用户/密码/邮箱)]
while True:
    user = input('请输入用户名：')
    pwd = input('请输入密码：')
    email = input('请输入邮箱：')
    p = Person(user,pwd,email)
    USER_LIST.append(p)

for item in USER_LIST:
    msg = item.info()
    print(msg)
```

3.游戏开发

```
class Police:
    def __init__(self,name)
        self.name = name
         self.hp = 10000

    def tax(self):
        msg = "%s收了个税。" %(self.name,)
        print(msg)

    def fight(self):
        msg = "%s去战了个斗。" %(self.name,)

lsq = Police('李邵奇')
zzh = Police('渣渣会')
tyg = Police('堂有光')


class Bandit:
    def __init__(self,nickname)
        self.nickname = nickname
        self.hp = 1000

    def murder(self,name):
```

```
        msg = "%s去谋杀了%s" %(self.nickname, name,)


lcj = Bandit('二蛋')
lp = Bandit('二狗')
zsd = Bandit('狗蛋')

# 1. 二狗去谋杀渣渣会，二狗生命值-100；渣渣会生命值减5000
lp.murder(zzh.name)
lp.hp = lp.hp - 100
zzh.hp = zzh.hp - 5000
# ...
```

```
class Police:
    def __init__(self,name)
        self.name = name
         self.hp = 10000

    def dao(self,other):
        msg = "%s个了%s一刀。" %(self.name,other.nickname)
        self.hp = self.hp - 10
        other.hp = other.hp - 50
        print(msg)

    def qiang(self):
        msg = "%s去战了个斗。" %(self.name,)

    def quan(self,other):
        msg = "%s个了%s一全。" %(self.name,other.nickname)
        self.hp = self.hp - 2
        other.hp = other.hp - 10
        print(msg)


class Bandit:
    def __init__(self,nickname)
        self.nickname = nickname
        self.hp = 1000

    def qiang(self,other):
        msg = "%s个了%s一全。" %(self.nickname,other.name)
        self.hp -= 20
        other.hp -= 500


lcj = Bandit('二蛋')


lsq = Police('李邵奇')
lsq.dao(lcj)
lsq.quan(lcj)
lcj.qiang(lsq)
```

4.继承

```
# 父类(基类)
class Base:
    def f1(self):
        pass
# 子类（派生类）
class Foo(Base):
    def f2(self):
        pass

# 创建了一个字类的对象
obj = Foo()
# 执行对象.方法时，优先在自己的类中找，如果没有就是父类中找。
obj.f2()
obj.f1()

# 创建了一个父类的对象
obj = Base()
obj.f1()
```

问题：什么时候才能用到继承？多个类中如果有公共的方法，可以放到基类中避免重复编写。

```
class Base:
    def f1(self):
        pass

class Foo(Base):
    def f2(self):
        pass

class Bar(Base):
    def f3(self):
        pass

obj1 = Foo()

obj2 = Bar()
```

继承关系中的查找方法的顺序：

```
# 示例一
class Base:
    def f1(self):
        print('base.f1')

class Foo(Base):
    def f2(self):
        print('foo.f2')
```

```python
obj = Foo()
obj.f1()
obj.f2()

# 示例二
class Base:
    def f1(self):
        print('base.f1')

class Foo(Base):
    def f2(self):
        self.f1()
        print('foo.f2')

obj = Foo()
obj.f2()

# 示例三
class Base:
    def f1(self):
        print('base.f1')

class Foo(Base):
    def f2(self):
        self.f1()
        print('foo.f2')
    def f1(self):
        print('foo.f1')

obj = Foo()
obj.f2()

# 示例四
class Base:
    def f1(self):
        self.f2()
        print('base.f1')
    def f2(self):
        print('base.f2')
class Foo(Base):
    def f2(self):
        print('foo.f2')

obj = Foo()
obj.f1()

# 示例五
class TCPServer:
    pass
class ThreadingMixIn:
    pass
class ThreadingTCPServer(ThreadingMixIn, TCPServer):
    pass
```

```python
# 示例六
class BaseServer:
    def serve_forever(self, poll_interval=0.5):
        self._handle_request_noblock()
    def _handle_request_noblock(self):
        self.process_request(request, client_address)

    def process_request(self, request, client_address):
        pass

class TCPServer(BaseServer):
    pass

class ThreadingMixIn:
    def process_request(self, request, client_address):
        pass

class ThreadingTCPServer(ThreadingMixIn, TCPServer):
    pass

obj = ThreadingTCPServer()
obj.serve_forever()
```

注意事项：

- self 到底是谁？
- self 是哪个类创建的，就从此类开始找，自己没有就找父类。

5.多态（多种形态/多种类型）鸭子模型

```python
# Python
def func(arg):
    v = arg[-1] # arg.append(9)
    print(v)

# java
def func(str arg):
    v = arg[-1]
    print(v)
```

面试题：什么是鸭子模型。

对于一个函数而言，Python对于参数的类型不会限制，那么传入参数时就可以是各种类型，在函数中如果有例如：
arg.send方法，那么就是对于传入类型的一个限制（类型必须有send方法）。
这就是鸭子模型，类似于上述的函数我们认为只要能呱呱叫的就是鸭子（只有有send方法，就是我们要想的类型）

总结

1. 面向对象的三大特性：封装/继承/多态
    - 封装

```
class File:
    def read(self):
        pass
    def write(self):
        pass
```

```
class Person:
    def __init__(sef,name,age):
        self.name = name
        self.age = age
p = Person('alex',19)
```

    - 继承

```
class Base:
    pass
class Foo(Base):
    pass
```

        - 多继承
        - self到底是谁?
        - self是由于那个类创建，则找方法时候就从他开始找。
    - 多态

```
def func(arg): # 多种类型，很多事物
    arg.send() # 必须具有send方法，呱呱叫
```

2. 格式和关键词

```
class 类:
    def __init__(self,x):
        self.x = x

    def 方法(self,name):
        print(self.x, name)

# 实例化一个类的对象
v1 = 类(666)
v2.方法('alex')
```

三个词：

- 类
- 对象

- 方法

3. 什么时候用面向对象?
    - 函数（业务功能）比较多，可以使用面向对象来进行归类。
    - 想要做数据封装（创建字典存储数据时，面向对象）。
    - 游戏示例：创建一些角色并且根据角色需要再创建人物。