day21

今日内容

- 嵌套
- 特殊方法：`__init__`
- type/isinstance/issubclass/super
- 异常处理

内容回顾

```
def login():
    pass

login()
```

```
class Account:

    def login(self):
        pass

obj = Acount()
obj.login()
```

1. 谈谈你了解的面向对象？
2. 类和对象是什么关系？对象是类的一个实例。

    ```
    class Foo:
        def __init__(self,name):
            self.name = name
        def run(self):
            pass


    obj1 = Foo('ale')
    obj2 = Foo('eric')
    ```

3. self是什么？

```python
# self就是一个形式参数，对象调用方法时，python内部会将该对象传给这个参数。
class Foo:
    def run(self,num):
        pass

obj = Foo()
obj.run(5)
```

4. 类成员 & 对象成员 以及他们之间的关系。

```python
class Foo:
    name = 'alex'
    def run(self):
        pass

obj = Foo()
```

5. 类/方法/对象 都可以当作变量或嵌套到其他类型中。

```python
class Foo:
    def run(self):
        pass


v = [Foo,Foo]
v = [Foo(),Foo()]
obj = Foo()
v = [obj.run,obj.run,obj.run]
```

```python
class School(object):
    def __init__(self,title):
        self.title = title

class Course(object):
    def __init__(self,name,school_object):
        self.name = name
        self.school = school_object

class Classes(object):
    def __init__(self,cname,course_object):
        self.cname = cname
        self.course = course_object

s1 = School('北京')

c1 = Course('Python',s1)
c2 = Course('Go',s1)

cl1 = Classes('全栈1期',c1)
```

```
class School(object):
    def __init__(self,title):
        self.title = title
    def rename(self):
        pass

class Course(object):
    def __init__(self,name,school_object):
        self.name = name
        self.school = school_object
    def reset_price(self):
        pass
class Classes(object):
    def __init__(self,cname,course_object):
        self.cname = cname
        self.course = course_object
    def sk(self):
        pass
s1 = School('北京')

c1 = Course('Python',s1)
c2 = Course('Go',s1)

cl1 = Classes('全栈1期',c1)
```

内容详细

1.嵌套

- 函数：参数可以是任意类型。
- 字典：对象和类都可以做字典的key和value
- 继承的查找关系

```
class StarkConfig(object):
    pass

class AdminSite(object):
    def __init__(self):
        self.data_list = []

    def register(self,arg):
        self.data_list.append(arg)

site = AdminSite()

obj = StarkConfig()
site.register(obj)
```

```python
class StarkConfig(object):
    def __init__(self,name,age):
        self.name = name
        self.age = age

class AdminSite(object):
    def __init__(self):
        self.data_list = []
        self.sk = None

    def set_sk(self,arg):
        self.sk = arg


site = AdminSite() # data_list = []  sk = StarkConfig
site.set_sk(StarkConfig)
site.sk('alex',19)
```

```python
class StackConfig(object):
    pass

class Foo(object):
    pass

class Base(object):
    pass

class AdminSite(object):
    def __init__(self):
        self._register = {}

    def registry(self,key,arg):
        self._register[key] = arg

site = AdminSite()
site.registry(1,StackConfig)
site.registry(2,StackConfig)
site.registry(3,StackConfig)
site.registry(4,Foo)
site.registry(5,Base)

for k,v in site._register.items():
    print(k,v() )
```

```
class StackConfig(object):
    pass

class UserConfig(StackConfig):
    pass


class AdminSite(object):
```

```
    def __init__(self):
        self._register = {}

    def registry(self,key,arg=StackConfig):
        self._register[key] = arg

    def run(self):
        for key,value in self._register.items():
            obj = value()
            print(key,obj)
site = AdminSite()
site.registry(1)
site.registry(2,StackConfig)
site.registry(3,UserConfig)
site.run()
```

```
class StackConfig(object):
    list_display = '李邵奇'

class UserConfig(StackConfig):
    list_display = '利奇航'


class AdminSite(object):
    def __init__(self):
        self._register = {}

    def registry(self,key,arg=StackConfig):
        self._register[key] = arg

    def run(self):
        for key,value in self._register.items():
            obj = value()
            print(key,obj.list_display)
site = AdminSite()
site.registry(1)
site.registry(2,StackConfig)
site.registry(3,UserConfig)
site.run()
```

```
class StackConfig(object):
    list_display = '李邵奇'

    def changelist_view(self):
        print(self.list_display)

class UserConfig(StackConfig):
    list_display = '利奇航'

class AdminSite(object):
    def __init__(self):
        self._register = {}
```

```
    def registry(self,key,arg=StackConfig):
        self._register[key] = arg

    def run(self):
        for key,value in self._register.items():
            obj = value()
            obj.changelist_view()
site = AdminSite()
site.registry(1)
site.registry(2,StackConfig)
site.registry(3,UserConfig)
site.run()
```

2.特殊成员

2.1 `__init__`

```
class Foo:
    """
    类是干啥的。。。。
    """
    def __init__(self,a1):
        """
        初始化方法
        :param a1:
        """
        self.a1 = a1

obj = Foo('alex')
```

2.2 `__new__`

```
class Foo(object):
    def __init__(self):
        """
        用于给对象中赋值，初始化方法
        """
        self.x = 123
    def __new__(cls, *args, **kwargs):
        """
        用于创建空对象，构造方法
        :param args:
        :param kwargs:
        :return:
        """
        return object.__new__(cls)

obj = Foo()
```

2.3 `__cal__`

```
class Foo(object):
    def __call__(self, *args, **kwargs):
        print('执行call方法')

# obj = Foo()
# obj()
Foo()()
```

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
from wsgiref.simple_server import make_server

def func(environ,start_response):
    start_response("200 OK", [('Content-Type', 'text/plain; charset=utf-8')])
    return ['你好'.encode("utf-8")  ]

class Foo(object):

    def __call__(self, environ,start_response):
        start_response("200 OK", [('Content-Type', 'text/html; charset=utf-8')])
        return ['你<h1 style="color:red;">不好</h1>'.encode("utf-8")]


# 作用：写一个网站，用户只要来方法，就自动找到第三个参数并执行。
server = make_server('127.0.0.1', 8000, Foo())
server.serve_forever()
```

2.4 `__getitem__ __setitem__ __delitem__`

```
class Foo(object):

    def __setitem__(self, key, value):
        pass

    def __getitem__(self, item):
        return item + 'uuu'

    def __delitem__(self, key):
        pass


obj1 = Foo()
obj1['k1'] = 123  # 内部会自动调用 __setitem__方法
val = obj1['xxx']  # 内部会自动调用 __getitem__方法
print(val)
del obj1['ttt']  # 内部会自动调用 __delitem__ 方法
```

2.5 `__str__`

```
class Foo(object):
    def __str__(self):
        """
        只有在打印对象时，会自动化调用此方法，并将其返回值在页面显示出来
        :return:
        """
        return 'asdfasudfasdfsad'

obj = Foo()
print(obj)
```

```
class User(object):
    def __init__(self,name,email):
        self.name = name
        self.email = email
    def __str__(self):
        return "%s %s" %(self.name,self.email,)
user_list = [User('二狗','2g@qq.com'),User('二蛋','2d@qq.com'),User('狗蛋','xx@qq.com')]
for item in user_list:
    print(item)
```

2.6 `__dict__`

```
class Foo(object):
    def __init__(self,name,age,email):
        self.name = name
        self.age = age
        self.email = email

obj = Foo('alex',19,'xxxx@qq.com')
print(obj)
print(obj.name)
print(obj.age)
print(obj.email)
val = obj.__dict__ # 去对象中找到所有变量并将其转换为字典
print(val)
```

2.7 上下文管理【面试题】

```
class Foo(object):
    def __enter__(self):
        self.x = open('a.txt',mode='a',encoding='utf-8')
        return self.x
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.x.close()

with Foo() as ff:
    ff.write('alex')
    ff.write('alex')
    ff.write('alex')
    ff.write('alex')
```

```
# class Context:
#     def __enter__(self):
#         print('进入')
#         return self
#
#     def __exit__(self, exc_type, exc_val, exc_tb):
#         print('推出')
#
#     def do_something(self):
#         print('内部执行')
#
# with Context() as ctx:
#     print('内部执行')
#     ctx.do_something()


class Foo(object):
    def do_something(self):
        print('内部执行')

class Context:
    def __enter__(self):
        print('进入')
        return Foo()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('推出')

with Context() as ctx:
    print('内部执行')
    ctx.do_something()
```

2.8 两个对象相加

```
val = 5 + 8
print(val)

val = "alex" + "sb"
```

```
print(val)

class Foo(object):
    def __add__(self, other):
        return 123

obj1 = Foo()
obj2 = Foo()
val  = obj1 + obj2
print(val)
```

特殊成员：就是为了能够快速实现执行某些方法而生。

3.内置函数补充

3.1 type，查看类型

```
class Foo:
    pass

obj = Foo()

if type(obj) == Foo:
    print('obj是Foo类的对象')
```

3.2 issubclass

```
class Base:
    pass

class Base1(Base):
    pass

class Foo(Base1):
    pass

class Bar:
    pass

print(issubclass(Bar,Base))
print(issubclass(Foo,Base))
```

3.3 isinstance

```python
class Base(object):
    pass

class Foo(Base):
    pass

obj = Foo()

print(isinstance(obj,Foo))  # 判断obj是否是Foo类或其基类的实例（对象）
print(isinstance(obj,Base)) # 判断obj是否是Foo类或其基类的实例（对象）
```

4.super

```python
class Base(object):
    def func(self):
        print('base.func')
        return 123


class Foo(Base):
    def func(self):
        v1 = super().func()
        print('foo.func',v1)

obj = Foo()
obj.func()
# super().func() 去父类中找func方法并执行
```

```python
class Bar(object):
    def func(self):
        print('bar.func')
        return 123

class Base(Bar):
    pass

class Foo(Base):
    def func(self):
        v1 = super().func()
        print('foo.func',v1)

obj = Foo()
obj.func()
# super().func() 根据类的继承关系，按照顺序挨个找func方法并执行(找到第一个就不在找了)
```

```python
class Base(object): # Base -> object
    def func(self):
        super().func()
        print('base.func')
```

```
class Bar(object):
    def func(self):
        print('bar.func')

class Foo(Base,Bar): # Foo -> Base -> Bar
    pass

obj = Foo()
obj.func()

# super().func() 根据self对象所属类的继承关系，按照顺序挨个找func方法并执行(找到第一个就不在找了)
```

5.异常处理

5.1 基本格式

```
try:
    pass
except Exception as e:
    pass
```

```
try:
    v = []
    v[11111] # IndexError
except ValueError as e:
    pass
except IndexError as e:
    pass
except Exception as e:
    print(e) # e是Exception类的对象，中有一个错误信息。
```

```
try:
    int('asdf')
except Exception as e:
    print(e) # e是Exception类的对象，中有一个错误信息。
finally:
    print('最后无论对错都会执行')

# ##################### 特殊情况 ##########################
def func():
    try:
        # v = 1
        # return 123
        int('asdf')
    except Exception as e:
        print(e) # e是Exception类的对象，中有一个错误信息。
        return 123
    finally:
        print('最后')

func()
```

5.2 主动触发异常

```
try:
    int('123')
    raise Exception('阿萨大大是阿斯蒂') # 代码中主动抛出异常
except Exception as e:
    print(e)
```

```
def func():
    result = True
    try:
        with open('x.log',mode='r',encoding='utf-8') as f:
            data = f.read()
        if 'alex' not in data:
            raise Exception()
    except Exception as e:
        result = False
    return result
```

5.3 自定义异常

```
class MyException(Exception):
    pass

try:
    raise MyException('asdf')
except MyException as e:
    print(e)
```

```
class MyException(Exception):
    def __init__(self,message):
        super().__init__()
        self.message = message

try:
    raise MyException('asdf')
except MyException as e:
    print(e.message)
```

总结

- 特殊成员 (**)
- 嵌套
- type/issubclass/isinstacne
- super
- 异常