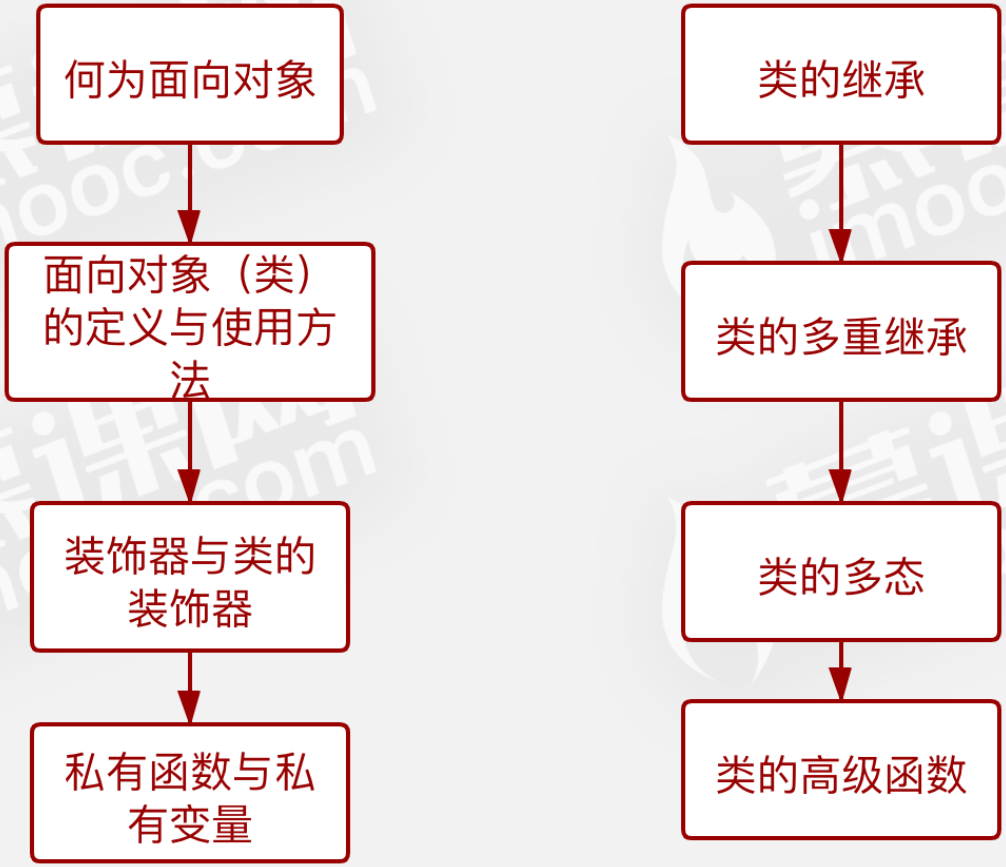
Python面向对象与异常

本周内容

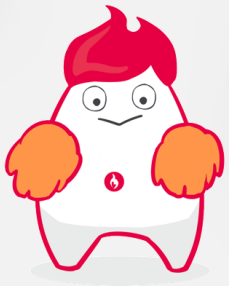
- 面向对象
- 异常

面向对象


何为面向对象
面向对象（类）的定义与使用方法
装饰器与类的装饰器
私有函数与私有变量
类的继承
类的多重继承
类的多态
类的高级函数

异常

初识面向对象编程



本节课内容

- 什么是面向对象编程（类）
- 类的定义与调用
- self
- 类的构造函数

什么是面向对象编程

- 利用（面向）对象（属性与方法）去进行编码的过程
- 自定义对象数据类型就是面向对象中的类（class）的概念

类的关键字class

class来声明类，类的名称首字母大小，
多单词情况下每个单词首字母大写

类的定义

```
class Name(object):
    attr = something

    def func(self):
        do
```

注意缩进

类的使用

```
class Person(object):
    name = '小慕'

    def dump(self):
        print(f'{self.name} is dumping')
```

```
xiaomu = Person()
print(xiaomu.name)
xiaomu.dump()
```

类的实例化

通过实例化进行属性调用

通过实例化进行函数调用

类的参数self

- self 是类函数中的必传参数，且必须放在第一个参数位置
- self 是一个对象，他代表实例化的变量自身
- self 可以直接通过点来定义一个类变量　　self.name = ‘dewei’
- self中的变量与含有self参数的函数可以在类中的任何一个函数内随意调用
- 非函数中定义的变量在定义的时候不用self

类的构造函数

类中的一种默认函数， 用来将类实例化的同时， 将参数传入类中

构造函数的创建

```
def __init__(self, a, b):
    self.a = a
    self.b = b
```

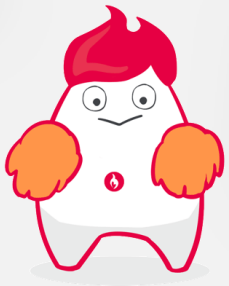
构造函数的用法

```
In [18]: class Test(object):
    ...:     def __init__(self, a):
    ...:         self.a = a
    ...:     def run(self):
    ...:         print(self.a)
    ...:

In [19]: t = Test(1)

In [20]: t.run()
1
```

对象的生命周期



本节课内容

- 对象的生命周期

对象的生命周期

- __开头__结尾的类函数都是类的默认函数

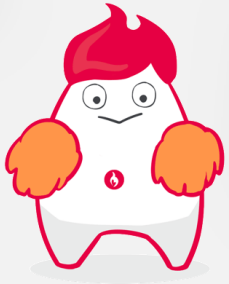
内存中分配一个内存块

实例化__init__：对象生命的开始

从内存中释放这个内存块

__del__：删除对象

私有函数和私有变量



本节课内容

- 什么是私有函数和私有变量
- 私有函数与私有变量的定义方法

什么是私有函数私有变量

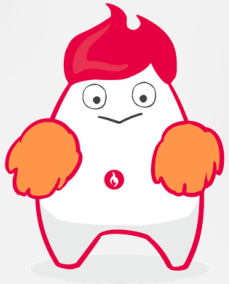
- 无法被实例化后的对象调用的类中的函数与变量
- 类内部可以调用私有函数与变量
- 只希望类内部业务调用使用，不希望被使用者调用

私有函数与私有变量的定义方法

- 在变量或函数前添加__(2个下横线），变量或函数名后边无需添加

```
class Person(object):
    def __init__(self, name):
        self.name = name
        self.__age = 33
    def dump(self):
        print(self.name, self.__age)
    def __cry(self):
        return  ‘I want cry’
```

Python中的封装



本节课内容

◆python中的封装概念

python中封装的概念

将不对外的私有属性或方法通过可对外使用的函数而使用（类中定义私有的,只有类内部使用,外部无法访问）

这样做的主要原因：保护私隐，明确区分内外

封装的例子

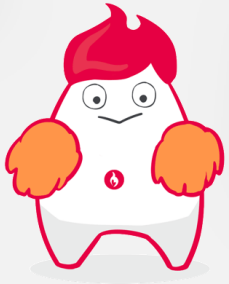
```
class Parent(object):
    def __hello(self, data):
        print('hello %s' % data)

    def helloworld(self):
        self.__hello('world')

if __name__ == '__main__':
    p = Parent()
    p.helloworld()
```

```
hello world
```

装饰器



本节课内容

- 什么是装饰器
- 装饰器的定义
- 装饰器的用法

什么是装饰器

- 也是一种函数
- 可以接受函数作为参数
- 可以返回函数

```
def a():
    def b():
        print('hello')
    b()
a()
b()
```

- 接收一个函数，内部对其处理，然后返回一个新函数，动态的增强函数功能
- 将b函数在a函数中执行，在a函数中可以选择执行或不执行b函数，也可以对b函数的结果进行二次加工处理

装饰器的定义

```
def out(func_args):          外围函数
    def inter(*args, **kwargs):          内嵌函数
        return func_args(*args, **kwargs)
    return inter          外围函数返回内嵌函数
```

装饰器的用法

- 将被调用的函数直接作为参数传入装饰器的外围函数括弧
- 将装饰器与被调用函数绑定在一起
- @符号+装饰器函数放在被调用函数的上一行，下一行创建函数，只需要直接调用被执行函数即可

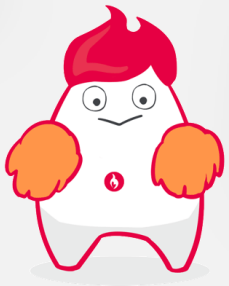
```
def a(func):
    def b(*args, **kwargs):
        return func(*args, **kwargs)
    return b

def c(name):
    print(name)
a(c('dewei'))  # dewei
```

```
@a
def c(name):
    print(name)

c('dewei')
```

类中的装饰器



本节课内容

- classmethod
- staticmethod
- property

classmethod的功能

将类函数可以不经过实例化而直接被调用

classmethod的功能与用法

用法：

```
@classmethod
def func(cls, ...):
    do
```

参数介绍：

cls 替代普通类函数中的self，

变为cls，代表当前操作的是类

```
class Test(object):
    @classmethod
    def add(cls, a, b):
        return a + b

Test.add(1, 2)
```

staticmethod的功能

- 将类函数可以不经过实例化而直接被调用,被该装饰器调用的函数不许传递self或cls参数，且无法再该函数内调用其它类函数或类变量

staticmethod的用法

用法：

```
@staticmethod
def func(...):
    do
```

参数介绍：

函数题内无cls或self参数

```
class Test(object):
    @staticmethod
    def add(a, b):
        return a + b

Test.add(1, 2)
```

property的功能

将类函数的执行免去括弧，类似于调用属性（变量）,只适用于无参数的类函数（self与cls除外）

property的用法

用法：

```
@property
def func(self):
    do
```

参数介绍：

无重要函数说明

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

    @property
    def call_name(self):
        return 'hello {}'.format(self.name)

test = Test('小慕')
result = test.call_name
print(result)  # hello 小慕
```

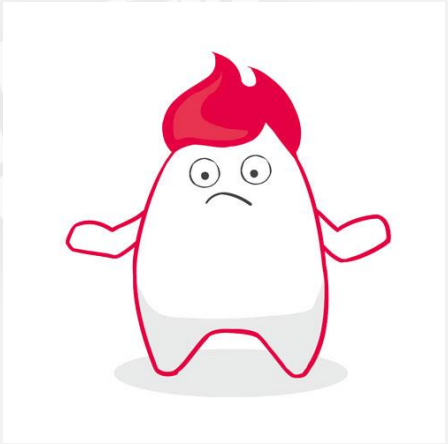
类的继承

本节课内容

- 什么是继承
- 父（基）类与子类
- 继承的用法

什么是继承

- 通过继承基类来得到基类的功能
- 所以我们把被继承的类称作父类或基类，继承者被称作子类
- 代码的重用



说话
直立行走
独立思考

父类与子类的关系

- 子类拥有父类的所有属性和方法
- 父类不具备子类自有的属性和方法

Python中类的继承

```
class Parent(object):
    def talk(self):
        print('talk')
    def think(self):
        print('think')

class Child(Parent):
    def swimming(self):
        print('child can swimming')
```

```
In [2]: c = Child()

In [3]: c.talk()
talk

In [4]: c.swimming()
child can swimming
```

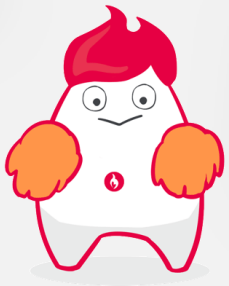
```
In [5]: p = Parent()

In [6]: p.talk()
talk

In [7]: p.swimming()
---------------------------------------------------------------
AttributeError                                Traceback
<ipython-input-7-fe1637923429> in <module>()
```

- 定义子类时，将父类传入子类参数内
- 子类实例化可以调用自己与父类的函数与变量
- 父类无法调用子类的函数与变量

类的super函数



本节课内容

- super函数的作用
- super函数的用法

super函数的作用

python子类继承父类的方法而使用的关键字，
当子类继承父类后，就可以使用父类的方法

super的用法

```
class Parent(object):
    def __init__(self):
        print('hello i am parent')
```

```
class Child(Parent):
    def __init__(self):
        print('hello i am child')
        super(Child, self).__init__()
```

当前类　类的实例　使用父类的方法

类的多态

本节课内容

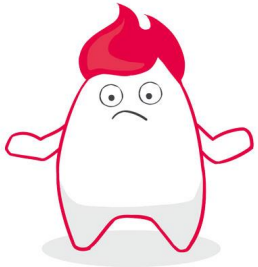
- 什么是类的多态
- 多态的用法

什么是类的多态

- 同一个功能的多状态化

小慕爸爸
平淡的说话

小慕哥哥
说话，语速很快



小慕
说话，语速很慢

多态的用法

子类中重写父类的方法

类的多重继承

本节课内容

- 什么是多重继承
- 多重继承的方法

什么是多重继承

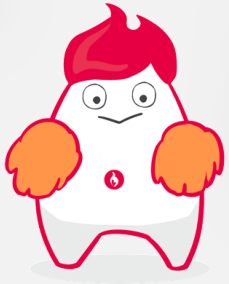
可以继承多个基（父）类

多重继承的方法

class Child(Parent1, Parent2, Parent3...)

- 将被继承的类放入子类的参数位中，用逗号隔开
- 从左向右一次继承

类的高级函数



本节课内容

- __str__
- __getattr__
- __setattr__
- __call__

__str__的功能

如果定义了该函数，当print当前实例化对象的时候，会返回该函数的return信息

__str__的用法

用法：

def __str__(self):

return str_type

参数：

无

返回值：

一般返回对于该类的描述信息

```
class Test(object):
    def __str__(self):
        return '这是关于这个类的描述'

test = Test()
print(test)
```

__getattr__的功能

当调用的属性或者方法不存在时，会返回该方法定义的信息

__getattr__的用法

用法：

```
def __getattr__(self, key):
    print(something...)
```

参数：

key：调用任意不存在的属性名

返回值：

可以是任意类型也可以不进行返回

```
class Test(object):
    def __getattr__(self, key):
        print('这个key: {} 不存在'.format(key))

test = Test()
test.a
```

这个key: a 不存在

__setattr__的功能

拦截当前类中不存在的属性与值

__setattr__的用法

用法:

```
def __settattr__(self, key, value):
    self.__dict__[key] = value
```

参数:

key 当前的属性名

value 当前的参数对应的值

返回值:

无

```
In [2]: class Test(object):
   ...:     def __setattr__(self, key, value):
   ...:         if key not in self.__dict__:
   ...:             self.__dict__[key] = value
   ...:

In [3]: t = Test()

In [4]: t.name = 'dewei'

In [5]: t.name
Out[5]: 'dewei'
```

__call__的功能

本质是将一个类变成一个函数

__call__的用法

用法：

```
def __call__(self, *args, **kwargs):
    print( ‘call will start’ )
```

参数：

可传任意参数

返回值：

与函数情况相同可有可无

```
In [13]: class Test(object):
   ....:     def __call__(self, **kwargs):
   ....:         print('args is {}'.format(kwargs))
   ....:

In [14]: t = Test()

In [15]: t(name='dewei')
args is {'name': 'dewei'}
```