A L G O R I T H M

P R A C T I C E

Python面向对象

Steven Tang

渡一教育
Duyi Education

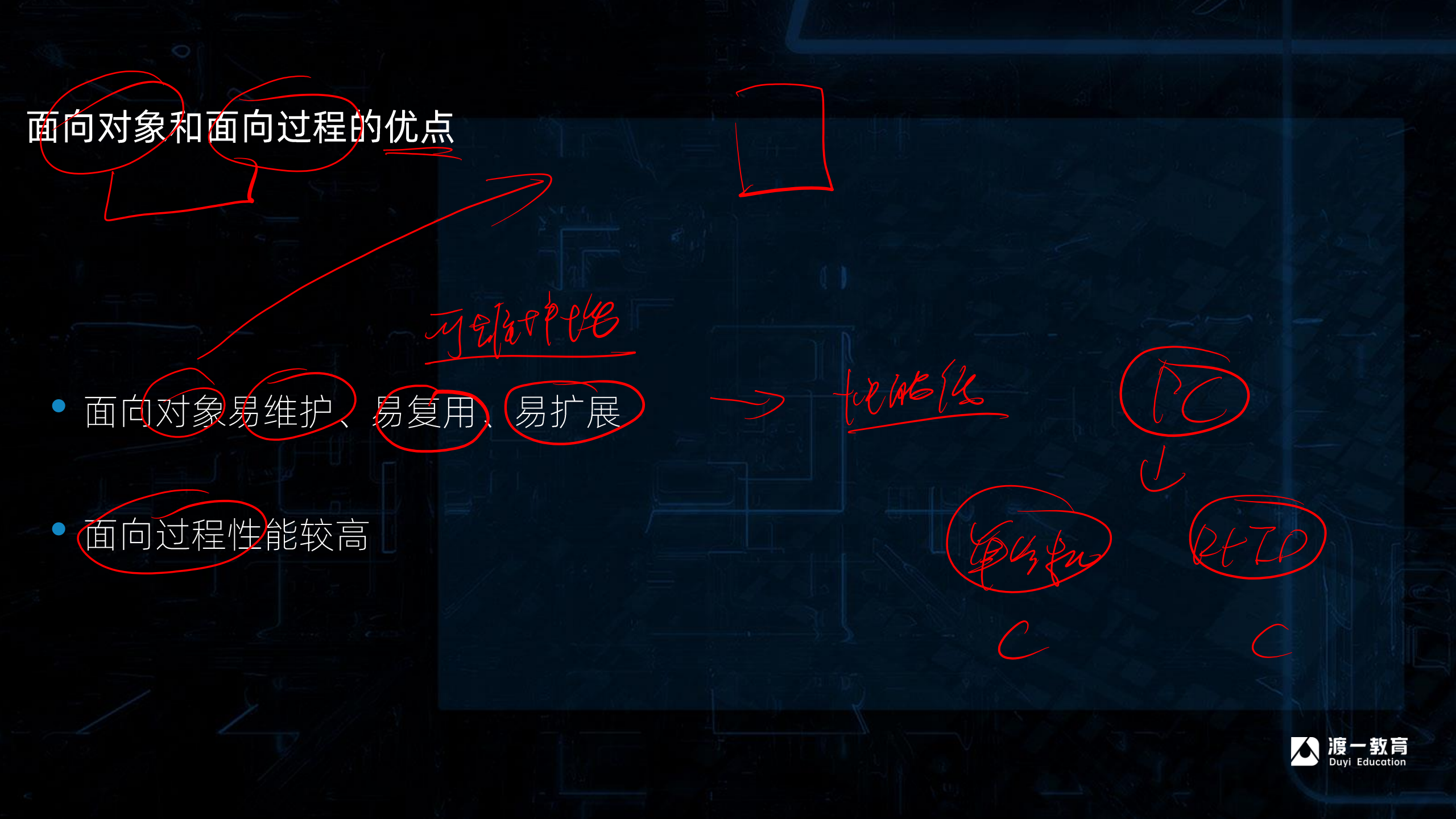
面向对象的编程---object oriented programming

- OOP，是一种编程的思想。OOP把对象当成一个程序的基本单元，一个对象包含了数据和操作数据的函数。面向对象的出现极大的提高了编程的效率，使其编程的重用性增高。

面向对象与面向过程

面向对象和面向过程的优点

- 面向对象易维护、易复用、易扩展
- 面向过程性能较高



面向对象的编程的一些基本概念

类：用来描述具有相同属性和方法的对象的集合，它定义该集合中每个对象所共有的属性和方法，对象是类的实例。

类属性：类属性在整个实例化的对象中是公用的，类属性定义在类中且在函数体之外，类属性通常不作为实例属性使用。

数据成员：类变量或者实例变量用于处理类及其实例对象的相关数据。

实例化：创建一个类的实例，类的具体对象。

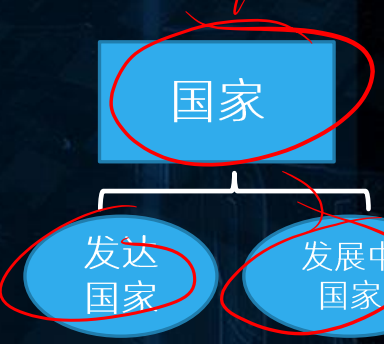
方法：类中定义的函数。

类的定义

- 类是对现实世界中一些事物的封装，所有类的开头都要包括关键字class，紧接着的是类名和冒号，随后是定义类的类体代码。

  语法格式如下：

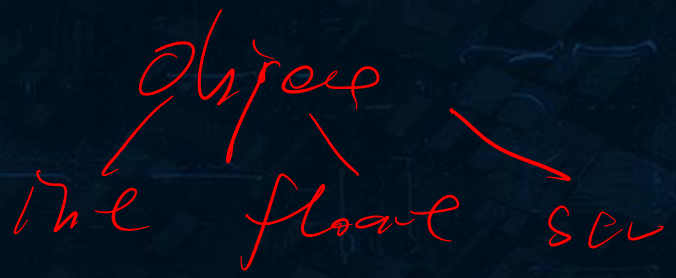
```
class ClassName:
    """documentation string"""
    <statement_1>
    <statement_2>
    .....
    <statement_N>
```



object类

object是“所有类之父”。如果你的类没有继承任何其他父类，object将作为默认的父类，它位于所有类继承结构的最上层。

```
class test(object):
    ......
    ......
```



类的定义和实例化

- 类是对象的类型（概念），对象是类的实例（事实）。类是抽象的概念，而对象是一个你能够摸得着，看得到的实体。二者相辅相成，谁也离不开谁。万物皆对象，对象拥有自己的特征和行为。

类的定义和实例化

```
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性，私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    #定义类本身的方法
    def speak(self):
        print("%s is speaking: I am %d years old" %(self.name,self.age))
#类调用
p = people('tom',10,30)
p.speak()

p.name='B'
p.speak()
```

```
tom is speaking: I am 10 years old
B is speaking: I am 10 years old
```

渡一教育 Duyi Education

类的属性与方法

1.类的公有属性

public_attrs：符合正常的变量命名规则，开头没有下划线，类外部可以直接进行访问。如上例中的name、age等。

2.类的私有属性

__private_attrs：由两个下划线开头，声明该属性为私有，不能在类的外部被使用或直接访问。在类内部的方法中使用时的格式为self.__private_attrs。

类属性与方法

```python
class Counter:
    __privateCount = 1  # 私有属性

    publicCount = 1     # 公有属性

    def count(self):
        self.__privateCount += 1
        self.publicCount += 1
        print (self.__privateCount)
counter_1 = Counter()
counter_1.count()
# 打印数据
print (counter_1.publicCount)
# 打印数据
print (counter_1.__privateCount)
# 报错，实例不能访问私有属性
```

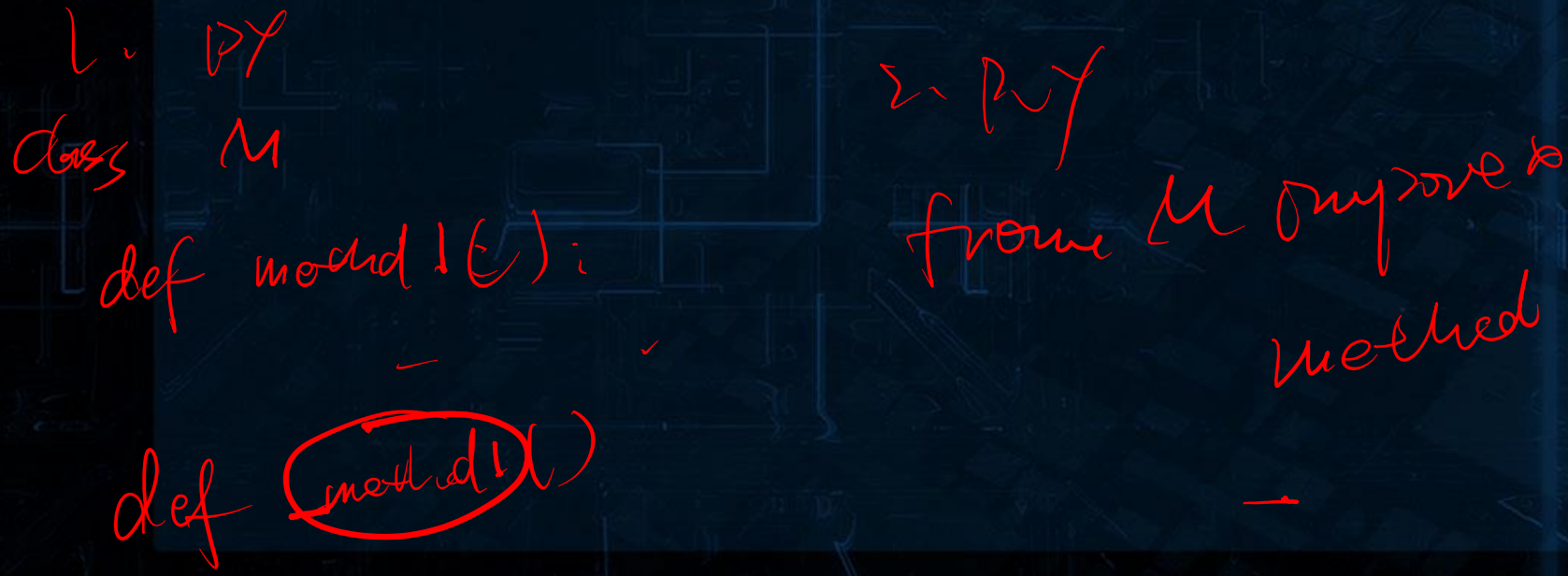
```
Traceback (most recent call last):

  File "<ipython-input-8-1aaf0f262703>", line 3, in <module>
    print (counter_1.__privateCount)

AttributeError: 'Counter' object has no attribute '__privateCount'
```

Python中 的_ 和 __

- 在python的类中，没有真正的私有化，不管是方法还是属性，为了编程的需要，约定加了下划线 _ 的属性和方法不属于API，不应该在类的外面访问，也不会被from M import * 导入。下面的代码演示加了_ 的方法，以及在类外面对其的可访问性。



Python面向对象的重要术语：

- 1、多态（polymorphism）：一个函数有多种表现形式，调用一个方法有多种形式，但是表现出的方法是不一样的。
- 2、继承（inheritance）子项继承父项的某些功能，在程序中表现某种联系
- 3、封装（encapsulation）把需要重用的函数或者功能封装，方便其他程序直接调用

封装（Encapsulation）

- 封装，顾名思义就是将内容封装到某个地方，以后再去调用被封装在某处的内容。
- 对于面向对象的封装来说，其实就是使用构造方法将内容封装到 对象 中，然后通过对象直接或者self间接获取被封装的内容。

封装（Encapsulation）

- 隐藏对象的属性和实现细节，仅对外提供公共访问方式
- 好处
  将变化隔离、便于使用、提高复用性、提高安全性
- 原则
  将不需要对外提供的内容隐藏起来；把属性隐藏，提供公共方法对其访问

继承（Inheritance）

多态（polymorphism）

面向对象的各种方法

- 静态方法（用这个装饰器来表示 @staticmethod ）
- 类方法（用这个装饰器来表示 @classmethod)
- 属性方法 (用这个装饰器表示 @property)

面向对象的各种方法

```
class Person(object):
    name="杰克"
    def __init__(self, name):
        self.name = name

    def eat0(self):
        print("%s is eating" % self.name)

    @classmethod
    def eat1(cls):
        print("%s is eating" % cls.name)

    @staticmethod  # 把eat方法变为静态方法
    def eat2(name):
        print("%s is eating" % name)

    @property
    def getname(self):
        return self.name
d = Person("xiaoming")
d.eat0()
d.eat1()
d.eat2("zhangsan")
print(d.getname)
```

python 面向对象中的set,get,del

- get方法 实际上就是属性方法 (用@property)
- set方法 (用这个装饰器来表示 @方法名.setter)
- del方法 (用这个装饰器来表示 @方法名.deleter)

```python
    @property
    def getname(self):
        return self.name

    @getname.setter
    def getname(self,name):

        self.name=str(name)

    @getname.deleter
    def getname(self):
        self.name=None
```

特殊成员__init__

- __init__()：是一个特殊的方法属于类的专有方法，被称为类的构造函数或初始化方法,方法的前面和后面都有两个下划线。这是为了避免Python默认方法和普通方法发生名称的冲突。每当创建类的实例化对象的时候，__init__()方法都会默认被运行作用就是初始化已实例化后的对象。
- Python 中不允许同一类中有多个构造方法。

特殊成员__add__

```
class abc:
    def __init__(self, age):
        self.age=age
    def __add__(self, obj):
        return self.age+obj.age

a1=abc(10)
a2=abc(15)
print(a1+a2)
```

特殊成员__dict__

- python 中__dict__存储了该对象的一些属性，类和实例分别拥有自己的__dict__，在__init__中声明的变量，会存到实例的__dict__中

```
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性，私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    #定义类本身的方法
    def speak(self):
        print("%s is speaking: I am %d years old" %(self.name,self.age))
#类调用
p = people('tom',10,30)
p.speak()
```

```
In [7]: people.__dict__
Out[7]:
mappingproxy({'__module__': '__main__',
              'name': '',
              'age': 0,
              '_people__weight': 0,
              '__init__': <function
__main__.people.__init__(self, n, a, w)>,
              'speak': <function
__main__.people.speak(self)>,
              '__dict__': <attribute '__dict__' of
'people' objects>,
              '__weakref__': <attribute '__weakref__' of
'people' objects>,
              '__doc__': None})
```

```
In [9]: p.__dict__
Out[9]: {'name': 'tom', 'age': 10, '_people__weight': 30}
```

isinstance和issubclass

- isinstance不只可以判断数据类型，也可以判断对象是否是这个类的对象或者是这个类的子类的对象。
- issubclass用来判断一个类是否是某个类的子类，返回的是一个bool类型数据。

Python中的main方法

```
类和方法定义
def class1:
...
def class2:
...
```

```
主函数定义def main():
...
...
```

```
调用主函数：
if __name__ ==‘__main__’:
main()
```

if __name__ ==‘__main__’:的作用

- 一个python文件通常有两种使用方法，第一是作为脚本直接执行，第二是 import 到其他的 python 脚本中被调用执行。

- 因此 if __name__ == 'main': 的作用就是控制这两种情况执行代码的过程，在 if __name__ == 'main': 下的代码只有在第一种情况下（即文件作为脚本直接执行）才会被执行，而 import 到其他脚本中是不会被执行的。

Python 中import 自定义py文件

argparse库使用

- public static void main(String[] args){
  - 代码.....
- }

argparse库使用

```python
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-i","--testint",type=int,help="防止老年痴呆")
    parser.add_argument("--teststr", type=str, help="防止老年痴呆",required=True)
    parser.add_argument("-n", "--testnum", nargs="+", type=int)

    parser.add_argument("--test", action='store_false', help="防止老年痴呆")

    args = parser.parse_args()
    print(args.testint)
    print(args.test)
    print(args.teststr)
    print(args.testnum)

if __name__=='__main__':
    main()
```

*，**函数参数传递

- *args可以理解为变元的缩写，而**kwargs可以理解为关键字变元的缩写。

```
def test(param1,*args,**kwargs):
    
    print('first parameter is :', param1)
    for index,var in enumerate(args):
        print("index is %d,var is %s"%(index,var))
    for key in kwargs:
        print ("the key is: %s, and the value is: %s" %(key,kwargs[key]))
        
        
test('hello','a','b',a = 1,b = 2,c = 3)
```

```
first parameter is : hello
index is 0,var is a
index is 1,var is b
the key is: a, and the value is: 1
the key is: b, and the value is: 2
the key is: c, and the value is: 3
```

vars函数

```python
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性，私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    #定义类本身的方法
    def speak(self):
        print("%s is speaking: I am %d years old" %(self.name,self.age))
#类调用
p = people('tom',10,30)
p.speak()
```

vars可以把某个类的实例转成一个由它的属性和值组成的字典，包括公共属性和私有属性。

```
In [2]: vars(p)
Out[2]: {'name': 'tom', 'age': 10, '_people__weight': 30}
```

面向对象综合运用

本节课难点

1. python 与其他面向对象语言的区别

2. 普通方法、静态方法(@staticmethod)、类方法（@classmethod)的区别

3. *，**和argpaser的用法

作业

- 构建一个类，利用__dict__方法来更新它的属性。
- 在另外一个main文件中，调用这个类，并利用argparse传入参数新建实例。
- 设计一个类方法，能够将类中的默认属性值取出。