python基础

1、不定长参数

（1）*args：（表示的就是将实参中按照位置传值，多出来的值都给args，且以元祖的方式呈现）

```
def sum(*a):
 for n in a:
 result=result+n
 print(result)
```

（2）kwargs：（表示的就是形参中按照关键字传值把多余的传值以字典的方式呈现）**

```
def fn3(**a):
 print("a"=,a,type(a))
fn3(b=1,d=2,c=3)
-->>{"b":1,"d":2,"c":3}
```

2、文档注释，在形参中用：号添加数据类型的注释，但不会强制使用这种数据类型，只是作为提示

```
def man(a:int,b:str,c:int):
    print(a,b,c)
```

3、递归

递归将大问题分解为小问题，知道问题无法分解时，再去解决问题

为避免递归死循环，因此递归的设计必须符合两个条件

1.基线条件

-问题可以被分解为最小问题，当满足基线条件时，递归就不在执行了

2.递归条件

-将问题继续分解的条件

```
#递归#求10的阶乘
#普通累加法
def factorial(n):
    """
    该函数用来求任意数的阶乘
    参数:
        n要求阶乘的数字
    """
    result=n;
     for i in range(1,result):
         result=result*i;
     return result;
```

```
13
14 #递归法
15 def factorial_1(n):
16     """
17     该函数用来求任意数的阶乘
18     参数:
19         n要求阶乘的数字
20     """
21     #基线条件，判断n是否为1，如果为1则此时不能再继续递归
22     if n==1:
23         #将1最为阶乘的最小值，直接返回
24         return 1;
25     #递归条件
26     return n*factorial(n-1);
27
28
29 if __name__ == "__main__":
30     print(factorial(10));
```

4、装饰器

避免重复写同样的代码，同时避免维护的时候要一个个手动修改相同的代码。例如时间函数

在不改变原有函数的情况下，给函数赋予新的功能

例如在语法糖@装饰的代码中，@的部分是装饰器函数。@的下一行写的函数就是原函数

可以同时为一个函数附加多个装饰器（越靠里面的装饰器越先执行）

```
1  import time
2
3  # 定义装饰器
4  def time_calc(func):
5   def wrapper(*args, **kargs):
6   start_time = time.time()
7   f = func(*args,**kargs)
8   exec_time = time.time() - start_time
9   return f
10   return wrapper
11
12 # 使用装饰器
13 @time_calc #@是一种语法糖符号，@以下的内容相当于是time_calc中定义的函数
14 def add(a, b):
15   return a + b
16
17 add(1,2)
```

```
18
19 @time_calc
20 def sub(a, b):
21  return a - b
22
23 sub(1,2)
```

5、面向对象

（1）构造方法

```
1 #面向对象
2 #1--init方法中传入参数
3 class Dog_one:
4     def __init__(self,name,age,color):
5         self.name=name
6         self.age=age
7         self.color=color
8     def eat(self):
9         print("狗狗在吃东西")
10         print(self.name,"狗狗说骨头很好吃")
11     def run(self):
12         print("狗狗在飞快的跑")
13         print(self.name,"是条柯基狗")
14
15 dog=Dog_one(name="小白",age=1,color="白色")
16 dog.eat()
17
18 #2--init方法中可以不用传参数（self参数除外）,直接将参数写在inti的变量中固定死例如: self.name="小黑"
19 class Dog_two:
20     def __init__(self):
21         self.name="小黑"
22         self.age=2
23         self.color="黑色"
24     def eat(self):
25         print("狗狗在吃东西")
26         print(self.name,"狗狗说骨头很好吃")
27     def run(self):
28         print("狗狗在飞快的跑")
29         print(self.name,"是条柯基狗")
30
31 ddog=Dog_two()
```

```
ddog.eat()
ddog.name="小白" #可以重新定义属性
print(ddog.name) #实例化后可以直接访问对象的属性
```

（2）封装

```
#3--类的封装(self.__name就是类的封装的写法，只能在类里面被调用，外部不能调用)
class card:
    def __init__(self,name,pwd,ban):
        self.__name=name  #加两个下划线__就是封装的方式
        self.__pwd=pwd
        self.__ban=ban
    def cun(self):
        print("存款")
    #如果想要外部调用私有属性，必须先写一个方法return出去
 #类似java写getter和setter方法
    #获取对象
 def get_name(self):
 return self.__name;
 #设置对象
 def set_name(self,name):
 self.__name=name;

c=card("xiongyuan","123456",10000)
c.cun()
print(c.get_name())
c.set_name("小黑")

```

6、property装饰器

用来将get方法转换为对象的属性，使用property装饰器的时候，get/set方法的属性名都要和构造方法中的属性一致

```
class person:
    def __init__(self,name):
        self._name=name;

    #get方法用property装饰器
    @property
    def name(self):
        return self._name;

     #set方法用‘属性名.setter’
     @name.setter
```

```
    def name(self,name):
        self._name=name;

p=person("xiaoxiong");
#调用set方法给属性name赋值
p.name="xiongyuan";
#调用get方法
print(p.name);
```

7、继承

```
#1--继承(全部继承)
class father:
    def football(self):
        pass
    def pingpang(self):
        pass
    def movie(self):
        pass

class sun(father): #加个括号，加入father类名这样就继承了父类的方法
    def music(self):
        pass

man=sun()
man.football()
man.movie()
man.music()
man.pingpang()

#2--继承（部分）
class father:
    def football(self):
        pass
    def pingpang(self):
        pass
    def movie(self):
        pass

class sun(father): #加个括号，加入father类名这样就继承了父类的方法
    def music(self):
        pass
```

```
    def movie(self):
        pass   #当子类方法的名称和父类方法的名称一致的时候，表示子类这个方法自有，非继承
```

8、重写

```
class sun(father): #加个括号，加入father类名这样就继承了父类的方法
    def music(self):
        pass
    def movie(self):
        pass
 #当子类方法的名称和父类方法的名称一致的时候，
 表#示子类这个方法自有，方法的重写
```

9、继承的super()

```
class Animal:
    def __init__(self,name,age):
        self._name=name;
    def run(self):
        print("动物会跑");
    def bark(self):
        print("狗叫");

class Dog(Animal):
    #获取当前类的父类
    super().__init__(self,name)
    self._age=age;
```

10、多继承

```
class a():
 pass
class b():
 pass
class c(a,b):
 pass
```

11、多态

```

```