# 172.16.77.152

# 1、python基础

## 1、python中的变量

## 2、python中的数据类型

### 整型 浮点型 复数 布尔类型

### 复数分为实部和虚部

## 3、python中的算术运算符

### +

### -

### *

### /

### // 地板除(取整运算)

### % 取余运算

### ** 幂运算

## 4、if条件判断语句

### 让程序根据条件选择性的执行某条或者某些语句

```
if 条件表达式1:
    语句块1
elif 条件表达式2:
    语句块2
else:
    语句块3
```

### if与elif语句的区别

**当程序中需要使用多条语句进行判断的时候，如果判断语句全部为if语句，那当第一条if语句满足条件后，程序不会结束，而会继续向下判断所有的if语句中的真值表达式，执行所有满足表达式的语句。如果使用if语句和elif语句，当第一条if语句或者任意一个elif语句满足条件后，程序不会再向下继续判断，直接结束程序的执行**

## 5、if语句的嵌套

```
n=int(input("请输入一个月份："))
if 1<=n<=12:
    if n<=3:
        print("春季！")
    elif n<=6:
        print("夏季！")
    elif n<=9:
        print("秋季！")
    else:
        print("冬季！")
else:
    print("您输入的月份有误！")
```

## 6、字符串 str

**在python中用引号引起来的部分都称为字符串**

''

""

''''''

""""""""""

**引号的区别:**

**1、单引号内的双引号不算做结束符**

**2、双引号内的单引号不算做结束符**

**3、三引号一般用于表示函数或者类的文档字符串**

**4、三引号内可以包含单引号和双引号，三引号内的换行符会自动转换为\n**

## 7、字符串的运算(不可变序列)

**+ 用于拼接字符串**

**+= 用于原字符串与运算符右侧的字符串进行拼接**

*** 生成重复的字符串(只能和正整型数进行运算)**

***= 生成重复的字符串**

# 8、字符串的索引与切片

## 索引 index

**从字符串中获取任意一个字符**

**索引分为正向索引和反向索引**

**正向索引从0开始，第一个元素的索引为0，第二个元素索引为1，以此类推，最后一个元素的索引为字符串的长度-1**

**反向索引从-1开始，最后一个元素的索引为-1，倒数第二个元素的索引为-2，以此类推，第一个元素的索引为字符串长度的相反数**

**s="A B C D E F"**

**正向索引 0 1 2 3 4 5**

**反向索引 -6 -5 -4 -3 -2 -1**

## 语法规则:

**字符串[整数表达式]**

## 切片 slice

**从字符串中获取连续或者带有一定间隔的字符**

## 语法规则:

**字符串[起始索引:终止索引:步长]**

**起始索引:切片切下的位置**

**终止索引:切片的终止点，但是不包含终止点**

**步长：切片每次获取完元素后，移动的方向和偏移量(没有步长相当于步长为1)**

**当步长为正数时，取正向切片**

**当步长为负数时，取反向切片**

**当切片带有步长时，切片获取完第一个元素后，用第一个元素的索引加上步长得到一个新的索引，然后获取这个新索引所对应的元素，以此类推**

**当步长为负数时，起始索引所对应的元素必须在终止索引所对应元素的右边**

# 9、while循环

## 让程序根据条件重复的执行某条或某些语句

```
i=循环变量初始值
while 真值表达式:
    语句块1
else:
    语句块2
```

## 从终端输入一个数字打印出如下的图形

**如输入：5**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

```
n=int(input("请输入一个整数:"))
i=1
while i<=n:
    j = 1
    while j <= n:
        print(j, end=" ")
        j += 1
    print()
    i+=1
```

# 10、for循环

## 遍历可迭代对象中的数据元素

## 可迭代对象是指能够依次获取数据元素的对象

```
for 变量 in 可迭代对象:
    语句块1
else:
    语句块2
```

```
s="ABCDEF"
for i in s:
    print(i)
else:
    print("for循环因迭代结束而终止！")
```

# 11、for循环嵌套

```
for i in "ABC":
    for j in "123":
        print(i+j)
```

# 12、列表 list

## 列表可以存储任意类型的数据

## 列表是可变序列

## 1、表示方式

```
>>> L=[]
>>> L
[]
>>> type(L)
<class 'list'>
>>> L=["abc",100,200,3.14,[1,2]]
>>> L
['abc', 100, 200, 3.14, [1, 2]]
>>>
```

## 2、列表的运算

**+**

**+=**

*****

***=**

```
>>> s="abc"
>>> id(s)
2537354902976
>>> s=s+"123"
>>> s
'abc123'
>>> id(s)
```

```
2537384186128
>>>
>>>
>>> s="abc"
>>> id(s)
2537354902976
>>> s+="123"
>>> s
'abc123'
>>> id(s)
2537384186184
>>>
>>>
>>>
>>> L=[1,2,3]
>>> id(L)
2537384176264
>>> L=L+[4,5,6]
>>> L
[1, 2, 3, 4, 5, 6]
>>> id(L)
2537384175432
>>>
>>>
>>> L=[1,2,3]
>>> id(L)
2537384176264
>>> L+=[4,5,6]
>>> L
[1, 2, 3, 4, 5, 6]
>>> id(L)
2537384176264
>>>
```

## 3、索引与切片

**列表的索引与切片规则和字符串的索引切片规则完全相同**

## 4、索引与切片赋值

```
>>> L=[1,2,3,4,5,6,7]
>>>
>>> L[2]
3
>>> L[2]="B"
>>> L
[1, 2, 'B', 4, 5, 6, 7]
```

**切片赋值**

**1、当切片赋值取出的数据为连续的时候，可以赋值给他任意个数据**

2、当切片赋值的步长不为1时，切片取出的数据个数和赋值的数据个数必须相同

3、当切片赋值的起始索引和终止索引相同时，代表在这个索引所对应的元素前面插入数据

# 13、字典 dict

## 字典是可变序列，字典的存储是无序的

## 字典的存储方式是由键值对映射存储

## 每个键值对之间用逗号分隔，键和值之间用冒号进行连接

## 字典的键必须为不可变元素，字典的值可以为任意元素

## 字典的键不可重复，是唯一的

## 1、表示方式

```
>>> d={}
>>> d
{}
>>> type(d)
<class 'dict'>
>>> d={"name":"小明","age":20}
>>> d
{'name': '小明', 'age': 20}
>>>
```

## 2、字典的基本操作

### 1、增加键值对

**增加的键值对的键值不存在原字典中**

```
>>> d
{'name': '小明', 'age': 20}
>>> d["sex"]="男"
>>> d
{'name': '小明', 'age': 20, 'sex': '男'}
>>>
```

### 2、修改键值对

**增加的键值对的键存在于原字典中**

```
>>> d
{'name': '小明', 'age': 20, 'sex': '男'}
>>> d["age"]=30
>>> d
{'name': '小明', 'age': 30, 'sex': '男'}
>>>
```

### 3、删除键值对

```
>>> d
{'name': '小明', 'age': 30, 'sex': '男'}
>>> del d["sex"]
>>> d
{'name': '小明', 'age': 30}
>>>
```

## 3、字典的遍历

**字典的任何操作都是有键来进行操作**

```
d={"name":"小明","age":20,"sex":"男"}

for key in d:
    print(d[key])
```

# 14、元组 tuple

## 元组是不可变序列

## 元组可以存储任意类型的元素

## 元组的索引与切片规则和字符串中的规则完全相同

## 1、表示方式

**当元组中只存放一个元素时，需要在这个元素后面加一个逗号，用来区分是单个的数据对象还是一个元组**

```
>>> t=(100)
>>> t
100
>>> type(t)
<class 'int'>
>>>
>>> t=(100,)
>>> t
(100,)
>>>
>>> type(t)
<class 'tuple'>
>>>
```

# 15、函数

```
def 函数名(参数列表):
    pass
```

## 有参数有返回值函数

## 有参数无返回值函数

## 无参数有返回值函数

## 无参数无返回值函数

```
def say_demo():
    print("hello world!")
    print("你好世界！")

say_demo()


def my_add(a,b): # a,b为函数的形参
    c=a+b
    print("a+b的值为:",c)

my_add(10,20) # 10,20为函数的实参
```

## 1、return语句

**如果函数想返回一个指定的对象，就需要用到return语句**

## 语法:

**return 对象1，对象2,....**

**1、如果一个函数中没有return语句相当于在函数末尾加了一行return None**

### 2、如果return语句后面没有返回的指定对象，相当于return None

### 3、return的作用是结束当前函数的执行，并且返回到调用该函数的地方

```
def my_sum(a,b):
    if a > b:
        return  a
    else:
        return  b

print(my_sum(10,20))
```

```
a=100 # a 为全局变量
def my_func(b,c): # b,c为函数的形参 是局部变量
    d=30
    a=200 # 相当于创建了一个和全局变量相同名称的局部变量，在函数内部没有权限更改全局变量
    print("b=",b)
    print("c=",c)
    print("a=",a) #函数内部可以访问全局变量

my_func(10,20)
# print(d) #函数外部不可以访问局部变量
print("函数执行完毕后a=",a)
```

# 16、面向对象编程

## 类 class

### 具有相同属性或者行为的对象归为一个集合即为一个类

### 类是产生对象(实例)的工厂

## 1、类的创建

```
class 类名(继承列表):
    实例方法
    类方法
    类变量
    静态方法
```

## 2、类的实例化(类的调用)

### 对象=类名(参数列表)

```
class Car(object):
    """
    此类是一个汽车类
    """
    pass

car=Car()  # 类的实例化
print(id(car))
car1=Car()
print(id(car1))
```

## 3、实例属性

### 给类产生的对象或者实例添加一个属性

### 对象.实例属性名=属性值

**car . color="红色"**

```
class Car(object):
    """
    此类是一个汽车类
    """
    pass

car=Car()  # 类的实例化
car.color="黑色" #给对象添加上颜色的属性
car1=Car()
car1.color="红色"
```

## 4、实例方法

### 让类产生的对象可以具备某些行为或者属性

```
class 类名():
    def 实例方法名(参数列表):
        pass
```

### 实例方法的本质就是定义在类内的函数，每个对象或者实例都可以调用

```
class Car(object):
    """
    此类是一个汽车类
    """
    def get_speed(self,speed):
        """
        此实例方法是给对象添加速度的一个方法
        """
        self.car_speed=speed #给对象添加上速度的属性
        print(self.color,"正在以",self.car_speed,"的速度行驶！")
```

```
car=Car()  # 类的实例化
car.color="黑色" #给对象添加上颜色的属性
car1=Car()
car1.color="红色"
```

## 5、实例方法的调用

### 实例.实例方法名(参数列表)

### self代表类的实例，他是实例方法中的第一个参数，一般默认为self

```
class Car(object):
    """
    此类是一个汽车类
    """
    def get_speed(self,speed):
        """
        此实例方法是给对象添加速度的一个方法
        """
        self.car_speed=speed #给对象添加上速度的属性
        print(self.color,"的汽车正在以",self.car_speed,"的速度行驶！")

car=Car()  # 类的实例化
car.color="黑色" #给对象添加上颜色的属性
car.get_speed(200) #调用实例方法
car1=Car()
car1.color="红色"
car1.get_speed(150)
```

## 6、类的构造函数(初始化函数)

```
class 类名(继承列表):
    def __init___(self,参数列表):
        pass
```

### 类的构造函数会在类的实例化过程中自动调用，构造函数中一般会存放一些类产生的对象所具备的公有属性

### 构造函数的名称必须为init不可改变

```
class Car(object):
    def __init__(self,color,brand):
        self.color=color
        self.brand=brand

    def get_speed(self,speed):
        self.speed=speed
        print(self.color, "的", self.brand, "汽车正在以", self.speed, "的速度行驶！")


car=Car("黑色","奔驰")
```

```
car.get_speed(120)
car1=Car("红色","奥迪")
car1.get_speed(200)
```

## 7、类的继承与派生

**类的继承代表子类可以具备父类所具有的属性和行为**

**类的派生是指在子类中添加新的功能**

```
class Father(object):
    def say(self,what):
        print("正在说:",what)

    def walk(self,distance):
        print("行走了",distance,"米")


class Son(Father):
    def study(self,subject):
        print("正在学",subject)


son=Son() # 子类实例化产生对象
son.say("你好")
son.walk(1000)
father=Father()
son.study("python")
```

# 17、python中的文本文件操作

## 操作模式

**r 只读模式**

**w 只写模式**

**a 追加写入**

**rb 二进制读取模式**

**wb 二进制写入模式**

```
with open(文件名,操作模式,encoding="utf-8") as f:
    操作语句
```

## 1、文件的写入

**f.write()**

```
with open("data.txt","w",encoding="utf-8") as f:
    f.write("hello 100 200 你好")
```

## 2、文件的读取

### f.read()

```
with open("data.txt","r",encoding="utf-8") as f:
    info=f.read()
    print(info)
```

## python -m pip install requests

## python -m pip install lxml

## python -m pip install bs4

## python -m pip install jieba

## python -m pip install wordcloud

## python -m pip install imageio

## python -m pip install matplotlib