一、python

1、什么是python?

一门编程语言，与 java , c++，php

每一种语言的编写都不一样

测试掌握的语言和项目使用的开发语言没有直接的联系

2、学习python的学习目的

- 基础的代码阅读能力
- 写自动化脚本
    - 接口自动化
    - UI自动化

3、为什么要学python

- 语法相对简单，容易入门，简单易懂
- 解释型语言
- 免费开源
- 可移植性高，所有的python程序不需要修改就可以在不同的程序上运行

4、安装python

- 安装python解释器
    - 软件包中--> python+自动化 -->python-3.7.0-amd64
        - 双击安装-->勾选add python 3.7 to path --> 点击install now -->安装完成
- 安装编辑程序
    - 根据pycharm的安装步骤安装

- 当前安装的python的版本是3.X版本
    - 与2.X版本有一定的区别是，例如：print a --> print(a)

二、python基础

1、声明编码格式

一般情况下放在首行，主要作用是在运行代码的时候程序可以快速的找到对应的编码运行

```
#!/usr/bin/python
#_*_coding:utf-8_*_
```

2、注释

将代码或者代码描述写在文件内，程序员可以看见，但是程序不会去执行被注释的内容

关键的注释，要有适量的注释

- 单行注释： 在需要注释的内容前面加上#即可，#后面的同一行的内容会被注释
- 单行注释快捷键：ctrl + /
- 多行注释： ''' 被注释的内容 '''

3、变量

- 变量的命名规则：
  - 变量是可以由只字母，数字和下划线组成（特殊字符和中文不能最为变量名称）
  - 变量不要以纯数字或者纯数字开头，区分大小写
  - 变量不能使用关键字
  - 尽量使用有意义的变量名
  - 可以使用驼峰命名法：
    - ```
      ClassNam
      ```
  - 给变量赋值：

```
ClassName = '张学友'     #变量的值为 '张学友'

一次定义多个变量：
    ClassName1,ClassName2='周杰伦','许嵩'

销毁变量：
    del ClassName

交换变量：
```

4、数据类型
```

```
数值：
    int   --整型
    float   --浮点型
布尔值：
    bool   --  True  False
字符
    str   -- '今天是阴天'
空：
    None
```

- type()

  返回参数的数据类型

  ```
  举例：
      int_=123
      t=type(int_)
      print(t)        #<class 'int'>
  ```

5、数据类型转换

转换成字符类型

```
str() :   #整型，浮点型，和布尔型的数据都可以正常转换成字符类型

int_=123
float_=33.33
bool_=True
str_='周二'

result=str(bool_)         #将括号内的任何数据转换成字符类型的函数，并且复制给result变量
t=type(result)  #使用type函数将result的类型返回并赋值给t变量

print(t)         #吧t变量打印出来
print(result)   #经result变量打印出来
```

转换成整型

```
int()  :

float_=33.93
bool_=True
str_='周二'

result=int(float_)         #将括号内的任何数据转换成字整数型的函数，并且复制给result变量
t=type(result)  #使用type函数将result的类型返回并赋值给t变量

print(t)         #吧t变量打印出来
print(result)   #经result变量打印出来


将浮点数转换成整数型时会将小数点后面的数字截取掉，不会进行四舍五入
```

布尔值的True和False转换成int数据类分别成：1和0
只有纯数据数字的字符类型的数据可以转化成整型，其他不行（带小数点的也不行）

转换成布尔值

```
bool()  :

    int类型转换成bool类型时，处理0以外都是true
    float类型转换成bool类型时，处理0.0以外都是true
    将非空数据转换成bool类型的时候，都为true
    将空数据转换成bool类型时，都为False
    空数据：  0 , 0.0 , '' , None , () , [] , {}
```

6、运算符

算数运算

```
加减乘除：  +  -  *  /
取模：  %    #求余，如果计算存在负数时，计算逻辑和正数的求余不太一样
整除：  //
幂 ：   **

注意：
    /的最终结果为浮点类型
    // 和 % 如果是int与int 进行 / 或者进行%，结果为int类型，其余为浮点数据

举例：
    num=20
    result = num//3.3
    print(result)
    print(type(result))
```

赋值运算

```
等于：    =
加等于：     +=
减等于：     -=
乘等于：     *=
除等于：     /=
幂等于：     **=
取模等于：    %=

举例：
    num=20
    num += 2    #num = num + 2
    print(num)
    print(type(num))
```

关系运算

```
>  <  ==  >=  <=  !=
```

逻辑运算

```
and :并且
not :非  # is not
or  :或者
举例：
    print(6>9 and 5<9)      #False
```

运算符的优先级

```
** --> 乘除 --> 加减  --> 关系  --> 逻辑（not>and>or）
通过（）可以改变运算优先级
```

7、转义符以及内置函数

```
转义符：
    \   #如果写在特定的字母前面，将有特殊的效果，如果是写在特殊字符前，可以起到转化为普通字符的效果

    举例：
        str_='我说：\'今天是阴天\' '
        print(str_)          #我说：'今天是阴天 '
\t  :  制表符，相当于一个tab,四个空格为一个制表符
\n  :  换行符，相当于一个enter键
```

8、序列

概念：指的是一块可以存放连续多个指的内存空间，这些值按照一定的顺序排列，可以通过下标、索引和切片进行访问。

序列包含的内容： 字符串，列表，元组，字典（字典不能使用切片和下标）

- **下标**
  - 每个元素在变量中的位置，每个元素都有两个下标

```
str=
'1   2    3    4    5    A    B    C    D    E    F    G'
 0   1    2    3    4    5    6    7    8    9    10   11
-12-11  -10  -9   -8   -7   -6   -5   -4   -3   -2   -1
```

正向下标：从左到右

逆向下标：从右到左

- **索引**
  - 可以通过下标找到变量内的某个元素

```
str_1='12345ABCDEFG'
result=str_1[8]
print(result)   #结果：D
```

- **切片**

```
格式：   变量名[start:end:step]
start:   开始的第一个元素
         包含本身
         默认为0
end :    取值的最后一位元素，不包含自己，一般情况下需要加1

step :   步长 #举例：上楼梯，一阶上就是1，两阶上就是2
         步长不能为0，默认是1
         当步长是正数时，取值方向是由左到右
         当步长是负数时，取值方向是从右到左
         注意：要注意步长的方向要和开始及结束的方向一致

举例：
    str_1='12345ABCDEFG'

    str_1[5:8:2]         #结果：AC
    str_1[-5:-8:-1]   #结果：CBA,反方向从-5到-8
    result=str_1[5::]   #结果：ABCDEFG
    result=str_1[5:-1:]   #结果：ABCDEF
    result=str_1[:7]         #结果：12345AB
    result=str_1[::2]        #结果：135BDF
```

9、元组（tuple）

格式： 变量名=（元素1，元素2，元素3，。。。。）

- 元组的元素可以为：元组，列表，字典，整型，浮点数。。。。
- 元组可以使用索引和切片
- 元组的元素不可以更改，删除等操作
- 如果一个元组只有一个元素，则再末尾需要加逗号 举例：tuple_2=('a',)

```
举例：
    tuple_1=(1,1.1,True,'字符串',(1,(3,4)),[4,5],{6:'a'})

切片：
    tuple_1[4][1][1]         #结果：4
```

对元组的操作

- 内置函数对元组
  - max() 获取到元组内的所有元素的最大值
  - len() 获取元组的长度
  - 加法运算：

```
tuple_1=(1,1.1,True,'字符串',(1,(3,4)),[4,5],{6:'a'})
tuple_2=(1,23,5435,654,543,7546,432432,76753643543,)

result=tuple_1+tuple_2
print(result)
```

  - 判断

```
tuple1=(1,2,3,7,5,8)
print(3 in tuple1)      #True
```

10、列表(list)

格式：

变量名=[元素1，元素2，元素3，。。。。]

- 列表的元素可以是任何数据类型和数据结构
- 列表的元素是可以进行修改和删除的
- 列表可以使用索引和切片

```
list_1=[1,2.3,'五',True,(6,7),[8,9],{10:'十'}]

增加元素：
    list_1.append('老王')      #在列表的末尾增加元素
    list_1.insert(1,'老气')    #在下标为1的位置插入一个元素

修改元素：
    list_1[4]=False      #修改下标位置为4的元素为False

删除元素：
    del list_1[4]        #删除
    list_1.pop(-1)       #删除下标位置的元素
    list_1.remove('五')       #按照元素的内容进行删除
    list_1.clear()       #将整个列表删除
```

11、字典dict

格式： 变量名={键：值，键2：值2}

- 字典是通过键值对的形式保存数据，字典的元素中的值是可以改变的
- 键和值是不能单独存在的，键必须是唯一的，但是值可以重复
- 字典中的每个键是不可变的，只能使用数字，字符串或元组，但是键不能使用字典和列表
- 字典没有下标，不可以使用索引和切片

```
字典的增加和修改：
格式：
    变量名[键]=值
    注意：
        如果键存在则修改值
        如果键不存在则新增组键值对
    dict_1={1:'A',2.0:'two','三':'three',(1,2):'list',1231:{6:7}}

    dict_1['o']='AAAAAA'          #在字典中增加一组键值对
    print(dict_1)

查询：
    变量名[键]    #通过键取查找值
        print(dict_1[2.0])
    变量名.keys()      #返回所有的键
         print(dict_1.keys())
    变量名.values()     #返回所有的值
        print(dict_1.values())
    变量名.item()      #返回所有的键值对
        print(dict_1.items())

删除：
    dict_1.clear()        #清空所有的键值对
    dict_1.pop(key)       #按照键去删除键值对
```

12、数据结构的转换

转换成列表

list () --将传入的参数转换成列表

- 可以将字符串，元组，字典转换成列表结构
- 原有的数据类型转换之后不变
- 字典将键和值同时转换后，是成对的

```
字符串--> 列表：

    str1='123456abcd'
    list1=[1,2,3,'a','b','c']
    tuple1=(1,2,3,'a','b','c')
    dict1={1:'a',2:'b',3:'c'}

    result=list(dict1.items())
    print(result)
    print(type(result))
```

转换成元组

tuple() -- 将传入的参数转换成元组

- 可以将字符串，列表、字典转换成元组，结构
- 原有的数据类型转换之后不变
- 字典将键和值同时转换后，是成对的

```
str1='123456abcd'
list1=[1,2,3,'a','b','c']
tuple1=(1,2,3,'a','b','c')
dict1={1:'a',2:'b',3:'c'}

result=tuple(dict1)
print(result)
print(type(result))
```

转换成字符串

str() --- 将参数转换成字符类型，涉及数据结构的转换不可以直接用str（）

```
''.join()       #要求传入的参数必须都是字符串类型的数据才可以将之转换为字符串

str1='123456abcd'
list1=['1','2','3','a','b','c']
tuple1=('1','2','3','a','b','c')
dict1={1:'a',2:'b',3:'c'}

result=''.join(list1)
print(result)
print(type(result))
```

转换成字典

- 其他数据结构转换成字典的时候，没有办法直接使用函数进行转换。
- 因为字典是成对出现的键值对，所以除非需要转换的数据是成对出现的

```
list2=[(1, 'a'), (2, 'b'), (3, 'c')]          #成对出现的数据
result=dict(list2)
print(result)
print(type(result))
```

13、流程控制

- 通过条件决定是否需要执行子代码的内容
  - 条件的结果为True或者为非空，则执行某些代码
  - 条件的结果为False或为空，则不执行代码
  - 所有条件都满足时，执行某些代码
- 子代码块不能为空
- if 要顶格写，子代码块要tab键缩进
- elif 和 else 是不能单独去使用的，必须是和if一起出现，并且要正确的配对
- elif 和 else可以单独与if一起使用

if的四种写法

- 单独的 if

```
if 条件 :          #如果满足条件，则执行子代码块
    子代码块
```

- if 和 else :

```
if  条件 :          #如果满足条件，则执行子代码块1
    子代码块1
else:            #如果以上的条件不满足则必然会执行子代码块2
    子代码块2
```

- if 和 elif :

```
if  条件 :          #如果满足条件，则执行子代码块1
    子代码块1
elif 条件 :         #如果满足条件，则执行子代码块2
    子代码块2
```

- if 、 elif 和 else :

```
if  条件 :          #如果满足条件，则执行子代码块1
    子代码块1
elif 条件 :         #如果满足条件，则执行子代码块2
    子代码块2
else:             #如果以上的条件不满足则必然会执行子代码块3
    子代码块
```

练习：

```
编写一段代码，输入7个数字，分别是1-7
然后程序返回今天是工作日还是周末
1-5  工作日
6-7是周末

day=input('请输入数字：')    #input获取到键盘的内容后返回的是str类型
if day  not  in ('1','2','3','4','5','6','7') :
    print('请输入数字！！！')
elif day >='1' and day <= '5' :
    print('工作日！')
else:
    print('周末！')
```

14、if的嵌套

当if判断的子代码也包含了if判断时，则父代码的判断会影响到子代码

举例：

```
'''
输入两位单位数（0-9），  组成双位数(10-99)，要求得到的数字为最大.。如输入4和5 的到的结果应为54
而不是45
'''

num1=input('第一位数字：')
num2=input('第二位数字：')
x=('0','1','2','3','4','5','6','7','8','9')
if num1 in x and num2  in x :                #判断num1和num2是否同时在x的范围内
    if num1 != '0' or num2 != '0':           #判断num1和num2是否同时为0
        if num1 > num2:          #判断大小，num1大于num2则将大的数放在第一位
            print(num1 + num2)
        elif num1 < num2:        #判断大小，num2大于num1则将大的数放在第一位
            print(num2 + num1)
        else:                    #num1和num2同时为0则打印提示
            print(num2 + num1)
    else:            #如果num1和num2同时为0则自行下面的子代码块
        print('两个数字不能同时为0！')
else:                                    #如果断num1和num2不在x的范围内就执行子代码块并结束
    print('数字输入错误！')
```

15、for循环

遍历：将所有的数据代入到代码中执行

```
for i in (1,2,'a','b'):
    print(i)

i   ：  自定义名称
in  ：  将in后面的元素的个数和元素赋值非变量


举例：
    for x in (1,2,3,4,5,6,7,8,9,10):
        print('hello word!',x)

    第一遍循环：将元组中的第一个元素赋值给x,然后执行子代码块，print('hello word!',x)将被执行一次，
        结果为：'hello word!' 1
    第二次循环：将元组中的第二个元素赋值给x,然后执行子代码块，print('hello word!',x)将被执行第二次，
        结果为：'hello word!' 2
    第三次循环：将元组中的第三个元素赋值给x,然后执行子代码块，print('hello word!',x)将被执行第三次，
        结果为：'hello word!' 3
    。。。。



range(start,end,step)
    产生从start开始到end的连续数字，step负责步进值
    start ： 取值的起始值
             为0是可以省略不写
     end  :取值的结束值，不包含本身
     step ： 步长
             步长为1时可以省略

练习：
求1到100的奇数
for i in range(1,101,2):
    print(i)
```

16、whlie 循环

- 当条件为True或者为非空的时候，循环执行子代码块
- 当条件一直为True时或非空，会无限执行子代码块

```
while 条件 :
    子代码块


while True :         #判断条件
    print('hello word!')     #执行子代码块的内容
    print(1)
'''
第一次循环：判断条件为True，执行1次子代码块
第二次循环：第二次判断条件为True，执行2次子代码块
第三次次循环：第三次判断条件为True，执行3次子代码块
.....
'''
```

循环次数的控制

- 在执行子代码块的内容是改变判断的条件

```
x=0      #定义一个初始值
while x<10 :         #判断条件
    print('hello word!',x)     #执行子代码块的内容
    x+=1     #每次执行进行加1


举例：
    打印1-100的奇数
    n=1
    while n<101:
        print(n)
        n=n+2
```

- 通过执行到 break 来终止循环

```
break ：代码执行到break后会直接终止整个循环

n=0
while 1 :
    print('hello word!', n)
    if n==10 :
        break
    n=n+1
```

17、循环控制语句

- pass 代码桩

  执行到pass时，继续执行其他的代码

- continue 进行下一次循环

  当执行到continue 会直接结束当次循环，继续下一次循环

```
举例：
num=0
for i in range(1,10,1):
    if i== 5:
        continue
    print(i)
```

- break 结束整个循环

  执行到break代码时，结束整个循环

- exit 退出整个程序

  退出整个程序，不会再执行任何代码

18、导入包/库

```
import  库/包
```