2 python常用语法结构

2.1 python运算符

函数名	说明
a+b	a加b
a-b	a减b
a*b	a乘以b
a/b	a除以b
a//b	a除以b，结果只取整数部分
a/b	a除以b
a//b	a除以b，结果只取整数部分
a**b	a的b次幂
a&b	a或b都为True，则为True；对于整数，取逐位AND
a|b	a或b有一个为True，则为True；对于整数，取逐位OR
a^b	对于布尔，a或b有一个为True，则为True，二者都为True或者都为False，为False；对于整数，取逐位EXCLUSIVE-OR
a==b	a等于b，则为True
a!=b	a不等于b，则为True
a<b，a<=b	a小于（或小于等于）b，则为True
a〉b，a>=b	a大于（或大于等于）b，则为True
a is b	a和b引用同一个Python对象，则为True
a is not b	a和b引用不同的Python对象，则为True

```
#is比较与==运算符不同,s 检查的是两个对象是否相同，而不是相等
a = b = [1,2,3]
c = [1,2,3]
print(a == b, a == c,a is b,a is c)
#变量 x 和 y 指向同一个列表，而c 指向另一个列表，虽然两个的值都是一样，但是并不是同一个对象，所以这里 a is c 返回 False
```

```
True True True False
```

2逻辑判断

2.2.1内置常量False、None、True

```
False == 0 ,True == 1,type(False),type(None)
```

```
(True, True, bool, NoneType)
```

2.2.2 逻辑与 或 非 and or not¶

优先级：not and or

```
#  x and y      如果 x 为 False 、空、0，返 回 x，否则返回 y
#  x or y       如果 x 为 False、 空、0，返回  y，否则返回x
#  not  x     如果 x 为 False、 空、0，返回 True，否则返回False

a = 1
b = '1'
a and b,a or b,not a
```

```
('1', 1, False)
```

2.3 控制流语句

2.3.1 for循环

for循环是在一个集合（列表或元组）中进行迭代，或者就是一个迭代器。

```
#range可以接受三个参数，分别是起始值，上限（不包括上限），步长
for i in range(10,19,2):
    print(i)
```

```
10
12
14
16
18
```

```
#for循环在一行，[for]
x= [i for i in range(5)]
x
```

```
[0, 1, 2, 3, 4]
```

2.3.2 迭代器与生成器

- 迭代器：所有你可以用在for...in...语句中的都是可迭代的:比如lists,strings,files...因为这些可迭代的对象你可以随意的读取所以非常方便易用,但是你必须把它们的值放到内存里,当它们有很多值时就会消耗太多的内存.例如前面的zip()就是可以构建一个迭代器，迭代器本身提供了一个next方法，用于获取下一个对象成员，当用next方法获取全部成员后，再次调用next方法时，会引发StopIteration异常，这个异常不是错误，只是表示迭代已完成，因此使用迭代器迭代对象成员时，需要加入异常处理语句。
- 迭代器有两个基本的方法：iter() (生成迭代器)和 next()（返回迭代器的下一个项目）
- 生成器：使用了 yield 的函数被称为生成器（generator）。yield 是一个类似 return 的关键字，只是这个函数返回的是个生成器。当你调用这个函数的时候，函数内部的代码并不立马执行 ，这个函数只是返回一个生成器对象。当你使用for进行迭代的时候，函数中的代码才会执行

```
def generator():
    i=0
    while True:
        i += 1
        yield i
for item in generator():
    print(item)
    if item > 4:
        break
```

```
1
2
3
4
5
```

2.3.3 if、elif和else

if是最广为人知的控制流语句。它检查一个条件，如果为True，就执行后面的语句。可以只使用if而不使用else
可以使用if-elif-else的结构，中间可以包括多个elif。

```
##猜数字游戏
import random
print('请猜一个1-20的数字，你有6次机会')
answer = random.randint(1,20)
for i in range(1,3):
    print('请输入：')
    num = int(input())
    if num > answer:
        print("太大了，再猜一个")
    elif num < answer:
        print('太小了，再猜一个')
```

```
    else:
        break
if num == answer:
    print('恭喜你猜中了！正确答案是%d' %answer )
else:
    print('已经超过限制次数了，正确答案是%d' %answer)
```

```
请猜一个1-20的数字，你有6次机会
请输入：
2
太小了，再猜一个
请输入：
34
太大了，再猜一个
已经超过限制次数了，正确答案是5
```

2.3.4while循环

while循环指定了条件和代码，当条件为False或用break退出循环，代码才会退出

```
##使用标志
#导致程序结束的事件有很多时，如果在一条while 语句中检查所有这些条件，将既复杂又困难。
#在要求很多条件都满足才继续运行的程序中，可定义一个变量
#用于判断整个程序是否处于活动状态。这个变量被称为标志
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)
    if message == 'quit':
        active = False
    else:
        print(message)
```

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program.  a
```

```
a
```

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program.  quit
```

使用while循环来处理列表和字典

for 循环是一种遍历列表的有效方式，但在for 循环中不应修改列表，否则将导致Python难以跟踪其中的元素。要在遍历列表的同时对其进行修改，可使用while 循环。通过
将while 循环同列表和字典结合起来使用，可收集、存储并组织大量输入，供以后查看和显示。

2.3.5 pass、continue、break

```
#pass是Python中的非操作语句。
#代码块不需要任何动作时可以使用（作为未执行代码的占位符）；
#因为Python需要使用空白字符划定代码块，所以需要pass
x=0
if x < 0:
    print('negative!')
elif x == 0:
    pass
else:
    print('positive!')

#continue,跳过该次循环;if+continue,符合条件的跳出循环
#输出双数
i = 0
while i < 10:
    i=i+1
    if i%2 >0:
        continue
    print(i)

#break，跳出整个循环
i=1
while 1:
    print(i)
    i = i+1
    if i >5:
        break
```

2.2.6 异常处理：try..except

- 错误可以由try 和except 语句来处理。那些可能出错的语句被放在try 子句中。
- 如果错误发生，程序执行就转到接下来的except 子句开始处。
- 如果不指定错误类型，except将会捕捉所有的错误，这样可能会导致异常不到的结果
- 可以指定多个except
- 使用pass，不报告错误

try:

代码1

except:

代码2

finally:

代码3

代码1发生异常就执行代码2，无论正常与否都要执行代码3.

```
## 加法器
print('请输入数字1：')
a = input()
print('请输入数字2：')
b = input()
try:
    print(int(a)+int(b))
except:
    print('请输入数字！')
```

```
请输入数字1：
```

```
 a
```

```
请输入数字2：
```

```
 b
```

```
请输入数字！
```

```
#try和except
#raise：主动触发异常
#一旦执行raise语句，后面的代码就不执行了,可以结合try 使用
try:
    raise EOFError#触发EOFError异常
    print('正常')
except EOFError:#这里指定了错误，可以指定多个错误
    print('EOFError异常。')
```

```
EOFError异常。
```

```
#try和else
list = [1, 2, 3, 4, 5, 6, 7, 8]
try:
    list.append(100)
    print(list[8])
except IndexError:
    print('数组越界')
else:
    print(list)
```

```
100
[1, 2, 3, 4, 5, 6, 7, 8, 100]
```

```python
#try和finally
list = [1, 2, 3, 4, 5, 6, 7, 8]
try:
    list.append(100)
    print(list[10])
except IndexError:
    print('数组越界')
finally:
    print(list)
```

```
数组越界
[1, 2, 3, 4, 5, 6, 7, 8, 100]
```

```python
##with...as with语句时用于对try except finally 的简化
f=open('file_name','r')
try:
    r=f.read()
except:
    pass
finally:
    f.close()

等价于

with open('file_name','r') as f:
    r=f.read()
```