Python 循环控制

什么是循环

循环就是不断重复动作的语句

Python 的两个主要循环结构:

for 循环， 遍历序列对象内的元素 对每个元素运行一个代码块

```
for … in ….:
    do somthing
```

while 编写通用循环的一种方法

```
while 1:
    do something
```

前面我们已经在非正式的环境下见过这两种循环了 在这里我们将会说一些细节 此外我们还会研究循环中的 continue 和 break 关键字，并且还会介绍循环中常用的一些内置函数 zip map range

while 和 for 语句是用来编写重复操作的主要语法 后面我们会说到迭代 列表解析 filter map reduce 等

while 循环，只要条件为真，就不断循环，条件为假时退出循环 称为循环是因为控制权会持续返回到语句开头部分

语法格式 首行是一个测试表达式判断真假 循环体中有一列或多列缩进语句的主体 以及一个可选的 else 组成 (控制权离开循环又没有碰到 break 语句的时候才执行)

```
while <test>:
    statements1
else: #可选
    statement2
```

示例代码如下：

```
n = 5

while n > 1:

    print '123'

    n  = n - 1

else:

    print 'abc'
```

一个简单的死循环例子：

```
while 1:

    print 'hello world'
```

Break continue 一般会出现在 while 或 for 循环主体的任何地方，但通常会进一步嵌套在 if 语句中，根据某些条件来才去对应的操作

Break 跳出整个循环

Continue 跳出本次循环 来到循环的首行

Pass 什么也不做 占位

Else 只有当程序正常离开时才执行 (没有碰到 break 语句)

```
#break 语句

n  = 10

while n >= 0:

    if n == 2:  ##当 n = 2 时跳出本次循环

        break

    print n
```

```
    n = n - 1
## continue 语句
n    = 10
while n >= 0:
    if n == 2:
        n = n – 1
        continue ##跳出循环
    print n
    n = n - 1


##pass  语句占位什么也不做
while n >= 0:
    if n == 2:
        pass
    elif n == 4:
        pass
    print n
    n = n - 1
~
while 0:    ##判断为假
    print '123'    ##不执行
else:       ##但是 else 仍然会执行    因为这个循环中没有 break 语句跳出循环  所以仍算正常退出
    print 'nothing'
~
```

for 是 Python 中一个通用的序列迭代器 可以遍历任何可迭代对象内的元素 如 字符串 列表 元组 集合 字典 ##分别通过代码来实现

基本应用

Python 的循环有两种，一种是 for...in 循环，依次把 list 或 tuple 中的每个元素迭代出来，看例子：

```
names = ['Michael', 'Bob', 'Tracy']

for name in names:

    print(name)
```

执行这段代码，会依次打印 `names` 的每一个元素：

```
Michael

Bob

Tracy
```

所以 `for x in ...` 循环就是把每个元素代入变量 `x`，然后执行缩进块的语句。

再比如我们想计算 1-10 的整数之和，可以用一个 `sum` 变量做累加：

```
sum = 0

for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:

    sum = sum + x

print(sum)
```

如果要计算 1-100 的整数之和，从 1 写到 100 有点困难，幸好 Python 提供一个 `range()` 函数，可以生成一个整数序列，再通过 `list()` 函数可以转换为 list。比如 `range(5)` 生成的序列是从 0 开始小于 5 的整数：

```
>>> list(range(5))

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

`range(101)` 就可以生成 0-100 的整数序列，计算如下：

```
sum = 0

for x in range(101):

    sum = sum + x

print(sum)
```

请自行运行上述代码，看看结果是不是当年高斯同学心算出的 5050。

In [2]: range(0, 10, 3) ##range(起始，截止，步长)

Out[2]: [0, 3, 6, 9]

另一种表达方式:

For <target> in <object>:

    <statement>

else: ## continue break else 与 while 循环中都一样 都是循环离开时没有碰到 break 语句 循环正常退出的时候 会执行这里

    <statement>

res = []

for j in l:

if j > 4: #匹配到这个条件则执行 break 退出本次循环 不执行 else 部分的语句

break

    if j > 3:

        res.append(j) ##循环正常退出 打印 else 部分的语句

else:

    print res

如果迭代对象是元组序列 比如 dict.items() 方法 可以采用元组解包的方式进行循环

d = {'a': 1, 'b': 2}

d.items() 等于 [('a', 1), ('b', 2)]

for (k, v) in d.items():

print k, v

并行遍历 map 和 zip

内置的 zip 函数使用 for 循环来并行遍历多个序列 在基本的运算中 zip 会取得一个或多个序列为参数 然后返回元组的列表 将这些序列中的并排的元素配成对

In [4]: l1 = [1, 2, 3]

In [5]: l2 = [2, 3, 4]

In [6]: zip(l1, l2)

Out[6]: [(1, 2), (2, 3), (3, 4)]

当参数长度不同时 zip 会以最短序列长度为准来截断所得到的元组

In [7]: l1 = [1, 2]

In [8]: l2 = [2, 3, 4]

In [9]: zip(l1, l2)

Out[9]: [(1, 2), (2, 3)]

Map 函数对一个可迭代对象进行遍历 并将每一个元素进行操作

In [12]: l2

Out[12]: [2, 3, 4]

In [13]: map((lambda x: x+1),l2) ##匿名函数后面再讨论

Out[13]: [3, 4, 5]

用 zip 构造字典

函数 dict 可以将 [(a, 1),(b, 2),(c, 3)] 的数据结构变成字典

In [5]: dict(a=1, b=2) ##dict 的另一种使用方法

Out[5]: {'a': 1, 'b': 2}

In [14]: dict([('a', 1),('b', 2)])

Out[14]: {'a': 1, 'b': 2}

In [15]: a = ['a', 'b', 'c']

In [16]: b = [1, 2, 3]

In [17]: zip(a, b)

Out[17]: [('a', 1), ('b', 2), ('c', 3)]

In [18]: dict(zip(a, b))

Out[18]: {'a': 1, 'b': 2, 'c': 3}

产生偏移和元素 enumerate

In [22]: for (k, v) in enumerate(a):

    ....:   print k, v

....:

0 a

1 p

2 p

3 l

4 e