Python基础–循环+range

python 有两种循环的方法

一、while循环

while 循环。当条件为真的时候，永远循环下去，如果条件为假，跳出循环或不循环。
语法：

```
while 条件:
    执行体
```

当你执行下面代码，程序会一直打印 hello world。那是因为 a < b 这个条件是成立的。

```
a = 1
b = 3
while a < b:
    print("hello world")
```

如果你想停止这个死循环，有 3 种办法：
第一种，把条件弄成不成立的，这样程序就只会循环一次

```
a = 1
b = 3
while a < b:
    print("hello world")
    a = 10
```

第二种，在循环体里加上 break 关键词，break 会帮你跳出循环

```
a = 1
b = 3
while a < b:
    print("hello world")
    break
```

第三种（最简单的一种），按下 ctrl + c，但这样会报出 KeyboardInterrupt 的错误。

二、for循环

for 循环。由用户设置循环次数，可以循环或者遍历一个目标体。

语法：

```
for i in 目标体:
    执行体
```

（i 是一个随便取的变量名，大家也可以把 i 改成其他变量名）

```
animals = ['cat','dog','tiger']
```

这里有一个列表，你可以用遍历的方式来打印出每种动物，举个例子：

```
for animal in animals:
    print("动物是: " + animal)
```

```
>>>animals = ['cat','dog','tiger']
>>>for animal in animals:
        print("动物是: " + animal)

动物是: cat
动物是: dog
动物是: tiger
>>>
```

for 循环把 animals 列表的所有动物都遍历了出来。

for 循环有一个关键词 ——range () 函数。range () 函数可以写一到三个参数。你可以用 range 来进行循环。

```
for i in range(10):
        print("我循环了10次! ")
```

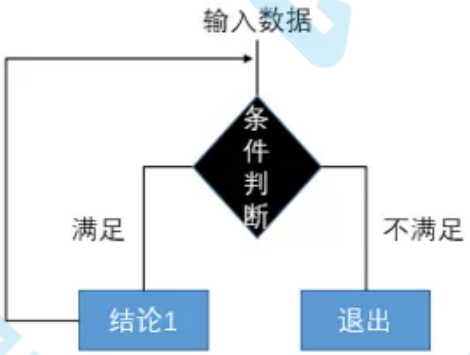
（range (10) 代表循环 10 次）

三、while循环与for循环的差异

while循环与for循环的**最大区别在于**，**while循环是基于条件判断的循环，而for循环则是基于容器的循环**。对于while循环来说，当条件满足时，将一直处于循环状态，除非碰见break关键词；对于for循环来说，当容器内的元素没有迭代结束，则一直处于循环状态，同样碰见break关键词时也会退出循环。

所以，在做循环问题时，**首先自问循环过程中有没有明确的迭代对象**（即容器），然后再根据判断结果，选择优先方案，即如果有迭代对象，则优先使用for循环，否则优先使用while循环。

while循环示意图及语法



如上图所示，当数据输入后，会立马进入条件判断，如果条件满足，则进入循环体，并继续下一轮的循环，直到条件不满足时，退出循环。所以，根据该逻辑，可以将while循环的语法表示如下：

```
# while循环通常会有初始值，这里不妨设置变量s的初始值为0
s = 0
# 无分支判断的for循环
while condition:
    statements

s = 0
# 有分支判断的for循环
while condition:
    if condition:
        statements1
    else:
        statements2
```

由于绝大多数的循环问题，都可以使用while循环或者for循环解决，为了表现while循环的优势，接下来举1个特殊的案例，体现while循环的优势。

案例：在[a,b]区间内猜一个整数

```
# 导入第三方模块
import random

# 设定被猜数据的范围
A = int(input('请输入被猜数据范围的最小值：'))
B = int(input('请输入被猜数据范围的最大值：'))
# 生成A,B之间的一个随机整数
number = random.randint(A,B)

while True:
    guess = int(input('请在{}和{}之间猜一个整数：'.format(A,B)))
    if guess > number:
        # 如果猜的偏大，则将猜的数字重新赋值给B，用于限定下一轮数据的猜测范围
        B = guess
        print('不好意思，您猜大了！')
    elif guess < number:
        # 如果猜的偏小，则将猜的数字重新赋值给A，用于限定下一轮数据的猜测范围
        A =guess
        print('不好意思，您猜小了！')
    else:
        print('恭喜，您猜正确了！')
        # break用于退出整个while循环
        break
```

如上代码所示，进入while循环之前设定了三个初始值，用于限定被猜数据的范围以及该范围内的一个随机整数。你会发现，while关键词后面不是一个具体的判断条件，而是布尔值True，这意味着while循环属于死循环（即永远不会出现条件为假而退出循环的可能）。为保证while循环可以正常退出，循环体内设置了break关键词（当用户猜对后，循环语句会来到break关键词）。

四、range()

range()是一个内置函数，这意味着Python是预先打包了它的。这个函数可以创建一个数字序列(称为range对象)并返回它。当然，您可以将这组数字用于各种目的: 如下所示，range()实际上能很好地与循环一起使用。

下面是Python help()模块提供的一个更专业的解释:

"返回一个对象，该对象会从开始数字(包括)到停止数字(不包括)按步长生成一个整数序列。range(i, j)会产生i, i+1, i+2，...，j-1。开始数字默认为0，停止数字被省略!range(4)会产生0,1,2,3。这些正是一个4元素列表的有效索引。当给定一个步长时，它指定了递增数(或递减数)。

Python中range()的语法

让我们设想最简单的场景: range(5)

生成序列.....

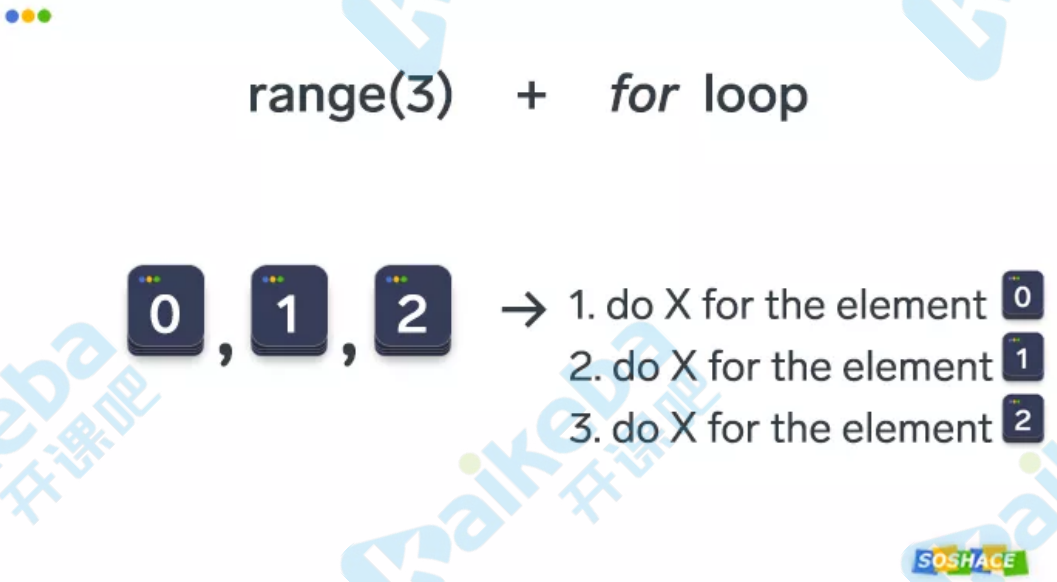
range()函数的语法很简单——我们调用函数并输入参数:

```
range([start], stop, [step])
```

让我们仔细看看每个参数的作用:

1. start是一个可选参数，它定义了序列的起始点。如果未指定，则默认为0。
2. stop是定义序列终止点的必需参数。
3. step是一个可选参数，它定义了步长大小(即序列中各个整数之间被忽略的整数数量)。如果未指定，则默认为1。

在Python中将Range()与for循环组合



循环和range的使用过程

或者，我们可以使用一个for循环——它允许我们多次执行给定的命令。这个过程称为迭代，我们可以使用各种数据结构(如字符串或列表)来指定“重复”的确切数量。当然，我们也可以使用range()函数来实现这个目的——它基本上会运行该命令N次。

```
1 for i in range(5):
2     print(i)
```

由于Python清晰的语法，我们很容易记住for循环是如何工作的:“对于某个东西中的每个元素/部分/项，这样做。”然而，它的输出将是垂直的(每个print调用将以一个新行结束)——一些开发人员可能会发现它不方便或可读性不强。要水平打印输出，我们可以在print函数中添加end参数:

```
1 for i in range(5):
2     print(i, end=', ')
```

这样，结果会更容易阅读:

```
1 0, 1, 2, 3, 4,
```

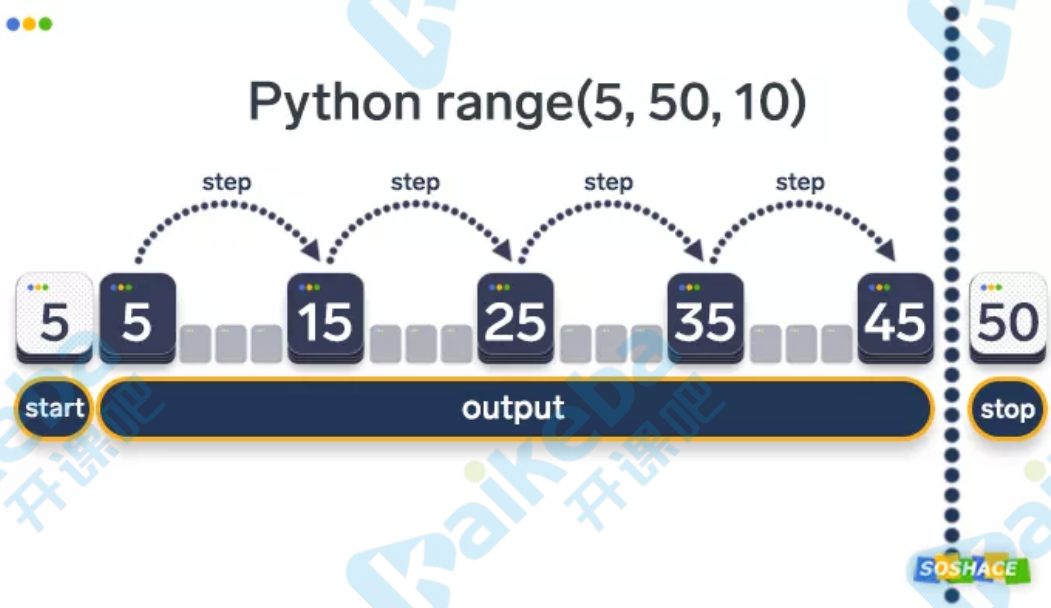
关于range()的“Stop”参数的说明(在Python中也称为包含范围)

现在，我们需要重申索引在几乎所有编程语言中的工作方式。因为我们没有指定start参数，它默认被指定为0，我们的序列就变成了0 - 5。你可能会想，如果我们把这个序列放入一个列表并打印出来，这个列表应该是这样的:

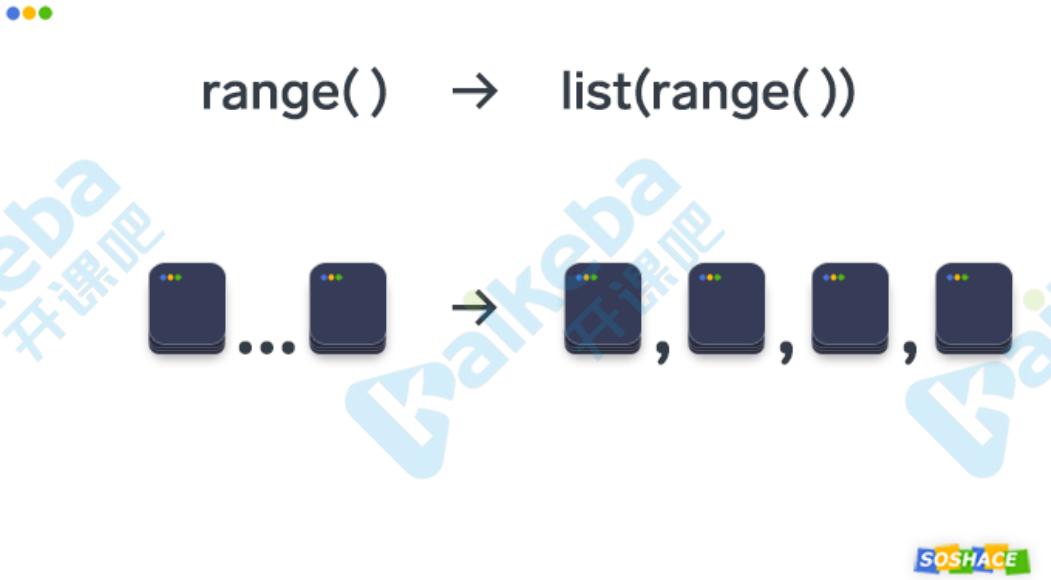
```
1 [0, 1, 2, 3, 4, 5]
```

但是，最后一个索引(即stop参数)不包含在此操作中，因此将其公式记为range(从数字X到——但是不包括——数字Z)是很有用的。

当我们想使用所有的三个参数时，我们可以来查看一个更复杂的情形:



在Python中组合range()和List()



range和list的使用过程

假设参数为5时，我们可能想要展示序列中实际使用了哪些数字。要做到这一点，我们可以使用另一个Python内置函数——list——来创建一个由函数调用range(5)将产生的数字组成的列表：

```
1 print(list(range(5)))
```

这将输出：

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