【Python】if语句、for语句、while语句

> 官方文档：
> 4. 其他流程控制工具 — Python 3.11.0a0 文档

一、if语句

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
```


```
if a == 2:
    print("a==2")
elif a == 1:
    print("a==1")
else:
    print("False")
```

二、for语句、while语句

无else子句	有else子句
while 条件： 语句块1	while 条件： 语句块1 else: 语句块2

无else子句	带else子句
for 循环变量 in 序列： 语句块1	for 循环变量 in 序列： 语句块1 else: 语句块 2

4.2. for 语句

Python 的 `for` 语句与 C 或 Pascal 中的不同。Python 的 `for` 语句不迭代算术递增数值（如 Pascal），或是给予用户定义迭代步骤和暂停条件的能力（如 C），而是迭代列表或字符串等任意序列，元素的迭代顺序与在序列中出现的顺序一致。例如：


```
for variable in ["""列表"""]:
    # 语句块
```


```
for i in [1, 2, 3, 4]:
    print(i, end="")
```

遍历某个集合的同时修改该集合的内容，很难获取想要的结果。要在遍历时修改集合的内容，应该遍历该集合的副本或创建新的集合：

```
# Create a sample collection
users = {'Hans': 'active', 'Éléonore': 'inactive', '景太郎': 'active'}

# Strategy:  Iterate over a copy
for user, status in users.copy().items():
    if status == 'inactive':
        del users[user]

# Strategy:  Create a new collection
active_users = {}
for user, status in users.items():
    if status == 'active':
        active_users[user] = status
```


因为list中当前int对象被删除了，

```
class CLanguage:
    def __init__(self):
        print("调用 __init__() 方法构造对象")
    def __del__(self):
        print("调用__del__() 销毁对象，释放其空间")
clangs = CLanguage()
del clangs
```


字典遍历时不使用列表副本，报错!

```
1   # Create a sample collection
2   users = {'Hans': 'active', 'Éléonore': 'inactive', '景太郎': 'active'}  users: {'Hans': 'act
3
4   # Strategy:  Iterate over a copy
5   for user, status in users.items():  status: 'inactive'   user: 'Éléonore'
6       if status == 'inactive':
7           del users[user]
8
9   print(users)
10
```



```
>>> users = {'Hans': 'active', 'Éléonore': 'inactive', '景太郎': 'active'}
>>> b=users.__iter__()
>>> c=b.__next__()
>>> c
'Hans'
>>> c=b.__next__()
>>> c
'Éléonore'
>>> del users['Éléonore']
>>> c=b.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size during iteration
>>>
```


但当在列表中时，似乎不会报错：

```
>>> a=[1,2,3,4]
>>> b=a.__iter__()
>>> del a[2]
>>> c=b.__next__()
>>> c
1
>>> c=b.__next__()
>>> c
2
>>> c=b.__next__()
>>> c
4
>>>
>>> k=[5,6,7,8]
>>> p=k.__iter__()
>>> v=p.__next__()
>>> v
5
>>> v=p.__next__()
>>> v
6
>>> del k[1]
>>> v=p.__next__()
>>> v
8
>>> v=p.__next__()
Traceback (most recent call last):
```


三、range()函数

可选择使用range()函数
Python3 range() 函数返回的是一个**可迭代对象**


```python
# range()返回一个可迭代对象

# 缺省形式
range(0, 5, 1)      # [0,1,2,3,4]
range(0, 5)         # [0,1,2,3,4]
range(5)            # [0,1,2,3,4]

# range范围计算
range(1, 11, 2)     # [1,3,5,7,9]

```

像C一样，**使用索引迭代序列**

range() 和 len() 组合在一起，可以按索引迭代序列：

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

不过，大多数情况下，enumerate() 函数更便捷，详见 循环的技巧 。

enumerate(*iterable*, *start=0*)

返回一个枚举对象。*iterable* 必须是一个序列，或 iterator，或其他支持迭代的对象。 enumerate() 返回的迭代器的 __next__() 方法返回一个元组，里面包含一个计数值（从 *start* 开始，默认为 0）和通过迭代 *iterable* 获得的值。

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

等价于:

```
def enumerate(sequence, start=0):
    n = start
    for elem in sequence:
        yield n, elem
        n += 1
```


使用**可迭代对象**可以**节约空间**:

range() 返回对象的操作和列表很像，但其实这两种对象不是一回事。迭代时，该对象基于所需序列返回连续项，并没有生成真正的列表，从而节省了空间。

这种对象称为可迭代对象 iterable，函数或程序结构可通过该对象获取连续项，直到所有元素全部迭代完毕。for 语句就是这样的架构， sum() 是一种把可迭代对象作为参数的函数:

```
>>> sum(range(4))  # 0 + 1 + 2 + 3
6
```


四、循环的技巧

5. 数据结构 — Python 3.11.0a0 文档

https://docs.python.org/zh-cn/3.11/tutorial/datastructures.html#tut-loopidioms

五、break\continue语句，else子句

4.4. 循环中的 break、 continue 语句及 else 子句

break 语句和 C 中的类似，用于跳出最近的 for 或 while 循环。

循环语句支持 else 子句； for 循环中，可迭代对象中的元素全部循环完毕时，或 while 循环的条件为假时，执行该子句； break 语句终止循环时，不执行该子句。 请看下面这个查找素数的循环示例：

与 if 语句相比，循环的 else 子句更像 try 的 else 子句： try 的 else 子句在未触发异常时执行，循环的 else 子句则在未运行 break 时执行。 try 语句和异常详见 异常的处理。