Python3

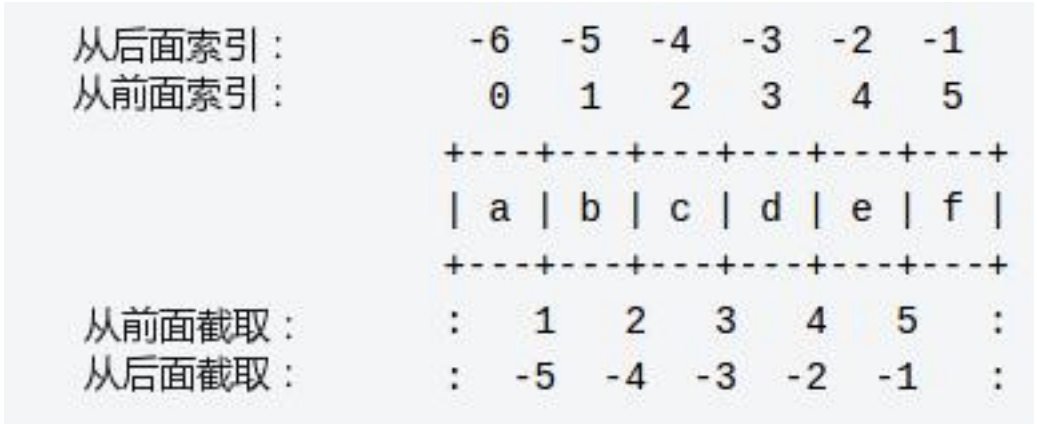
本次课程

- 深入介绍python的字符串
- 介绍python中的list（列表）、tuple（元组）、 dictionary（字典）
- 介绍python中的条件控制
- 介绍循环语句中的while循环
- 介绍无限循环

截取字符串

- Python中的字符串用单引号 ' 或双引号 " 括起来，同时使用反斜杠 \ 转义特殊字符。
- 字符串的截取的语法格式如下：

  变量[头下标:尾下标]
- 索引值以 0 为开始值，-1 为从末尾的开始位置。



截取字符串

- 加号 + 是字符串的连接符， 星号 * 表示复制当前字符串，与之结合的数字为复制的次数。

```
str = 'Runoob'

print (str)           # 输出字符串
print (str[0:-1])     # 输出第一个到倒数第二个的所有字符
print (str[0])        # 输出字符串第一个字符
print (str[2:5])      # 输出从第三个开始到第五个的字符
print (str[2:])       # 输出从第三个开始的后的所有字符
print (str * 2)       # 输出字符串两次，也可以写成 print (2 * str)
print (str + "TEST") # 连接字符串
```

- 执行以上程序会输出如下结果：

```
Runoob
Runoo
R
noo
noob
RunoobRunoob
RunoobTEST
```

转义字符串

- Python 使用反斜杠(\)转义特殊字符:

```
print('Ru\noob')
```

- 输出结果如下:

```
Ru
oob
```

- 如果你不想让反斜杠发生转义，可以在字符串前面添加一个 r，表示原始字符串:

```
print(r'Ru\noob')
```

- 输出结果如下:

```
Ru\noob
```

转义字符串

- 另外，反斜杠(\)可以作为续行符，表示下一行是上一行的延续。也可以使用 """...""" 或者 '''...''' 跨越多行。
- 注意，Python 没有单独的字符类型，一个字符就是长度为1的字符串。
- 实例：

```
>>>word = 'Python'
>>> print(word[0], word[5])
P n
>>> print(word[-1], word[-6])
n P
```

- 与 C 字符串不同的是，Python 字符串不能被改变。向一个索引位置赋值，比如word[0] = 'm'会导致错误。

小结

- 反斜杠可以用来转义，使用r可以让反斜杠不发生转义。
- 字符串可以用+运算符连接在一起，用*运算符重复。
- Python中的字符串有两种索引方式，从左往右以0开始，从右往左以-1开始。
- Python中的字符串不能改变。

列表

- List（列表） 是 Python 中使用最频繁的数据类型。
- 列表可以完成大多数集合类的数据结构实现。列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）。
- 列表是写在方括号 [] 之间、用逗号分隔开的元素列表。
- 和字符串一样，列表同样可以被索引和截取，列表被截取后返回一个包含所需元素的新列表。
- 列表截取的语法格式如下：

  变量[头下标:尾下标]

创建列表

- 创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。如下所示：

```
list1 = ['Google', 'Runoob', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

- 练习：

  创建列表list4，需包含你的名字、出生年份以及性别；

  使用print方法将list4的内容打印出来。

访问列表

- 使用下标索引来访问列表中的值，同样你也可以使用方括号的形式截取字符，如下所示：

```
list1 = ['Google', 'Runoob', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];

print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

- 以上实例输出结果：

```
list1[0]:  Google
list2[1:5]:  [2, 3, 4, 5]
```

更新列表

- 你可以对列表的数据项进行修改或更新，你也可以使用append()方法来添加列表项，如下所示：

```
list = ['Google', 'Runoob', 1997, 2000]

print ("第三个元素为 : ", list[2])
list[2] = 2001
print ("更新后的第三个元素为 : ", list[2])
```

- 以上实例输出结果：

```
第三个元素为 :  1997
更新后的第三个元素为 :  2001
```

删除列表元素

- 可以使用 del 语句来删除列表的的元素，如下实例：

```
list = ['Google', 'Runoob', 1997, 2000]

print ("原始列表 : ", list)
del list[2]
print ("删除第三个元素 : ", list)
```

- 以上实例输出结果：

```
原始列表 :  ['Google', 'Runoob', 1997, 2000]
删除第三个元素 :  ['Google', 'Runoob', 2000]
```

列表操作符

- 尝试运行下面的程序：

```
list1 = ['Python', 'Java', 'Matlab', 'Ruby']
list2 = ['Cpp', 'JavaScript', 'Swift']
len(list)
list1+list2
list1*3
'Cpp' in list2
```

列表操作符

- 列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。
- 如下所示：

Python表达式	结果	描述
len(list1)	4	长度
list1 + list2	['Python', 'Java', 'Matlab', 'Ruby', 'Cpp', 'JavaScript', 'Swift']	组合
list1*3	['Python', 'Java', 'Matlab', 'Ruby', 'Python', 'Java', 'Matlab', 'Ruby', 'Python', 'Java', 'Matlab', 'Ruby']	重复
'Cpp' in list2	True	元素是否在列表中

列表的截取与拼接

- Python的列表截取与字符串操作类型，如下所示：

```
L=['Google', 'Runoob', 'Taobao']
>>>L=['Google', 'Runoob', 'Taobao']
>>> L[2]
'Taobao'
>>> L[-2]
'Runoob'
>>> L[1:]
['Runoob', 'Taobao']
```

列表的截取与拼接

Python表达式	结果	描述
L[2]	'Taobao'	读取第三个元素
L[-2]	'Runoob'	从右侧开始读取倒数第二个元素
L[1:]	['Runoob', 'Taobao']	输出从第二个元素开始后的所有元素

列表的截取与拼接

- 列表还支持拼接操作：

```
>>>squares = [1, 4, 9, 16, 25]
>>> squares += [36, 49, 64, 81, 100]
>>> squares
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

嵌套列表

- 使用嵌套列表即在列表里创建其它列表，例如：

```
>>>a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

元组

- Python 的元组与列表类似，不同之处在于元组的元素不能修改。
- 元组使用小括号，列表使用方括号。
- 元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。如下实例：

```
>>>tup1 = ('Google', 'Runoob', 1997, 2000)
>>> tup2 = (1, 2, 3, 4, 5 )
>>> tup3 = "a", "b", "c", "d" # 不需要括号也可以
>>> type(tup3)
<class 'tuple'>
```

元组

- 创建空元组:

```
tup1 = ()
```

元组

- 元组中只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用：

```
>>>tup1 = (50)
>>> type(tup1) # 不加逗号，类型为整型
<class 'int'>
>>> tup1 = (50,)
>>> type(tup1) # 加上逗号，类型为元组
<class 'tuple'>
```

- 元组与字符串类似，下标索引从0开始，可以进行截取，组合等。

访问元组

- 元组可以使用下标索引来访问元组中的值，如下实例:

```
tup1 = ('Google', 'Runoob', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

- 以上实例输出结果:

```
tup1[0]:  Google
tup2[1:5]:  (2, 3, 4, 5)
```

修改元组

- 元组中的元素值是不允许修改的，但我们可以对元组进行连接组合，如下实例:

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')

# 以下修改元组元素操作是非法的。
# tup1[0] = 100

# 创建一个新的元组
tup3 = tup1 + tup2
print (tup3)
```

- 以上实例输出结果:

```
(12, 34.56, 'abc', 'xyz')
```

删除元组

- 元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组，如下实例:

```
tup = ('Google', 'Runoob', 1997, 2000)

print (tup)
del tup
print ("删除后的元组 tup : ")
print (tup)
```

- 以上实例元组被删除后，输出变量会有异常信息，输出如下所示:

```
('Google', 'Runoob', 1997, 2000)
删除后的元组 tup :
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print (tup)
NameError: name 'tup' is not defined
```

元组运算符

- 与字符串一样，元组之间可以使用 + 号和 * 号进行运算。这就意味着他们可以组合和复制，运算后会生成一个新的元组。

Python表达式	结果	描述
Len((1,2,3))	3	计算元素个数
(1,2,3)+(4,5,6)	(1,2,3,4,5,6)	连接
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	复制
3 in (1, 2, 3)	True	元素是否存在

元组的索引、截取

- 因为元组也是一个序列，所以我们可以访问元组中的指定位置的元素，也可以截取索引中的一段元素，如下所示：
- 元组：

```
L = ('Google', 'Taobao', 'Runoob')
```

Python表达式	结果	描述
L[2]	'Taobao'	读取第三个元素
L[-2]	'Runoob'	反向读取，读取倒数第二个元素
L[1:]	['Runoob', 'Taobao']	截取元素，从第二个开始后的所有元素。

元组的索引、截取

- 运行实例如下：

```
>>> L = ('Google', 'Taobao', 'Runoob')
>>> L[2]
'Runoob'
>>> L[-2]
'Taobao'
>>> L[1:]
('Taobao', 'Runoob')
```

字典

- 字典是另一种可变容器模型，且可存储任意类型对象。
- 字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中 ,格式如下所示：

```
d = {key1 : value1, key2 : value2 }
```

字典

- 键必须是唯一的，但值则不必。
- 值可以取任何数据类型，但键必须是不可变的，如字符串，数字或元组。
- 一个简单的字典实例：

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

创建字典

- 也可如此创建字典：

```
dict1 = { 'abc': 456 }
dict2 = { 'abc': 123, 98.6: 37 }
```

访问字典里的值

- 把相应的键放入到方括号中，如下实例:

```
dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

- 以上实例输出结果:

```
dict['Name']:  Runoob
dict['Age']:  7
```

访问字典里的值

- 如果用字典里没有的键访问数据，会输出错误如下：

```
dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

print ("dict['Alice']: ", dict['Alice'])
```

- 以上实例输出结果：

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    print ("dict['Alice']: ", dict['Alice'])
KeyError: 'Alice'
```

修改字典

- 向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对如下实例:

```
dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8               # 更新 Age
dict['School'] = "菜鸟教程"  # 添加信息


print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

- 以上实例输出结果:

```
dict['Age']:  8
dict['School']:  菜鸟教程
```

删除字典元素

- 能删单一的元素也能清空字典，清空只需一项操作。
- 显示删除一个字典用del命令，如下实例：

```
dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

del dict['Name'] # 删除键 'Name'
dict.clear()     # 清空字典
del dict         # 删除字典

print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

删除字典元素

- 但这会引发一个异常，因为用执行 del 操作后字典不再存在：

```
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    print ("dict['Age']: ", dict['Age'])
TypeError: 'type' object is not subscriptable
```

- 注：del() 方法后面也会讨论。

字典键的特性

- 字典值可以是任何的 python 对象，既可以是标准的对象，也可以是用户定义的，但键不行。
- 两个重要的点需要记住：

- 1）不允许同一个键出现两次。创建时如果同一个键被赋值两次，后一个值会被记住，如下实例：

```
dict = {'Name': 'Runoob', 'Age': 7, 'Name': '小菜鸟'}

print ("dict['Name']: ", dict['Name'])
```

字典键的特性

- 以上实例输出结果：

```
dict['Name']:  小菜鸟
```

字典键的特性

- 键必须不可变，所以可以用数字，字符串或元组充当，而用列表就不行，如下实例：

```
dict = {['Name']: 'Runoob', 'Age': 7}

print ("dict['Name']: ", dict['Name'])
```

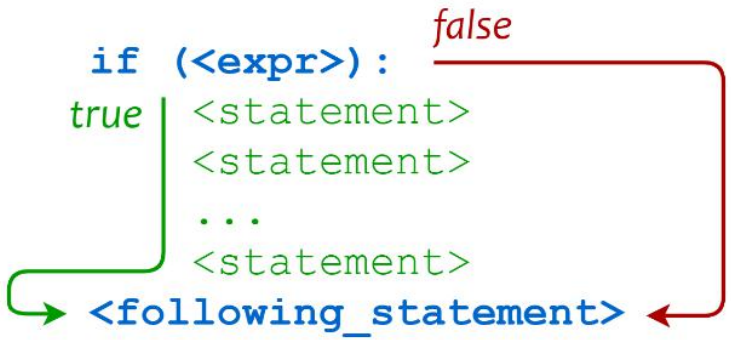
- 以上实例输出结果：

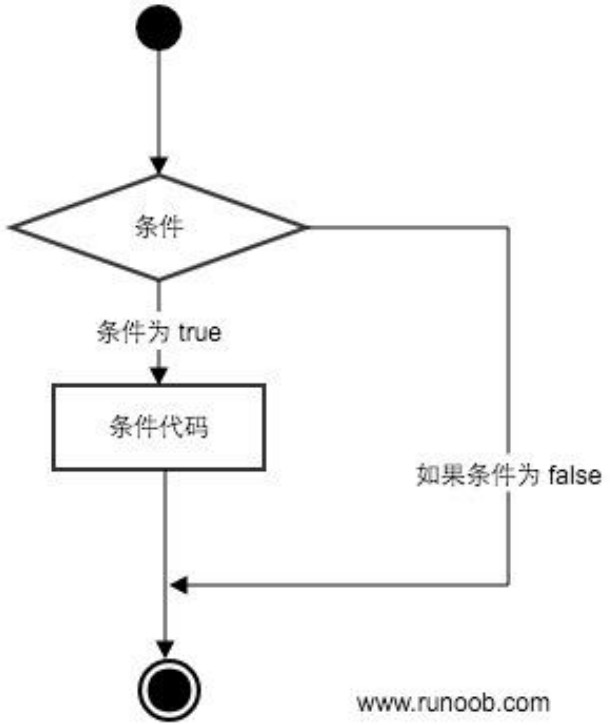
```
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    dict = {['Name']: 'Runoob', 'Age': 7}
TypeError: unhashable type: 'list'
```

条件判断

- Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。
- 可以通过下图来简单了解

  条件语句的执行过程:
- 代码执行过程：





if语句

- Python中if语句的一般形式如下所示：

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

- 如果 "condition_1" 为 True 将执行 "statement_block_1" 块语句
- 如果 "condition_1" 为False，将判断 "condition_2"
- 如果"condition_2" 为 True 将执行 "statement_block_2" 块语句
- 如果 "condition_2" 为False，将执行"statement_block_3"块语句

- Python 中用 elif 代替了 else if，所以if语句的关键字为：if – elif – else。

if语句

- 注意：

1、每个条件后面要使用冒号：，表示接下来是满足条件后要执行的语句块。

2、使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。

3、在Python中没有switch – case语句。

if语句

code

```
1  a = 1
2  while a < 7 :
3      if(a % 2 == 0):
4          print(a, "is even")
5      else:
6          print(a, "is odd")
7      a += 1
```

output

variables

if语句实例

- 以下是一个简单的 if 实例：

```
var1 = 100
if var1:
    print ("1 - if 表达式条件为 true")
    print (var1)

var2 = 0
if var2:
    print ("2 - if 表达式条件为 true")
    print (var2)
print ("Good bye!")
```

- 执行以上代码，输出结果为：

```
1 - if 表达式条件为 true
100
Good bye!
```

if语句实例

- 从结果可以看到由于变量 var2 为 0，所以对应的 if 内的语句没有执行。
- 以下实例演示了狗的年龄计算判断：

```
age = int(input("请输入你家狗狗的年龄: "))
print("")
if age <= 0:
    print("你是在逗我吧!")
elif age == 1:
    print("相当于 14 岁的人。")
elif age == 2:
    print("相当于 22 岁的人。")
elif age > 2:
    human = 22 + (age -2)*5
    print("对应人类年龄: ", human)

### 退出提示
input("点击 enter 键退出")
```

if语句实例

- 将以上脚本保存在dog.py文件中，并执行该脚本：

```
请输入你家狗狗的年龄： 1

相当于 14 岁的人。
点击 enter 键退出
```

操作运算符

操作符	描述
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于，比较两个值是否相等
!=	不等于

实例

```
# 程序演示了 == 操作符
# 使用数字
print(5 == 6)
# 使用变量
x = 5
y = 8
print(x == y)
```

```
False
False
```

实例

- high_low.py文件演示了数字的比较运算：

```
# 该实例演示了数字猜谜游戏
number = 7
guess = -1
print("数字猜谜游戏!")
while guess != number:
    guess = int(input("请输入你猜的数字："))

    if guess == number:
        print("恭喜，你猜对了！")
    elif guess < number:
        print("猜的数字小了...")
    elif guess > number:
        print("猜的数字大了...")
```

实例

- 执行以上脚本，实例输出结果如下：

```
数字猜谜游戏！
请输入你猜的数字：1
猜的数字小了...
请输入你猜的数字：9
猜的数字大了...
请输入你猜的数字：7
恭喜，你猜对了！
```

if嵌套

- 在嵌套 if 语句中，可以把 if...elif...else 结构放在另外一个 if...elif...else 结构中。

```
if 表达式1:
    语句
    if 表达式2:
        语句
    elif 表达式3:
        语句
    else:
        语句
elif 表达式4:
    语句
else:
    语句
```

if嵌套

```
num=int(input("输入一个数字：  "))
if num%2==0:
    if num%3==0:
        print ("你输入的数字可以整除 2 和 3")
    else:
        print ("你输入的数字可以整除 2，但不能整除 3")
else:
    if num%3==0:
        print ("你输入的数字可以整除 3，但不能整除 2")
    else:
        print  ("你输入的数字不能整除 2 和 3")
```

- 将以上程序保存到 test_if.py 文件中，执行后输出结果为：

```
输入一个数字：  6
你输入的数字可以整除 2 和 3
```

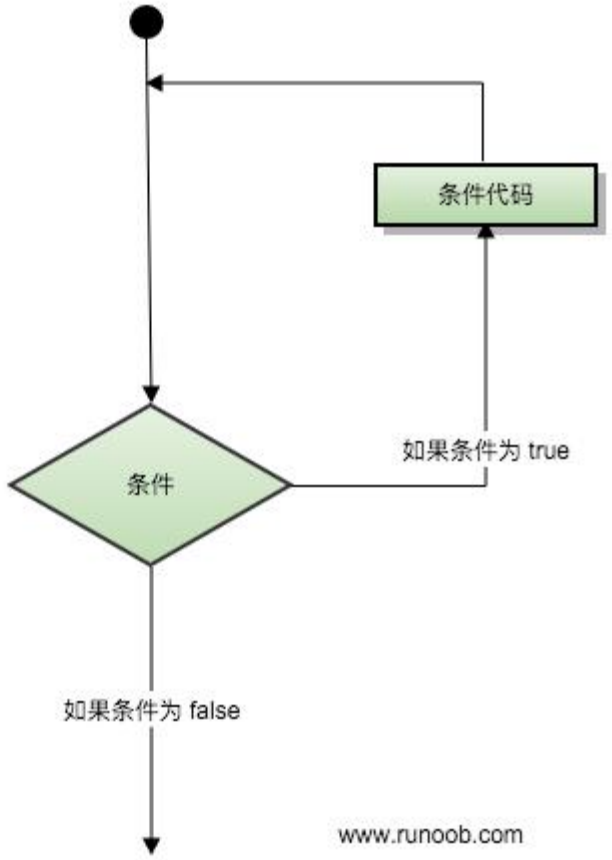
实例:BMI指数

- 使用判断语句来实现 BMI 的计算。
- BMI 指数（即身体质量指数，简称体质指数又称体重，英文为 Body Mass Index，简称BMI），是用体重公斤数除以身高米数平方得出的数字
- 将下面的程序保存在一个命名位bmi.py的文件中：

```
print('----欢迎使用BMI计算程序----')
name=input('请键入您的姓名:')
height=eval(input('请键入您的身高(m):'))
weight=eval(input('请键入您的体重(kg):'))
gender=input('请键入你的性别(F/M)')
BMI=float(float(weight)/(float(height)**2))
#公式
if BMI<=18.4:
    print('姓名:',name,'身体状态:偏瘦')
elif BMI<=23.9:
    print('姓名:',name,'身体状态:正常')
elif BMI<=27.9:
    print('姓名:',name,'身体状态:超重')
elif BMI>=28:
    print('姓名:',name,'身体状态:肥胖')
import time;
#time模块
nowtime=(time.asctime(time.localtime(time.time())))
if gender=='F':
    print('感谢',name,'女士在',nowtime,'使用本程序,祝您身体健康!')
if gender=='M':
    print('感谢',name,'先生在',nowtime,'使用本程序,祝您身体健康!')
```

循环语句

- Python 中的循环语句有 for 和 while。
- Python 循环语句的控制结构图如右所示：

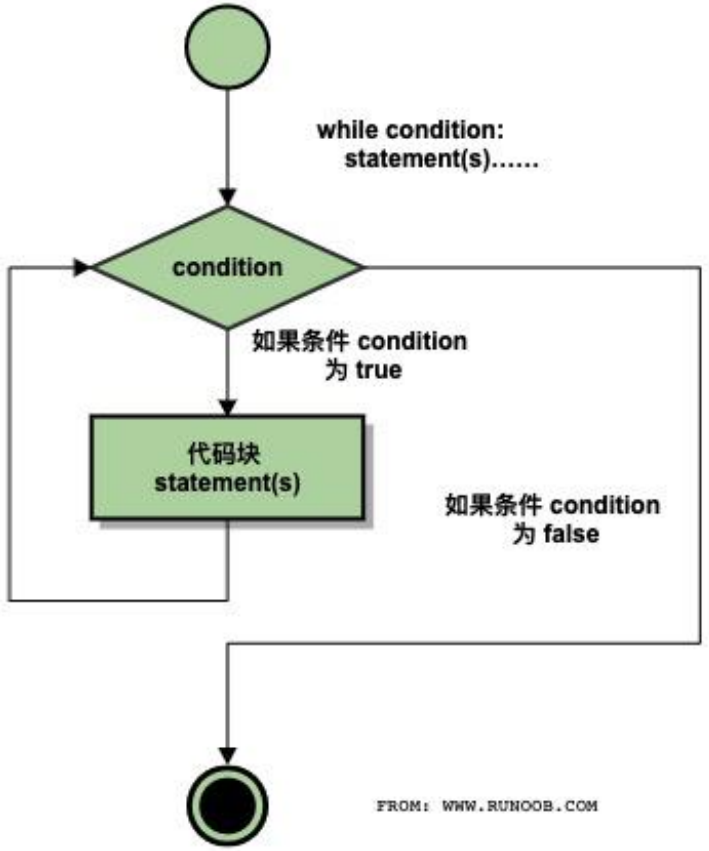


while 循环

- Python 中 while 语句的一般形式：

  while 判断条件(condition)：
      执行语句(statements)……

- 执行流程图如右：



while 循环

code

```
1  a = 1
2  while a < 10:
3   print (a)
4    a += 2
```

output

variables

while 循环

- 同样需要注意冒号和缩进。另外，在 Python 中没有 do..while 循环。
- 以下实例使用了 while 来计算 1 到 100 的总和：

```
n = 100

sum = 0
counter = 1
while counter <= n:
    sum = sum + counter
    counter += 1

print("1 到 %d 之和为: %d" % (n,sum))
```

- 执行结果如下：

```
1 到 100 之和为: 5050
```

无限循环

- 我们可以通过设置条件表达式永远不为 false 来实现无限循环，实例如下：

```
var = 1
while var == 1 :  # 表达式永远为 true
   num = int(input("输入一个数字  :"))
   print ("你输入的数字是: ", num)

print ("Good bye!")
```

- 执行以上脚本，输出结果如下：

```
输入一个数字  :5
你输入的数字是:  5
输入一个数字  :
```

无限循环

- 你可以使用 CTRL+C 来退出当前的无限循环。
- 无限循环在服务器上客户端的实时请求非常有用。

while 循环使用 else 语句

- 在 while … else 在条件语句为 false 时执行 else 的语句块。
- 语法格式如下：

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

while 循环使用 else 语句

- 循环输出数字，并判断大小：

```
count = 0
while count < 5:
   print (count, " 小于 5")
   count = count + 1
else:
   print (count, " 大于或等于 5")
```

- 执行以上脚本，输出结果如下：

```
0  小于 5
1  小于 5
2  小于 5
3  小于 5
4  小于 5
5  大于或等于 5
```

简单语句组

- 类似if语句的语法，如果你的while循环体中只有一条语句，你可以将该语句与while写在同一行中， 如下所示：
- ```
  flag = 1

  while (flag): print ('欢迎访问菜鸟教程!')

  print ("Good bye!")
  ```
- 注意：以上的无限循环你可以使用 CTRL+C 来中断循环。
- 执行以上脚本，输出结果如下：
  ```

接下来...

- python数据分析（一）
- 循环语句中的for语句
- range()函数
- break 和 continue 语句及循环中的 else 子句
- pass 语句
- ...