Python3

EDIT BY: CHEN GUO

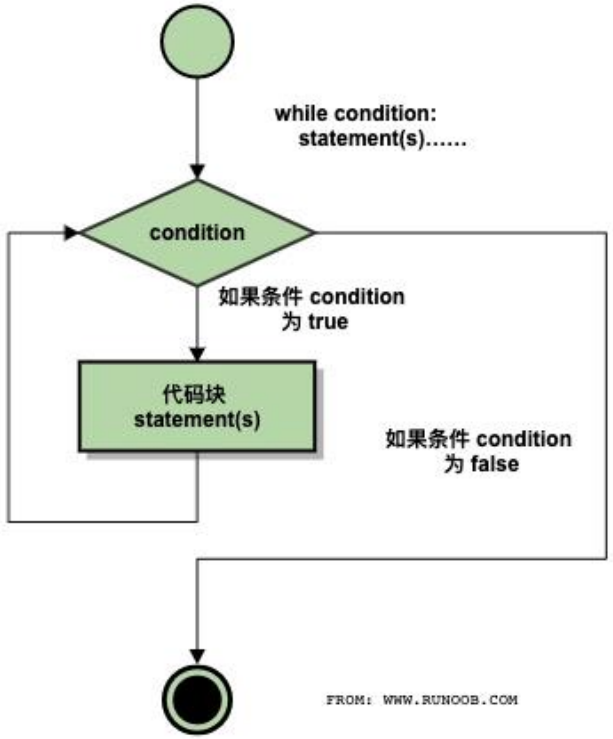
DATE: 5/3/2020

本次课程

- python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
- python函数及其参数
- return语句

回顾：while语句

- python中的循环语句有for和while
- while语句的执行流程图：



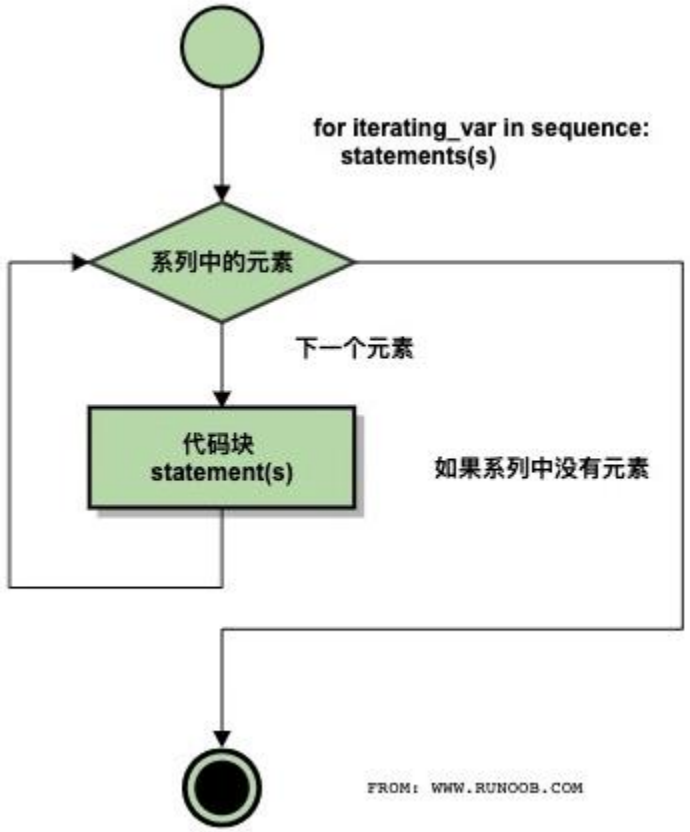
for语句

- Python for循环可以遍历任何序列的项目，如一个列表或者一个字符串。
- for循环的一般格式如下：

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

for语句

- for语句的执行流程图：



for语句

- Python for循环实例：

```
>>> languages = ["C", "C++", "Perl", "Python"]
>>> for x in languages:
        print (x)


C
C++
Perl
Python
>>>
```

range函数

- 如果你需要遍历数字序列，可以使用内置range()函数。它会生成数列，例如:

```
>>> for i in range(5):
        print(i)


0
1
2
3
4
>>>
```

range函数

- 你也可以使用range指定区间的值：

```
>>> for i in range(5,9):
        print(i)


5
6
7
8
>>>
```

range函数

- 也可以使range以指定数字开始并指定不同的增量(甚至可以是负数，有时这也叫做'步长'):

```
>>> for i in range(0, 10, 3):
        print(i)


0
3
6
9
>>>
```

range函数

- 负数：

```
>>> for i in range(-10, -100, -30):
        print(i)


-10
-40
-70
```

range函数

- 你可以结合range()和len()函数以遍历一个序列的索引,如下所示:

```
>>> a = ['Google', 'Baidu', 'Runoob', 'Taobao', 'QQ']
>>> for i in range(len(a)):
        print(i, a[i])


0 Google
1 Baidu
2 Runoob
3 Taobao
4 QQ
>>>
```

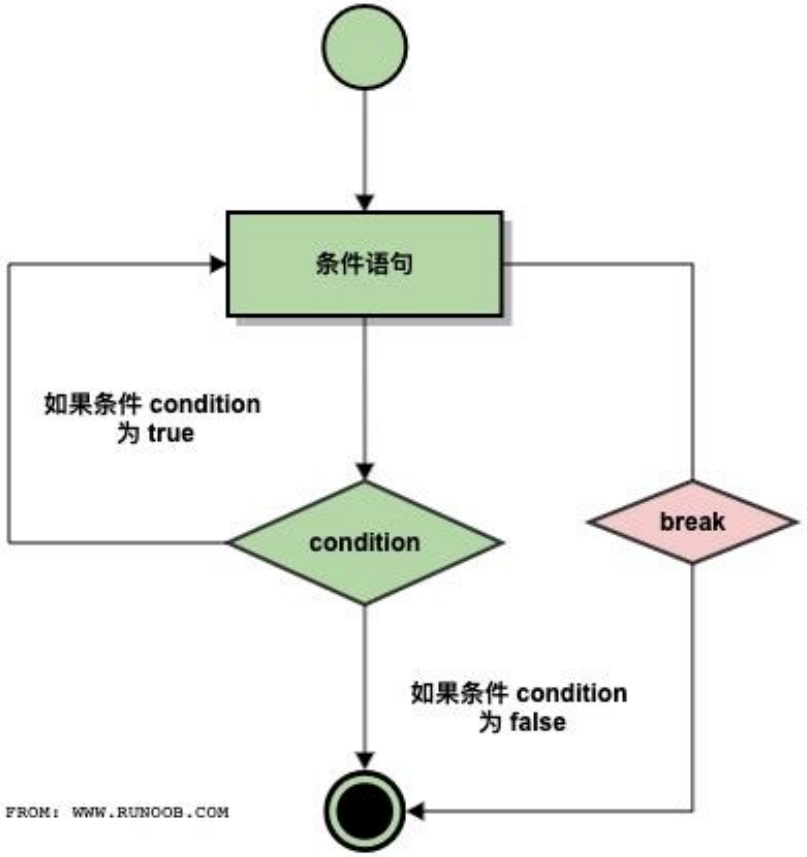
range函数

- 还可以使用range()函数来创建一个列表：

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>>
```

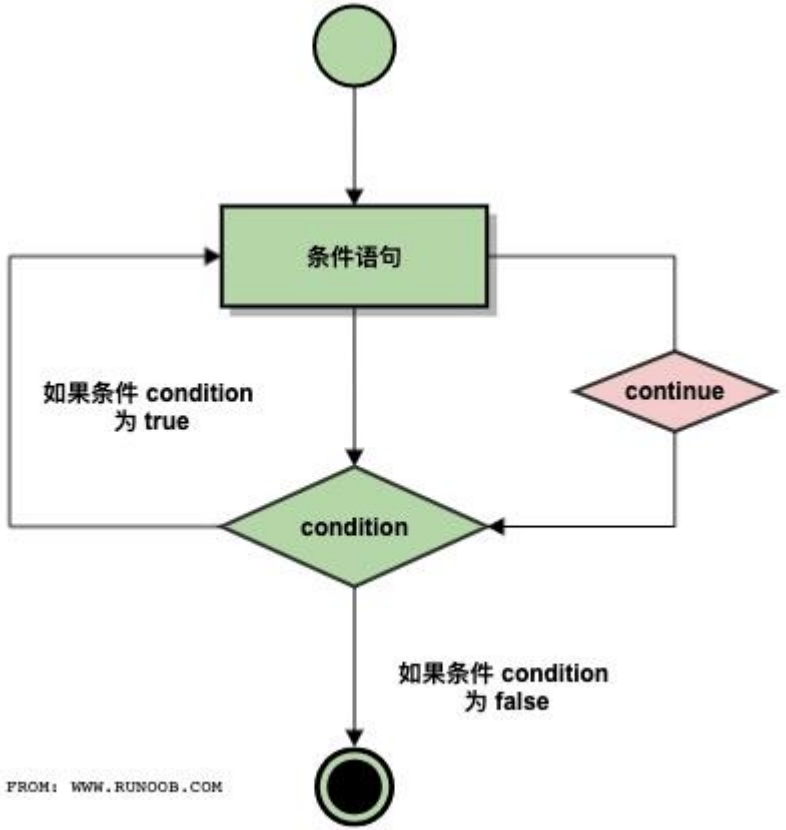
break和continue语句及循环中的else子句

- break语句执行流程图：



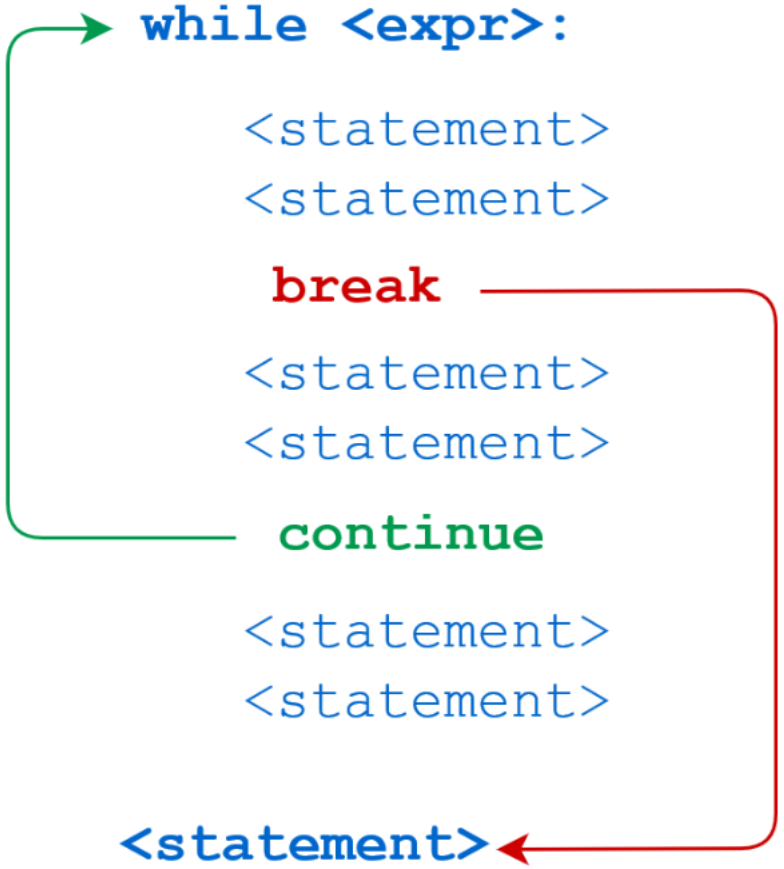
break和continue语句及循环中的else子句

- continue语句执行流程图：



break和continue语句及循环中的else子句

- 代码执行过程：



break和continue语句及循环中的else子句

- break语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行。
- continue语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

break和continue语句及循环中的else子句：实例

- while中使用 break：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('循环结束。')
```

- 输出结果为：

```
4
3
循环结束。
```

break和continue语句及循环中的else子句：实例

- while 中使用 continue：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('循环结束。')
```

- 输出结果为：

```
4
3
1
0
循环结束。
```

break和continue语句及循环中的else子句：实例

```
for letter in 'Runoob':     # 第一个实例
   if letter == 'b':
      break
   print ('当前字母为 :', letter)

var = 10                    # 第二个实例
while var > 0:
   print ('当期变量值为 :', var)
   var = var -1
   if var == 5:
      break

print ("Good bye!")
```

```
for letter in 'Runoob':     # 第一个实例
   if letter == 'o':        # 字母为 o 时跳过输出
      continue
   print ('当前字母 :', letter)

var = 10                    # 第二个实例
while var > 0:
   var = var -1
   if var == 5:             # 变量为 5 时跳过输出
      continue
   print ('当前变量值 :', var)
print ("Good bye!")
```

break和continue语句及循环中的else子句

- 循环语句可以有 else 子句，它在穷尽列表(以for循环)或条件变为 false (以while循环)导致循环终止时被执行，但循环被 break 终止时不执行。

break和continue语句及循环中的else子句：实例

- 如下实例用于查询质数的循环例子:

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, '等于', x, '*', n//x)
            break
    else:
        # 循环中没有找到元素
        print(n, ' 是质数')
```

- 执行以上脚本输出结果为：

```
2  是质数
3  是质数
4 等于 2 * 2
5  是质数
6 等于 2 * 3
7  是质数
8 等于 2 * 4
9 等于 3 * 3
```

pass语句

- Python pass是空语句，是为了保持程序结构的完整性。
- pass 不做任何事情，一般用做占位语句，如下实例

```
>>> while True:
        pass # 等待键盘中断(Ctrl+C)
```

- 最小的类:

```
>>> class MyEmpty:
        pass
```

pass语句：实例

- 以下实例在字母为 o 时 执行 pass 语句块:

```
for letter in 'Runoob':
   if letter == 'o':
      pass
      print ('执行 pass 块')
   print ('当前字母 :', letter)

print ("Good bye!")
```

- 执行以上脚本输出结果为:

```
当前字母 : R
当前字母 : u
当前字母 : n
执行 pass 块
当前字母 : o
执行 pass 块
当前字母 : o
当前字母 : b
Good bye!
```

Python函数

- 函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。
- 函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。但你也可以自己创建函数，这被叫做用户自定义函数。

定义函数

- 你可以定义一个由自己想要功能的函数，以下是简单的规则：
  - 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。
  - 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
  - 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
  - 函数内容以冒号起始，并且缩进。
  - **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

Python函数

- Python 定义函数使用 def 关键字，一般格式如下：

```
def 函数名（参数列表）：
    函数体
```

- 默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配起来的。

Python函数

- 让我们使用函数来输出"Hello World！"：

```
>>> def hello():
        print("Hello World!")


>>> hello()
Hello World!
>>>
```

Python函数

- 更复杂点的应用，函数中带上参数变量:

```
def area(width, height):
    return width * height

def print_welcome(name):
    print("Welcome", name)

print_welcome("Runoob")
w = 4
h = 5
print("width =", w, " height =", h, " area =", area(w, h))
```

- 以上实例输出结果：

```
Welcome Runoob
width = 4  height = 5  area = 20
```

函数调用

- 定义一个函数：给了函数一个名称，指定了函数里包含的参数，和代码块结构。
- 这个函数的基本结构完成以后，你可以通过另一个函数调用执行，也可以直接从 Python 命令提示符执行。

函数调用

- 如下实例调用了 **printme()** 函数：

```
# 定义函数
def printme( str ):
    # 打印任何传入的字符串
    print (str)
    return

# 调用函数
printme("我要调用用户自定义函数!")
printme("再次调用同一函数")
```

- 以上实例输出结果：

```
我要调用用户自定义函数!
再次调用同一函数
```

参数传递

- 在 python 中，类型属于对象，变量是没有类型的：

```
a=[1,2,3]

a="Runoob"
```

- 以上代码中，**[1,2,3]** 是 List 类型，**"Runoob"** 是 String 类型，而变量 a 是没有类型，她仅仅是一个对象的引用（一个指针），可以是指向 List 类型对象，也可以是指向 String 类型对象。

参数传递

可更改(mutable)与不可更改(immutable)对象

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- **不可变类型：**变量赋值 **a=5** 后再赋值 **a=10**，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变a的值，相当于新生成了a。
- **可变类型：**变量赋值 **la=[1,2,3,4]** 后再赋值 **la[2]=5** 则是将 list la 的第三个元素值更改，本身la没有动，只是其内部的一部分值被修改了。

参数传递

python 函数的参数传递

- **不可变类型：**类似 c++ 的值传递，如 整数、字符串、元组。如fun（a），传递的只是a的值，没有影响a对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。
- **可变类型：**类似 c++ 的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后fun外部的la也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象

参数传递

python 传不可变对象实例

```
def ChangeInt( a ):
    a = 10

b = 2
ChangeInt(b)
print( b ) # 结果是 2
```

- 实例中有 int 对象 2，指向它的变量是 b，在传递给 ChangeInt 函数时，按传值的方式复制了变量 b，a 和 b 都指向了同一个 Int 对象，在 a=10 时，则新生成一个 int 值对象 10，并让 a 指向它。

参数传递

传可变对象实例

可变对象在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。例如：

```
# 可写函数说明
def changeme( mylist ):
   "修改传入的列表"
   mylist.append([1,2,3,4])
   print ("函数内取值: ", mylist)
   return

# 调用changeme函数
mylist = [10,20,30]
changeme( mylist )
print ("函数外取值: ", mylist)
```

参数传递

传入函数的和在末尾添加新内容的对象用的是同一个引用。故输出结果如下：

```
函数内取值：  [10, 20, 30, [1, 2, 3, 4]]
函数外取值：  [10, 20, 30, [1, 2, 3, 4]]
```

参数

以下是调用函数时可使用的正式参数类型：

- 必需参数
- 关键字参数
- 默认参数
- 不定长参数

必需参数

- 必需参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。
- 调用 printme() 函数，你必须传入一个参数，不然会出现语法错误：

```
#可写函数说明
def printme( str ):
   "打印任何传入的字符串"
   print (str)
   return

# 调用 printme 函数，不加参数会报错
printme()
```

必需参数

- 以上实例输出结果：

```
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    printme()
TypeError: printme() missing 1 required positional argument: 'str'
```

关键字参数

- 关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。
- 使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

关键字参数

- 以下实例在函数 printme() 调用时使用参数名：

```
#可写函数说明
def printme( str ):
   "打印任何传入的字符串"
   print (str)
   return

#调用printme函数
printme( str = "菜鸟教程")
```

- 以上实例输出结果：

```
菜鸟教程
```

关键字参数

- 以下实例中演示了函数参数的使用不需要使用指定顺序：

```
#可写函数说明
def printinfo( name, age ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return

#调用printinfo函数
printinfo( age=50, name="runoob" )
```

- 以上实例输出结果：

```
名字:  runoob
年龄:  50
```

默认参数

- 调用函数时，如果没有传递参数，则会使用默认参数。以下实例中如果没有传入 age 参数，则使用默认值：

```
#可写函数说明
def printinfo( name, age = 35 ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return

#调用printinfo函数
printinfo( age=50, name="runoob" )
print ("------------------------")
printinfo( name="runoob" )
```

默认参数

- 以上实例输出结果：

```
名字：  runoob
年龄：  50
------------------------
名字：  runoob
年龄：  35
```

不定长参数

- 你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述 2 种参数不同，声明时不会命名。基本语法如下：

```
def functionname([formal_args,] *var_args_tuple ):
   "函数_文档字符串"
   function_suite
   return [expression]
```

不定长参数

- 加了星号 * 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数。

```
# 可写函数说明
def printinfo( arg1, *vartuple ):
    "打印任何传入的参数"
    print ("输出: ")
    print (arg1)
    print (vartuple)

# 调用printinfo 函数
printinfo( 70, 60, 50 )
```

- 以上实例输出结果：

```
输出:
70
(60, 50)
```

不定长参数

- 如果在函数调用时没有指定参数，它就是一个空元组。我们也可以不向函数传递未命名的变量。如下实例：

```
# 可写函数说明
def printinfo( arg1, *vartuple ):
   "打印任何传入的参数"
   print ("输出: ")
   print (arg1)
   for var in vartuple:
      print (var)
   return

# 调用printinfo 函数
printinfo( 10 )
printinfo( 70, 60, 50 )
```

不定长参数

- 以上实例输出结果：

```
输出：
10
输出：
70
60
50
```

匿名函数

- python 使用 lambda 来创建匿名函数。
- 所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数。
  - lambda 只是一个表达式，函数体比 def 简单很多。
  - lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
  - lambda 函数拥有自己的命名空间，且不能访问自己参数列表之外或全局命名空间里的参数。
  - 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

匿名函数

语法

- lambda 函数的语法只包含一个语句，如下：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

匿名函数

- 如下实例：

```
# 可写函数说明
sum = lambda arg1, arg2: arg1 + arg2

# 调用sum函数
print ("相加后的值为 : ", sum( 10, 20 ))
print ("相加后的值为 : ", sum( 20, 20 ))
```

- 以上实例输出结果：

```
相加后的值为 :  30
相加后的值为 :  40
```

return语句

- **return [表达式]** 语句用于退出函数，选择性地向调用方返回一个表达式。不带参数值的return语句返回None。之前的例子都没有示范如何返回数值，以下实例演示了 return 语句的用法：

```
# 可写函数说明
def sum( arg1, arg2 ):
   # 返回2个参数的和."
   total = arg1 + arg2
   print ("函数内 : ", total)
   return total

# 调用sum函数
total = sum( 10, 20 )
print ("函数外 : ", total)
```

return语句

- 以上实例输出结果：

```
函数内 :  30
函数外 :  30
```

接下来...

- python函数（深入）
- 迭代器与生成器
- python数据结构
- python模块
- 数据分析（二）
- ...