

## ★ 基本规则:

- Python use indentation to define code blocks (4 spaces as standard (other length still work))  
→ 缩进字符数必须相同. Python中换行意味着语句结束, 同行中用 ; 分割多个语句
- 若想在 python 中输入中文, 在文件首加入 "# -\*- coding: UTF-8 -\*-" 或 "# coding=utf-8" (注意空格位置)
- Python 中有一些关键字 (见右表)  
它们在代码中都具有特殊功能  
不能作为变量或常量名  
(注: 所有关键字只包含小写字母)
- The objects considered as empty  
会被判定为 False:
  - empty string: ""
  - empty list: []
  - number zero: 0
  - false boolean variable: False

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	with
def	finally	in	print	yield

## ★ 代表性实例

脚本文件一般用 .py 后缀

单行注释

导入其它代码模块

注意! Python 最好也最个性的语法: 使用缩进来代替其它语言块声明; 一般建议每个层级用4个空格来缩进。

变量得先实例化才可进一步计算

单行的语句块, 其实可以不换行的, 但是, 建议清晰起见, 规范点:

- 另起一行
- 缩进一级

函数声明, 注意使用冒号结束声明

多行注释的内容不用遵守当前缩进只要开始的''' 缩进正确就成!

每级语法块不用}之类的括号引领! 直接回车+4空格 (当然, 要在当前缩进基础上)

中文用户一定得先用这行来声明编码, 同时文件本身也得存储成UTF-8编码!

模块名, 其实导入了 os.py

函数名 'main' 在这儿并不是必须的, 调用在这段脚本的最后部分;

声明单行字符串, 使用双/单引号都成, 注意对字符串中的引号进行逃逸处理!

函数调用, 声明在后代码;

字符可乘, 等于: '\*\*\*\*\*'

调用了 os 模块中的函数

连接字符串

内置的列表类型对象, 其实可以包含不同类型数据, 甚至可以包含其它列表对象;

在循环中, i 指代了列表中按顺序的每个 "food"

range() 内置函数, 返回类似 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] 的数字列表, 注意 for in 循环语句使用冒号结束声明!

字符串的格式化输出基本类似C语言的

判定式也基本和C语言的相同

逻辑运算符, 不使用 && 和 ||, 使用直观的E文单词

用冒号来结束判断句, 在 if, elif else 行最后

这都是合法注释

一般在脚本最后调用主函数 main(); 而且使用 内置的运行脚本名来判定; 当且仅当我们直接运行当前脚本时, \_name\_ 才为 \_main\_ 这样当脚本被当作模块进行 import 导入时, 并不运行 main() 所以, 一般这里是进行测试代码安置的...

关于 UliPad 4.0

作者: Limodou (limodou@gmail.com)

如果你有任何问题请与我联系。

[The UliPad project homepage](#)

[The UliPad maillist](#)

[The UliPad Snippets Site](#)

[My Blog](#)

[Contact me](#)

确定

## ★ 基础函数

- 在界面展示 `print(" %s, %d " % (xx, xxx))` 另%r可表示r位浮点数. %x表示16进制数 注. 不换行用, 间隔
- 定义变量 整型: 直接赋值 `myint = 7` 同时符值 `a, b = 3, 4`  
浮点型: `myfloat = 7.0` 或 `myfloat = float(7)`  
字符串: `mystring = 'hello'` 或 `mystring = "hello"`  
例子: `hello = "hello"`  
`world = "world"`  
`helloworld = hello + " " + world`  
`print(helloworld)`  $\Rightarrow$  `hello world`  
注意, print里不要涉及多种变量

- List (similar to arrays) can contain any type of variable, no limits with quantity

`mylist []` 与 `mylist [1, 2]` 效果相同

`mylist.append(1)`

`mylist.append(2)`

注意 python中List也是从0开始数编号的

- Operators: %用于取余数. 如 `11 % 3` 结果为2  
双\*用于取次幂 如 `7 ** 2` 结果为49

对 string 型数据, "Ax" + "Bx" 可以将两串放在一起. "Ax" \* 10 可以重复10遍

对 List 型数据 + 与 \* 符号的用法同上

- Conditions: 最基本的包括 `>` `<` `==` `!=`, 返回值为 'True' 和 'False'

↳ 在其中使用 'and' 'or' 并列同时执行多个判断.

↳ 关键字 "in" 可以配合 if 检查某变量是否在某个组中

script.py	IPython Shell
<pre>1 name = "John" 2 if name in ["John", "Rick"]: 3     print("Your name is either John or Rick.")</pre>	<pre>Your name is either John or Rick. In [1]:  </pre>

↳ 关键字 if 搭配 关键字 elif, else 和 is, pass 使用构成 condition 逻辑

script.py	IPython Shell
<pre>1 statement = False 2 another_statement = True 3 if statement is True: 4     print(1) 5     pass 6 elif another_statement is True: # else if 7     print(2) 8     pass 9 else: 10    print(3) 11    pass</pre>	<pre>2 In [1]:  </pre>

↳ 关键字 is:

关注实例本身情况, 而非是否其中的变量相等  $\rightarrow$

↳ 关键字 not: 放在 boolean 值前使其反转: `not(True) == False`

```
1 x = [1, 2, 3]
2 y = [1, 2, 3]
3 print(x == y) # Prints out True
4 print(x is y) # Prints out False
```

- Loops 包含 <sup>[1]</sup> for 与 <sup>[2]</sup> while 两种 另配合 <sup>[3]</sup> 部分关键字使用, 注意循环关键字所在语句以 ":" 结束来包含其 block

<sup>[1]</sup> 使用 for 时常配合 in 调用数组或 List, 其后变量无需声明便可以从 0 计数, 除非配合关键字 range 使其特殊化:

script.py	例 1	IPython Shell
<pre>1 primes = [2, 3, 5, 7] 2 for prime in primes: 3     print(prime)</pre>		<pre>In [1]:  </pre>
script.py	例 2	IPython Shell
<pre>1 # Prints out the numbers 0,1,2,3,4 2 for x in range(5): 3     print(x) 4 print("\n") 5 # Prints out 3,4,5 6 for x in range(3, 6): 7     print(x) 8 print("\n") 9 # Prints out 3,5,7 10 for x in range(3, 8, 2): 11     print(x) 12 print("\n")</pre>		<pre>0 1 2 3 4 3 4 5 3 5 7</pre>



② 关键字 `while` 后加一个条件, 在条件为 `false` 前子代码会一直重复执行

```
script.py
1 # Prints out 0,1,2,3,4
2
3 count = 0
4 while count < 5:
5     print(count)
6     count += 1 # This is the same as count = count + 1

IPython Shell
0
1
2
3
4
```

③

```
script.py
1 # Prints out 0,1,2,3,4
2
3 count = 0
4 while True:
5     print(count)
6     count += 1
7     if count >= 5:
8         break
9     print("\n")
10 # Prints out only odd numbers - 1,3,5,7,9
11 for x in range(10):
12     # Check if x is even
13     if x % 2 == 0:
14         continue
15     print(x)

IPython Shell
0
1
2
3
4
1
3
5
7
9
In [1]:
```

与 `condition` 与语配合  
若 `True` 则跳出循环  
使光标回到 `loop` 外

与 `condition` 与语配合  
若 `True` 则立刻回到循环  
跳过了程序步骤

`loop` 中可以直接加用于 `condition` 的关键字 `else`

```
script.py
1 # Prints out 0,1,2,3,4 and then it prints "count value
  reached 5"
2
3 count=0
4 while(count<5):
5     print(count)
6     count +=1
7 else:
8     print("count value reached %d" %(count))
9
10 # Prints out 1,2,3,4
11 for i in range(1, 10):
12     if(i%5==0):
13         break
14     print(i)
15 else:
16     print("this is not printed because for loop is
    terminated because of break but not due to fail in
    condition")

IPython Shell
0
1
2
3
4
count value reached 5
1
2
3
4
In [1]:
```

## • Function

定义通过关键字 `def` + 函数名 + (输入值) 可以通过加关键字 `return` 设定返回值  
调用函数时直接 函数名 + (输入值) 及可

```
script.py
1 # Modify this function to return a list of strings as
  defined above
2 def list_benefits():
3     return "More organized code", "More readable code",
      "Easier code reuse", "Allowing programmers to share and
      connect code together"
4
5 # Modify this function to concatenate to each benefit - "
  is a benefit of functions!"
6 def build_sentence(benefit):
7     return "%s is a benefit of functions!" % benefit
8
9
10 def name_the_benefits_of_functions():
11     list_of_benefits = list_benefits()
12     for benefit in list_of_benefits:
13         print(build_sentence(benefit))
14
15 name_the_benefits_of_functions()

solution.py
1 # Modify this function to return a list of strings as
  defined above
2 def list_benefits():
3     return "More organized code", "More readable code",
      "Easier code reuse", "Allowing programmers to share and
      connect code together"
4
5 # Modify this function to concatenate to each benefit - "
  is a benefit of functions!"
6 def build_sentence(benefit):
7     return "%s is a benefit of functions!" % benefit
8
9
10 def name_the_benefits_of_functions():
11     list_of_benefits = list_benefits()
12     for benefit in list_of_benefits:
13         print(build_sentence(benefit))
14
15 name_the_benefits_of_functions()

IPython Shell
<script.py> output:
More organized code is a benefit of functions!
More readable code is a benefit of functions!
Easier code reuse is a benefit of functions!
Allowing programmers to share and connect code together is
a benefit of functions!

In [1]:
```

返回值用, 分隔

这个变量内存储的是 % 如果有两个, 用 % (A, B)

## • Classes 一种封装其它多种数据的数据类型

```
# define the Vehicle class
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." %
            (self.name, self.color, self.kind, self.value)
        return desc_str

car1 = Vehicle()
car2 = Vehicle()

car1.name = "Fer"
car1.kind = "convertible"
car1.color = "red"
car1.value = 60000.00

car2.name = "Jump"
car2.kind = "van"
car2.color = "blue"
car2.value = 10000.00

# test code
print(car1.description())
print(car2.description())
```

封装可包含任何数据类型  
以及函数

注意, string 型赋值要  
加 ""

调用类内容时加上类名. 前缀  
即可

• *Dictionaries* : 可以是任何类型值, 可直接访问来赋值. 含一个对应关系.

声明与赋值:

```
1 phonebook = {}  
2 phonebook["John"] = 938477566  
3 phonebook["Jack"] = 938377264  
4 phonebook["Jill"] = 947662781  
5 print(phonebook)
```

或

```
1 phonebook = {  
2     "John" : 938477566,  
3     "Jack" : 938377264,  
4     "Jill" : 947662781  
5 }  
6 print(phonebook)
```

Loop时应使用一对值做为count:

```
1 phonebook = {"John" : 938477566, "Jack" : 938377264, "Jill"  
2 : 947662781}  
3 for name, number in phonebook.items():  
4     print("Phone number of %s is %d" % (name, number))
```

使用关键字 `del` 或 `.pop` 的形式删除 *dictionary* 中部分值

```
1 phonebook = {  
2     "John" : 938477566,  
3     "Jack" : 938377264,  
4     "Jill" : 947662781  
5 }  
6 del phonebook["John"]  
7 print(phonebook)
```

```
1 phonebook = {  
2     "John" : 938477566,  
3     "Jack" : 938377264,  
4     "Jill" : 947662781  
5 }  
6 phonebook.pop("John")  
7 print(phonebook)
```