Python3 基础语法

编码

默认情况下，Python 3 源码文件以 **UTF-8** 编码，所有字符串都是 unicode 字符串。 当然你也可以为源码文件指定不同的编码：

```
# -*- coding: cp-1252 -*-
```

上述定义允许在源文件中使用 Windows-1252 字符集中的字符编码，对应适合语言为保加利亚语、白罗斯语、马其顿语、俄语、塞尔维亚语。

标识符

- 第一个字符必须是字母表中字母或下划线 `_` 。
- 标识符的其他的部分由字母、数字和下划线组成。
- 标识符对大小写敏感。

在 Python 3 中，非 ASCII 标识符也是允许的了。

python 保留字

保留字即关键字，我们不能把它们用作任何标识符名称。Python 的标准库提供了一个 keyword 模块，可以输出当前版本的所有关键字：

```
>>> import keyword>>> keyword.kwlist['False', 'None', 'True', 'and', 'as', 'assert',
 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', '
for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'o
r', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

注释

Python 中单行注释以 **#** 开头，实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3 # 第一个注释 print ("Hello, Python!") # 第二个注释
```

执行以上代码，输出结果为：

```
Hello, Python!
```

多行注释可以用多个 # 号，还有 ''' 和 """：

实例(Python 3.0+)

```
#!/usr/bin/python3 # 第一个注释 # 第二个注释 ''' 第三注释 第四注释 ''' """ 第五注释 第六注释 """ print ("Hello, Python!")
```

执行以上代码，输出结果为：

```
Hello, Python!
```

行与缩进

python 最具特色的就是使用缩进来表示代码块，不需要使用大括号 {} 。

缩进的空格数是可变的，但是同一个代码块的语句必须包含相同的缩进空格数。实例如下：

实例(Python 3.0+)

```
if True: print ("True") else: print ("False")
```

以下代码最后一行语句缩进数的空格数不一致，会导致运行错误：

```
if True:
    print ("Answer")
    print ("True")else:
    print ("Answer")
  print ("False")    # 缩进不一致，会导致运行错误
```

以上程序由于缩进不一致，执行后会出现类似以下错误：

```
 File "test.py", line 6
    print ("False")    # 缩进不一致，会导致运行错误
                                    ^IndentationError: unindent does not match an
y outer indentation level
```

多行语句

Python 通常是一行写完一条语句，但如果语句很长，我们可以使用反斜杠(\)来实现多行语句，例如：

```
total = item_one + \
        item_two + \
        item_three
```

在 [], {}, 或 () 中的多行语句，不需要使用反斜杠(\)，例如：

```
total = ['item_one', 'item_two', 'item_three',
        'item_four', 'item_five']
```

数字(Number)类型

python 中数字有四种类型：整数、布尔型、浮点数和复数。

- **int** (整数), 如 1, 只有一种整数类型 int，表示为长整型，没有 python2 中的 Long。
- **bool** (布尔), 如 True。
- **float** (浮点数), 如 1.23、3E-2
- **complex** (复数), 如 1 + 2j、 1.1 + 2.2j

字符串(String)

- python 中单引号和双引号使用完全相同。
- 使用三引号('''或""")可以指定一个多行字符串。
- 转义符 '\'
- 反斜杠可以用来转义，使用 r 可以让反斜杠不发生转义。。 如 r"this is a line with \n" 则\n 会显示，并不是换行。
- 按字面意义级联字符串，如"this " "is " "string"会被自动转换为 this is string。
- 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。

- Python 没有单独的字符类型，一个字符就是长度为 1 的字符串。
- 字符串的截取的语法格式如下：**变量[头下标:尾下标]**

```
word = '字符串'

sentence = "这是一个句子。"

paragraph = """这是一个段落，

可以由多行组成"""
```

实例(Python 3.0+)

```
#!/usr/bin/python3 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 print(str + '你好') # 连接字符串 print('------------------------------') print('hello\nrunoob') # 使用反斜杠(\)+n 转义特殊字符 print(r'hello\nrunoob') # 在字符串前面添加一个 r，表示原始字符串，不会发生转义
```

输出结果为：

```
RunoobRunoo

R

noo

noobRunoobRunoobRunoob 你好------------------------------

hello

runoob

hello\nrunoob
```

空行

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行并不是 Python 语法的一部分。书写时不插入空行，Python 解释器运行也不会出错。但是空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

记住：空行也是程序代码的一部分。

等待用户输入

执行下面的程序在按回车键后就会等待用户输入：

实例(Python 3.0+)

```
#!/usr/bin/python3 input("\n\n按下 enter 键后退出。")
```

以上代码中，"\n\n"在结果输出前会输出两个新的空行。一旦用户按下 enter 键时，程序将退出。

同一行显示多条语句

Python 可以在同一行中使用多条语句，语句之间使用分号(;)分割，以下是一个简单的实例：

实例(Python 3.0+)

```
#!/usr/bin/python3 import sys; x = 'runoob'; sys.stdout.write(x + '\n')
```

使用脚本执行以上代码，输出结果为：

```
runoob
```

使用交互式命令行执行，输出结果为：

```
>>> import sys; x = 'runoob'; sys.stdout.write(x + '\n')

runoob7
```

多个语句构成代码组

缩进相同的一组语句构成一个代码块，我们称之代码组。

像 if、while、def 和 class 这样的复合语句，首行以关键字开始，以冒号(:)结束，该行之后的一行或多行代码构成代码组。

我们将首行及后面的代码组称为一个子句(clause)。

如下实例：

```
if expression :

   suiteelif expression :

   suite else :

   suite
```

Print 输出

print 默认输出是换行的，如果要实现不换行需要在变量末尾加上 **end=""**：

实例(Python 3.0+)

```
#!/usr/bin/python3 x="a" y="b" # 换行输出 print( x ) print( y ) print('---------') # 不换行输出 print( x, end=" " ) print( y, end=" " ) print()
```

以上实例执行结果为：

```
a

b---------

a b
```

import 与 from...import

在 python 用 **import** 或者 **from...import** 来导入相应的模块。

将整个模块(somemodule)导入，格式为： **import somemodule**

从某个模块中导入某个函数,格式为： **from somemodule import somefunction**

从某个模块中导入多个函数,格式为： **from somemodule import firstfunc, secondfunc, thirdfunc**

将某个模块中的全部函数导入，格式为： **from somemodule import ***

导入 sys 模块

```
import sys print('================Python import mode==========================='); print ('命令行参数为:') for i in sys.argv: print (i) print ('\n python 路径为',sys.path)
```

导入 sys 模块的 argv,path 成员

```
from sys import argv,path # 导入特定的成员 print('================python from import===================================') print('path:',path) # 因为已经导入 path 成员，所以此处引用时不需要加 sys.path
```

命令行参数

很多程序可以执行一些操作来查看一些基本信息，Python 可以使用-h 参数查看各参数帮助信息：

```
$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...Options and arguments (and corresponding environment variables):-c cmd : program passed in as string (terminates option list)-d     : debug output from parser (also PYTHONDEBUG=x)-E
```

```
  : ignore environment variables (such as PYTHONPATH)-h     : print this help messa
ge and exit

[ etc. ]
```

我们在使用脚本形式执行 Python 时，可以接收命令行输入的参数，具体使用可以参照 Python 3 命令行参数。