Python3 中文手册

一、开胃菜

1.Python 易于使用，是一门完整的编程语言；与 Shell 脚本或批处理文件相比，它为编写大型程序提供了更多的结构和支持。另一方面，Python 提供了比 C 更多的错误检查，并且作为一门 高级语言，它内置支持高级的数据结构类型，例如：灵活的数组和字典。因其更多的通用数据类型，Python 比 Awk 甚至 Perl 都适用于更多问题领域，至少大多数事情在 Python 中与其他语言同样简单。

2.Python 允许你将程序分割为不同的模块，以便在其他的 Python 程序中重用。Python 内置提供了大量的标准模块，你可以将其用作程序的基础，或者作为学习 Python 编程的示例。这些模块提供了诸如文件 I/O、系统调用、Socket 支持，甚至类似 Tk 的用户图形界面（GUI）工具包接口。

3.Python 是一门解释型语言，因为无需编译和链接，你可以在程序开发中节省宝贵的时间。Python 解释器可以交互的使用，这使得试验语言的特性、编写临时程序或在自底向上的程序开发中测试方法非常容易。你甚至还可以把它当做一个桌面计算器。

4.Python 让程序编写的紧凑和可读。用 Python 编写的程序通常比同样的 C、C++ 或 Java 程序更短小，这是因为以下几个原因:

（1）高级数据结构使你可以在一条语句中表达复杂的操作;

（2）语句组使用缩进代替开始和结束大括号来组织;

（3）变量或参数无需声明。

5.Python 是 可扩展 的：如果你会 C 语言编程便可以轻易地为解释器添加内置函数或模块，或者为了对性能瓶颈作优化，或者将 Python 程序与只有二进制形式的库（比如某个专业的商业图形库）连接起来。一旦你真正掌握了它，你可以将 Python 解释器集成进某个 C 应用程序，并把它当作那个程序的扩展或命令行语言。

二、使用Python解释器

1.调用Python解释器

启动Python解释器:

（1）Python 解释器通常被安装在目标机器的 /usr/local/bin/python3.5 目录下。将 /usr/local/bin 目录包含进 Unix shell 的搜索路径里，以确保可以通过输入:python3.5启动Python解释器

Unix系统：Control+D结束，让解释器以0码退出

Python 解释器具有简单的行编辑功能。在 Unix 系统上，任何 Python 解释器都可能已经添加了 GNU readline 库支持，这样就具备了精巧的交互编辑和历史记录等功能。在 Python 主窗口中输入 Control-P 可能是检查是否支持命令行编辑的最简单的方法。

Python 解释器有些操作类似 Unix shell：当使用终端设备（tty）作为标准输入调用时，它交互的解释并执行命令；当使用文件名参数或以文件作为标准输入调用时，它读取文件并将文件作为脚本执行。

（2）命令行执行pyhton -c command [arg] ...

一般建议命令要用单引号包裹起来

有一些 Python 模块也可以当作脚本使用。你可以使用 python -m module [arg] ... 命令调用它们，这类似在命令行中键入完整的路径名执行 模块 源文件一样。

使用脚本文件时，经常会运行脚本然后进入交互模式。这也可以通过在脚本之前加上 -i 参数来实现。

(1)参数传递

调用解释器时，脚本名和附加参数传入一个名为 sys.argv 的字符串列表。你能够获取这个列表通过执行 import sys，列表的长度大于等于1；没有给定脚本和参数时，它至少也有一个元素:

1）sys.argv[0] 此时为空字符串);

2）脚本名指定为 '-' （表示标准输入）时， sys.argv[0] 被设定为 '-');

3）使用 -c 指令 时，sys.argv[0] 被设定为 '-c');

4）使用 -m 模块 参数时，sys.argv[0] 被设定为指定模块的全名)

-c 指令 或者 -m 模块 之后的参数不会被 Python 解释器的选项处理机制所截获，而是留在 sys.argv 中，供脚本命令操作。

(2)交互模式

解释器工作于交互模式：从tty读取命令时，根据主提示符来执行，主提示符：（>>>）；继续的部分为从属提示符（...）。

输入多行语句用从属提示符，例如下面的if语句：

```
the world is flat=1
if the world is flat:
    print("Be careful not to fall off!")
```

2.解释器及其环境

(1)源程序编码

默认情况下，Python源文件是UTF-8编码

在首行后插入至少一行特殊的注释行来定义源文件的编码

```
# -*- coding:encoding -*-
```

三、Python简介

表示注释，但是注释不能出现在字符串中

注意：在练习中遇到的从属提示符表示你需要在最后多输入一个空行，解释器才能知道这是一个多行命令的结束

```
    # this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

1.将Python当做计算器

(1)数字

1）int：整数,如2,4,6,10;

```
>>> 2+2
4
>>> 50-5*6
20
>>> 8/5
1.6
```

2）float：小数，如5.0,1.6,2.7等
3）除法“/”永远返回一个浮点数
4）如果使用floor除法并且得到整数结果（丢掉任何小数部分），可以使用//运算符;
5）余数：%运算符

```
>>> 17/3
5.666666666666667
>>> 17//3
5
>>> 17%3
2
>>> 5*3+2
17
```

6）乘方：**

```
>>> 5**2
25
>>> 2**7
128
```

7）变量赋值：=

```
>>> width=20
>>> height=5*9
>>> width*height
900
```

变量在使用前必须“定义（赋值）”，否则会出错

```
>>> # try to access an undefined variable
... n
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: name 'n' is not defined
```

8）浮点数有完整的支持，整数和浮点数的混合计算中，整数会被转换为浮点数

```
>>> 3*3.75/1.5
7.5
>>> 7.0/2
3.5
```

9）交互模式下，最近一个表达式的赋值给变量_。这样我们可以把它当作一个桌面计算器，很方便的用于连续计算，例如

```
>>> tax=12.5/100
>>> price=100.50
>>> price*tax
12.5625
>>> price+_
113.0625
>>> round(_,2)
113.06
```

此变量对于用户是只读的，不要尝试给它一一赋值，你只会创建一个独立的同名局部变量，它屏蔽了系统内置变量的魔术效果。 10）Python支持其他数字类型，例如Decimal和Fraction，Python还内建了支持复数，使用后缀j或者J表示虚数部分（例如：3+5j）

(2)字符串

1）字符串用'...'或者"..."来标识
"\"可以用来做转义字符

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

2）print()函数可以生成可读性更好的输出，它会省去引号并且打印出转义字符后的特殊字符

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

3）如果前面带有\的字符被当做特殊字符，可以用原始字符串，在第一个引号的前面加上r：

```
>>> print('C:\some\name')   # here \n means newline!换行了
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

4）多行字符串：”””...””” 或者”’...”’
行尾换行符会自动被包含到字符串中，但是可以在行尾加上\来避免这个行为

```
print("""\
Usage: thingy [OPTIONS]
     -h
     -H hostname
""")
```

5）字符串可以由+操作符连接，可以由*表示重复

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

相邻的两个字符串文本自动连接在一起

```
>>> 'Py' 'thon'
'Python'
```

但是这只用于两个字符串文本，不能用于字符串表达式：

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string literal
  ...
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  ...
SyntaxError: invalid syntax
```

6）连接多个变量或者连接一个变量和一个字符串文本，使用 +：

```
>>> prefix = 'Py'
>>> prefix + 'thon'
'Python'
```

在很想切分很长的字符串时很有用：

```
>>> text = ('Put several strings within parentheses '
            'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

7）字符串也可以被截取(检索)。类似于 C ，字符串的第一个字符索引为 0 。Python没有单独的字符类型；一个字符就是一个简单的长度为1的字符串。:

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

索引也可以是负数，这将导致从右边开始计算。例如:

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

注意：-0实际上就是0，所以它不会导致从右边开始计算 8）支持切片：索引用于获得单个字符，切片 让你获得一个子字符串:

```
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

注意：包含起始的字符，不包含末尾的字符。这使得:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

切片的索引有非常有用的默认值；省略的第一个索引默认为零，省略的第二个索引默认为切片的字符串的大小。:

```
>>> word[:2]  # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]  # characters from position 4 (included) to the end
'on'
>>> word[-2:] # characters from the second-last (included) to the end
'on'
```

记住切片的工作方式：切片时的索引是在两个字符 之间 。左边第一个字符的索引为 0，而长度为 n 的字符串其最后一个字符的右界索引为 n。例如:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

文本中的第一行数字给出字符串中的索引点 0...6。第二行给出相应的负索引。切片是从 i 到 j 两个数值标示的边界之间的所有字符。

对于非负索引，如果上下都在边界内，切片长度就是两个索引之差。例如，word[1:3] 是 2 。

试图使用太大的索引会导致错误:

```
 >>> word[42]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

9）Python可以优雅的处理那些没有意义的切片索引：一个过大的索引值(即下标值大于字符串实际长度)将被字符串实际长度所代替，当上边界比下边界大时(即切片左值大于右值)就返回空字符串:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

10）Python的字符串不可以被更改，它们是不可变的，因此，赋值给字符串索引的位置会导致错误:

```
>>> word[0] = 'J'
  ...
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
  ...
TypeError: 'str' object does not support item assignment
```

如果需要一个不同的字符串，应该建一个新的:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

11）内置函数len()返回字符串长度

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

(3)列表

1）Python有几个复合数据类型，用来表示其他的值，最常用的是list列表：中括号表示，列表的元素不必是同一类型:

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

2）就像字符串（以及其他所有内建的序列类型一样），列表可以被索引和切片：

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0]    # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

3）所有的切片操作都会返回一个包含请求元素的新列表，这意味着下面的切片操作会返回一个新的（浅）拷贝副本：

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0]    # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
>>> squares[:]
[1, 4, 9, 16, 25]
```

4）列表也支持连接这样的操作：

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

5）不像不可变的字符串，列表是可变的，它允许修改元素：

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

6）使用 append() 方法 （后面我们会看到更多关于列表的方法的内容）在列表的末尾添加新的元素：

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

7）也可以对切片赋值，此操作可以改变列表的尺寸，或清空它:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

8）内置函数 len() 同样适用于列表:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

9)允许嵌套列表(创建一个包含其它列表的列表)，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'
```

2.编程的第一步

写一个生成菲波那契子序列的程序:

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
...
```

```
1
1
2
3
5
8
```

这个例子出现的新功能:

（1）第一行: 多重赋值：变量 a 和 b 同时获得了新的值 0 和 1 ,最后一行又使用了一次。变量赋值前，右边首先完成计算，右边的表达式从左到右计算。

（2）条件（这里是 b ¡ 10 ）为 true 时， while 循环执行。在 Python 中，类似于 C，任何非零整数都是 true；0 是 false。条件也可以是字符串或列表，实际上可以是任何序列;

（3）所有长度不为零的是 true，空序列是 false。示例中的测试是一个简单的比较。标准比较操作符与 C 相同： < ， > ， == ， <=， >= 和 ! =。

（4）循环 体 是 缩进 的：缩进是 Python 组织语句的方法。Python (还)不提供集成的行编辑功能，所以你要为每一个缩进行输入 TAB 或空格，一般是4个空格;

（5）大多数文本编辑器提供自动缩进。交互式录入复合语句时，必须在最后输入一个空行来标识结束（因为解释器没办法猜测你输入的哪一行是最后一行），需要注意的是同一个语句块中的每一行必须缩进同样数量的空白。

（6）关键字 print() 语句输出给定表达式的值。它控制多个表达式和字符串输出为你想要字符串（就像我们在前面计算器的例子中那样）。

（7）字符串打印时不用引号包围，每两个子项之间插入空间，所以你可以把格式弄得很漂亮，像这样:

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

（8）用一个逗号结尾就可以禁止输出换行:

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print(b, end=',')
...     a, b = b, a+b
...
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

补充:

（1）因为 ** 的优先级高于 -，所以 -3**2 将解释为 -(3**2) 且结果为 -9。为了避免这点并得到 9，你可以使用 (-3)**2。

（2）与其它语言不同，特殊字符例如 \n 在单引号('...')和双引号("...")中具有相同的含义。两者唯一的区别是在单引号中，你不需要转义 "（但你必须转义 '\')，反之亦然。

四、深入Python流程控制

1.if语句

2.for语句

3.range()函数

4.break和continue语句，以及循环中的else子句

5.pass语句

6.定义函数

7.深入Python函数定义

(1)默认参数值

(2)关键字参数

(3)可变参数列表

(4)参数列表的分析

(5)Lambda形式

(6)文档字符串

(7)函数注解

8.插曲：编码风格