Python

Python 是一个计算机语言的名称，它只是一种计算机语言。

- 使用起来比较简单，但它也是一个真正的计算机语言，相比一些简单的脚本语言来说，它能够为大型程序
  提供更多结构和支持。
- 相比C语言，能够提供更多的错误检验
- 它被作为一种非常高级的语言，内置高级的数据类型，例如数组和字典；而且因为具有更通用的数据类型，
  可以用于更多的问题处理种

Python支持将程序划分为模块（Module），这样能够复用。而且Python自己就附带了大量的标准库模块，
比如文件的I/O，系统功能调用，sockets（网络底层编程），而且也能够使用图形用户界面。

Python是一种解释性语言，它不需要编译（Compile）和连接过程（Link），因此可以节省编写时间。
解释器能够交互方式运行，因此可以在编写过程中进行验证确认。

相比一些底层编程语言，Python能够写出更加简短，易读的代码，相似的功能，使用Python可以比C，C++
或者Java语言更加简短。

- 高级语言可以使用简单的表达式完成更加复杂的操作
- Python利用语句的缩进来识别语句的分组
- 不需要明确的变量和参数声明（定义）

通过使用C，还能够为Python编写新的内建函数和扩展模块，这样既可以最快速度（高效率）执行一些关键操作，

也可以利用这种方式将一些希望使用的库提供给Python使用。

运行

通过执行python，可以进入到Python解释器的交互环境，环境每一行默认是“>>>”开始。在这里写入
Python语句就可以执行了。

非正式介绍

在解释器运行时，在一行之前是否有提示符（“>>>”和“...”）是不同的，在提示符后，应当输入完整的全部
内容。解释器的输出是没有提示符显示的。当有辅助提示符的多行输入情况下，你需要键入一个空行，最后多
一次回车换行以结束所有的输入。

输入内容如果有“#”，在这之后的内容表示注释，并且注释一直到本行结束为止。#如果出现在Python代码的
字符串表达式中，那么就不是注释了，请看下面的例子

```
# 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."
```

计算器（数值）

可以把Python解释器当作一个计算器直接使用，通过键入数字和对应的操

作符，就可以让解释器告诉你结果

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

上面的几个例子中，输入的都是整数，这在Python语言中定义为`int`类型，如果输入有小数点的数字，
那么他们被定义为浮点数类型`float`，除法`/`总是返回浮点数结果。但是如果我们就是想得到
最接近的整数结果该怎么办？这个时候可以使用`//`运算符，如果想得到余数，可以使用`%`运算符。

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the div
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

使用`**`运算符可以计算幂：

```
>>>
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

等号`=`用来给一个变量赋值，赋值操作在解释器中没有输出：

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

如果一个变量没有定义（也没有给它赋值），直接使用这个变量会报错：

```
>>>
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

浮点数的运算可以支持自动的类型转换，解释器会把整型数自动转换为浮点数：

```
>>>
>>> 4 * 3.75 - 1
14.0
```

在解释器的交互模式下，有一个特殊的变量`_`，最后一个表达式的返回结果会被赋值在这个变量中：

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

最好把这个变量当成一个只读变量，不要尝试给它赋值。

在Python中，除了`int`和`float`类型的数值，还有`Decimal` `Fraction`等，另外还有内建对复数的支持。

字符串

除了数值，Python也提供处理字符串的能力。字符串在Python中有几种表示方式。使用单引号（'...'）
或者双引号（"..."）都能得到相同的字符串。字符串当中使用`\`可以用来输入引号字符：

```
>>>
>>> '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," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

在交互解释器当中，输出的字符串会被放在引号当中显示，其中特殊字符会斜线 `\` 转义表示。采用这样
的表达方式，两个相同的字符串在显示时可能会不太相同，但是他们实际上是一样的。如果在字符串中存在
单引号并且没有双引号，字符串会放在双引号中，否则就会放在单引号当中。如果使用 `print()` 函数
会输出更方便人读取的形式，不会用引号，直接输出转义和特殊字符：

```
>>>
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they 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.
```

如果你不希望字符串中的斜线字符 `\` 被当作转义符号，你可以在字符串的第一个引号前添加 `r`
将字符串作为原始字符串看待：

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

字符串文字可以跨越多行。一个方法是使用三引号 `"""..."""` 或者 `'''...'''` 。这样三引号
之中所有的内容都被看作字符串的一部分，包括其中的换行等。可以通过在在行末尾添加 `\` 阻止换行被

包含，示例：

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

上例会输出下面的内容，注意其中第一行的换行并没有包含在内：

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

字符串可以通过`+`运算符被连接在一起，可以通过`*`运算符重复：

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

两个或者多个字符串顺序在一起时，也可以自动被连接成一个字符串。

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

这个特性可以在你期望阻止出现一个很长的字符串时有用：

```
>>>
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined t
```

但是这种方式仅仅对两个字符串文字有效，如果是变量或者表达式是没有作用的：

```
>>>
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a strin
  File "<stdin>", line 1
    prefix 'thon'
                ^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
                   ^
SyntaxError: invalid syntax
```

要连接变量或者表达式，使用计算符 `+`：

```
>>>
>>> prefix + 'thon'
'Python'
```

字符串可以使用索引（下标）的方式读取字符，第一个字符的索引是0。没有独立的字符类型，一个字符就是一个长度为1的字符串：

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

下标索引可以是负数，负数表示从字符串的结尾往前计数（从右向左数）：

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

注意因为-0等于0，因此负数索引从-1开始。

除了索引，Python也支持切片。所以获得一个独立的字符，而切片可以获得一个子字符串：

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

注意切片表示中，开始的索引是被包含的，而结尾索引是去除（不包含的）。这样做可以让 `s[:i]+s[i:]` 总是等于s：

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

切片索引有非常有用的默认值，第一个索引不填写默认为0，第二个索引不填写默认为字符串的长度：

```
>>>
>>> word[:2]   # character from the beginning to position 2 (
'Py'
>>> word[4:]   # characters from position 4 (included) to the
'on'
>>> word[-2:]  # characters from the second-last (included) t
'on'
```

记住切片索引如何处理的一个方法是将索引当作字符串之间的点，最左边的边界位置是0，最右边最后一个字符之后的位置是长度n：

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

上表中第一行给的是索引 `0...6` 的位置，第二行是对应负值索引的位置。从i到j的切片就是在边界位置i和j之间的所有字符：

对于非负数索引，当两者都在边界内时，切片长度等于索引的差，例如 `word[1:3]` 的长度为2。

尝试使用一个大于字符串长度的索引会导致错误：

```
>>>
>>> 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
```

但是，在切片中，如果尝试使用一个大于长度的索引会比较友好的处理：

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

Python中的字符串是不能改变的，他们是不可变量。因此尝试给一个字符串位置赋值会报错：

```
>>>
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

如果你需要一个不同的字符串，你只能去创建一个新的字符串：

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

内建的函数`len()`可以返回一个字符串的长度：

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

Lists（列表）

Python中有很多复合数据类型，可以将不同的数据组合放在一起。其中最常用的是列表（List），它可以用
方括号内使用逗号分割值的方式表示。列表可以包含不同类型的数据项，但是通常都使用相同的类型。

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

类似字符串，还有其他内建的顺序类型，列表能够被索引和分片：

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

所有的分片操作都会返回一个新的包含请求元素的列表。这样意味着使用下面的分片会返回一个列表的复制：

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

列表也支持连接操作：

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

和字符串是不可变量不同，列表是可变类型，因此我们可以去改变列表中

的内容：

```
>>>
>>> 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]
```

你可以使用`append()`在列表末尾增加新的项（后面还会看到更多的方法）：

```
>>>
>>> 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]
```

给切片赋值也是可以的，这将会改变列表长度甚至清空：

```
>>>
>>> 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 em
>>> letters[:] = []
>>> letters
[]
```

内建方法 `len()` 也支持列表：

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

嵌套列表也是可行的，在其他的列表中包含列表，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'
```

走向编程的第一步

我们可以使用Python去编写更加复杂的程序，完成很复杂的工作。下面，我们首先编写一个初步的斐波那契数列的计算过程：

```
>>>
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

这个例子引入这样几个有意思的特性：

- 第一行的赋值是一个多项赋值：变量a和b同时获得了他们的值，分别是0和1。而在最后一行，
  这个多项赋值再次使用，这次变的复杂了一点，变量计算后赋值给变量，这里要注意的是，表达式首先需要
  完成计算，优先处理赋值语句右侧的内容，而在赋值运算符右侧的内容中，计算顺序是从左向右完成，然后
  按照赋值顺序分别给两个变量对应赋值。
- `while`是一个循环执行的控制语句，这里用一个比较条件表达式：`a<10`，当条件成立的
  时候，这个控制语句继续执行，一直到条件不成立，则跳到下一句执行。比较条件表达式有：`<`小于，

`>` 大于， `==` 等于， `<=` 小于等于， `>=` 大于等于， `!=` 不等于。

- 循环控制中的内容都有行缩进（空格）： 缩进是Python中用来对表达式进行分组的方法。在交互模式下，
  你需要在缩进行输入 `tab` 或者空格。如果使用文本编辑软件，通常都会有自动的缩进功能来完成这些
  行的缩进输入。当一个组合表达式交互模式下输入完成后，需要在后面输入一个空行代表这个表达式块完成输入。
  另外要注意一点是这个表达式组合块中的每一行的缩进应该是相同的。
- `print()` 方法输出参数的数值。利用这个方法可以帮助我们输出需要看到的有格式的数据，
  它的作用很大：

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

i的值是 `65536` 。
函数当中可以使用关键字 `end` ，它能在输出后避免换行，或者需要在结尾使用不同的字符串结束输出

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