第2章 变量和简单数据类型

2.1 运行hello_world.py时发生的情况

运行hello_world.py时，Python都做了些什么呢？下面来深入研究一下。 实际上，即便是运行简单的程序，Python所做的工作也相当多。

运行文件hello_world.py时，末尾的.py指出这是一个Python程序，因此编 辑器将使用Python解释器 来运行它。Python解释器读取整个程序，确定 其中每个单词的含义。

```
In [4]:
```
```
print("Hello World")
```
```
Hello World
```

2.2 变量

```
In [5]:
```
```
message = 'hello world!'

print(message)
```
```
hello world!
```

添加变量导致Python解释器需要作更多的工作。 处理第一行代码的时候，它将变量message与文本'hello world'相关联。 处理第二行代码时，它将与变量相关连的值打印到屏幕。

2.2.1 变量的命名与使用

在Python中使用变量时，需要遵守一些规则和指南。违反这些规则将引发错误，而指南旨在让你编写的代码更容易阅读和理解。 请务必牢记下述有关变量的规则。

- 变量名只能包含字母、数字和下划线。变量名能以字母或下划线打头，但不能以数字打头。例如，可将变量命名为message_1 ，但不能将其命名为1_message。
- 变量名不能包含空格，但能使用下划线来分隔其中的单词。例如，变量名greeting_message 可行，但变量名greeting message 会引发错误。
- 不要将Python关键字和函数名用作变量名，即不要使用Python保留用于特殊用途的单词，如print。
- 变量名应既简短又具有描述性。例如，name 比x好，student_name比s_n好，name_length比length_of_persons_name好。
- 慎用小写字母i和大写字母O，因为它们可能被人错看成数字1和0。

练习

```
In [6]:
```
```
message = "Hello Python Crash Course reader!"
print(mesage)
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [6], in <cell line: 2>()
      1 message = "Hello Python Crash Course reader!"
----> 2 print(mesage)

NameError: name 'mesage' is not defined
```

程序存在错误时，Python解释器将竭尽所能地帮助你找出问题所在。程序无法成功运行时，解释器将提供一个traceback。traceback是一条记录，指出了解释器尝试运行代码时，在什么地方陷入了困境。

2.2.3 变量是标签

变量常被描述为可用于存储值的盒子。在你刚接触变量时，这种定义可能很有帮助，但它并没有准确描述Python内部表示变量的方式。

变量是可以赋给值的标签，也可以说变量指向特定的值。

2.3 字符串

大多数程序定义并收集某种数据，然后使用它们来做些有意义的事情。有鉴于此，对数据进行分类大有裨益。我们将介绍的第一种数据类型是字符串。字符串虽然看似简单，但能够以很多不同的方式使用。

字符串就是一系列字符。在Python中，用引号括起的都是字符串，其中的引号可以是单引号，也可以是双引号，如下所示：

```
In [7]:
```
```
print("This is a string.")
print('This is also a string.')


print("I told my friend, "Python is my favorite language!"")
print("The language 'Python' is named after Monty Python, not the snake.")
print("One of Python's strengths is its diverse and supportive community.")
```
```
This is a string.
This is also a string.
I told my friend, "Python is my favorite language!"
The language 'Python' is named after Monty Python, not the snake.
One of Python's strengths is its diverse and supportive community.
```

字符串的一些常用方法

```
In [8]:
```
```
#修改字符串的大小写等

name = "Noel O'Connor"
print(name.upper())
print(name.lower())
print(name.title())
print(name.capitalize())
print(name.)
```
```
  Input In [8]
    print(name.)
               ^
SyntaxError: invalid syntax
```

```
In [9]:
```
```
#在字符串中使用变量
first_name = "Noel"
last_name = "O'Connor"

print(f"{first_name} {last_name}")
print(f"{first_name.lower()} {last_name.lower()}")
print("{} {}".format(first_name, last_name))

#注意 f字符串是Python 3.6引入的。如果你使用的是Python 3.5或更早的版本，需要使用format()方法，而非这种语法。
```
```
Noel O'Connor
noel o'connor
Noel O'Connor
```

```
In [10]:
```
```
# 使用制表符或换行符来添加空白
print("Advanced\tProgramming")
print("Languages:\nPython\nC\nJavaScript")
```
```
Advance Programming
Languages:
Python
C
JavaScript
```

```
In [11]:
```
```
# 删除空白
print(" hello world!   ")
print(" hello world!   ".rstrip())
print(" hello world!   ".lstrip())
print(" hello world!   ".lstrip())
```
```
 hello world!
hello world!
 hello world!
hello world!
```

2.4 数

在编程中，经常使用数来记录得分、表示可视化数据、存储Web应用信息，等等。Python能根据数的用法以不同的方式处理它们。鉴于整数使用起来最简单，下面就先来看看Python是如何管理它们的。

2.4.1 整数

在Python中，可对整数执行加（+）减（-）乘（*）除（/）等运算。

```
In [12]:
```
```
print(2+3)
print(2-3)
print(2*3)
print(2/3)
print(2//3)
print(2%3)
print(2**3)
```
```
5
-1
6
0.6666666666666666
0
2
8
```

2.4.2 浮点数

Python将所有带小数点的数称为浮点数 。 从很大程度上说，使用浮点数时无须考虑其行为。你只需输入要使用的 数，Python通常会按你期望的方式处理它们：

```
In [13]:
```
```
print(0.1 + 0.1)
print(0.2 + 0.2)
print(2 * 0.1)
```
```
0.2
0.4
0.2
```

但需要注意的是，结果包含的小数位数可能是不确定的：

```
In [14]:
```
```
print(0.1 + 0.1)
print(0.2 + 0.1)
print(3*0.1)
```
```
0.2
0.30000000000000004
0.30000000000000004
```

2.4.3 整数和浮点数

将任意两个数相除时，结果总是浮点数，即便这两个数都是整数且能整除：

```
In [15]:
```
```
print(4/2)
```
```
2.0
```

在其他任何运算中，如果一个操作数是整数，另一个操作数是浮点数，结果也总是浮点数：

```
In [16]:
```
```
# 在其他任何运算中，如果一个操作数是整数，另一个操作数是浮点数，结果也总是浮点数：
print(1 + 0.1)
print(2 * 0.1)
print(3.0 ** 2)
```
```
1.1
0.2
9.0
```

2.4.4 数中的下划线

书写很大的数时，可使用下划线将其中的数字分组，使其更清晰易读：

```
In [17]:
```
```
universe_age = 14_000_000_000
print(universe_age)
```
```
14000000000
```

当你打印这种使用下划线定义的数时，Python不会打印其中的下划线。这是因为存储这种数时，Python会忽略其中的下划线。
将数字分组时，即便不是将每三位分成一组，也不会影响最终的值。在Python看来，1000与1_000没什么不同，1_000与10_00也没什么不同。这种表示法适用于整数和浮点数，但只有Python3.6和更高的版本支持。

2.4.5 同时给多个变量赋值

```
In [18]:
```
```
x, y, z = 0, 0, 0
print(x,y,z)
```
```
0 0 0
```

2.4.6 常量

常量类似于变量，但其值在程序的整个生命周期内保持不变。Python没有内置的常量类型，但Python程序员会使用全大写来指出应将某个变量视为常量，其值应始终不变：

```
In [19]:
```
```
MAX_CONNECTIONS = 5000
```

2.5 注释

在大多数编程语言中，注释是一项很有用的功能。本书前面编写的程序中都只包含Pythonn代码，但随着程序越来越大、越来越复杂，就应在其 中添加说明，对你解决问题的方法进行大致的阐述。注释让你能够使用自然语言在程序中添加说明。

2.5.1 如何编写注释

在Python中，注释用井号（#）标识。井号后面的内容都会被Python解释器忽略，如下所示：

```
In [20]:
```
```
# 向大家问好。
print("Hello Python people!")
```
```
Hello Python people!
```

```
In [21]:
```
```
print??
```

2.6 Python之禅

在Python的解释器中隐藏一个彩蛋，输入import this就会返回19条Python之禅

```
In [22]:
```
```
import this
```
```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

彩蛋

最开始Python是吉多·范罗苏姆个人的一个实验项目（skunkworks）。为了加快Python发展，他采用了一些原则。其中包括省时规则（timesaving rules）：

- 尽可能从其他地方借用想法，只有它有意义。
- “事情应该尽可能简单，但不要简单。”（来自爱因斯坦）
- 做好一件事情。（来自UNIX哲学）
- 不要太关心性能，可以在后面需要时进行优化。
- 不要与环境抗争，顺其自然。
- 不要尝试完美，因为“足够好”通常就是这样。
- （因此）有时可以偷工减料。尤其是后面可以完善的情况下。

https://zhuanlan.zhihu.com/p/154171577

2.7 小结

在本章中，你学习了：

- 如何使用变量；
- 如何创建描述性变量名以及如何消除名称错误和语法错误；
- 字符串是什么，以及如何使用小写、大写和首字母大写方式显示字符串；
- 使用空白来显示整洁的输出，以及如何删除字符串中多余的空白；
- 如何使用整数和浮点数；
- 一些使用数值数据的方式。

你还学习了如何编写说明性注释，让代码对你和其他人来说更容易理解。最后，你了解了让代码尽可能简单的理念。

```
In [23]:
```
```
# The end
```