第2章 - Python 的变量

作者： 何吉波博士，优视眼动科技公司创始人，hejibo@usee.tech，
http://www.usee.tech

Python语言的变量主要用于存储数据。变量的类型有许多重，包括数字(Numbers)、字符串 (Strings)、逻辑变量 (Booleans)、集合 (Sets)、列表(Lists)、元组 (Tuples)、数组 (NumPy arrays)和字典 (Dictionary)等。下面我们将分别介绍这些变量类型。

变量名

Python的变量名可以是以英文字符或者下划线(_)开始的任何变量，如_hello, This, thisVariable等。变量名可以包括英文字符，数字和下划线。大家需要注意的是， Python的变量名是大小写敏感的。This和this是不同的变量。 此外，Python的变量名不能是语言内置的保留字。下面的表格显示的是Python的保留字的列表。

赋值

与Java或者C语言等不同， Python的变量并不需要提前申明，可以直接赋值。Python的一个变量名也可以重新赋予一个新的类型的变量值，而通常不需要强制地进行数据类型的转换。 比如下面的代码，可以直接将一个字符串“Hello World”和浮点数1.0直接赋给thisString这个变量名。

```
thisString = "Hello World"
thisString = 1.0
```

Python的变量不需要提前申明，赋予新值时也不需要强制转换，可以让Python的代码更加简洁。但缺乏申明等也会让代码容易出现Bug。我们的Python代码出错时，经常是由于变量名的大小写没有注意， 或者是后面的一个变量名与代码前部

分的变量名重复，导致了变量被覆盖了。

如果您想得到一个变量的当前值，您只需要在Python编辑器中输入print thisString 或者在Python解释器中输入thisString然后回车运行就可以了。

您可以通过调用变量本身来更新变量的值，例如

```
aNumber = 1314
aNumber = aNumber + 520
```

您也可以通过其它变量来为新的变量赋值，例如

```
a=3
b=4
c= a + b
```

###关于赋值的小技巧

加入深度复制和浅度复制的讨论。

数字 (Numbers)

Python的数字变量可以直接进行常见的加(+)、减(-)、乘(*)、*除(/)，求模(%)和求幂(*)*等表达式。Python的这些表达式非常接近我们常见的数学表达式。由于Python表达式的可读性，我经常使用Python来做计算，而不是用操作系统自带的计算器。

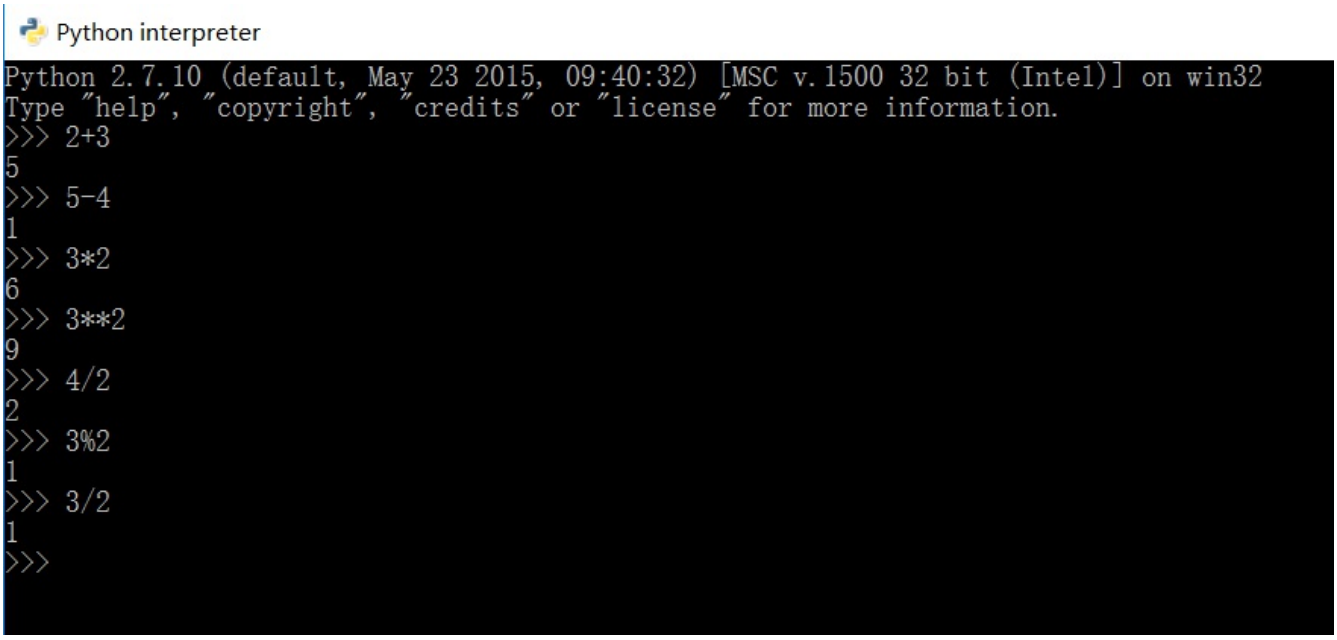

图2.1. 常见python表达式：加、减、乘、除，求模和求幂

细心的朋友们可能已经发现，其它的表达式的结果都正确，可是为什么在Python中3/2的结果是1，而不是1.5呢？这是一个很重要的问题。下面我们将详细解释一下这个问题。

Python的数字变量主要包括整型(Integer)和浮点型（Float）两类。整型数字就是不带小数部分的变量， 比如520，0，-52。浮点数字是带小数部分的变量，如0.0， 3.1415926和520025.1314等。整型和浮点型可以相互转换。int(520025.1314)的输出结果为整型520025。float(0)的输出结果为浮点数0.0。由于Python允许我们不用申明变量的类型，因此，我们绝大多数时候都不需要注意一个数字变量是整型变量还是浮点型变量。但是，当参与计算的有除法时，我们就要特别小心了。 因为，Python的一个表达式的结果的数字类型是由参与计算的数字类型决定的。 如果表达式中全都是整型，那么计算结果也是整型。如果表达式中有一个是浮点型，那么计算结果也是浮点型。在Python中3/2的结果是1. 而3.0/2和3/2.0, 3.0/2.0的结果都是1.5。如果我们预期计算结果应该是浮点数，那么，我们在参与计算的数字中至少需要有一个浮点数。 我们也可以通过float()函数明确地表示我们要求计算结果也是浮点数。例如，float(3)/2和3/float(2)的结果都是1.5。请读者朋友们理解图2.1.中显示的结果。

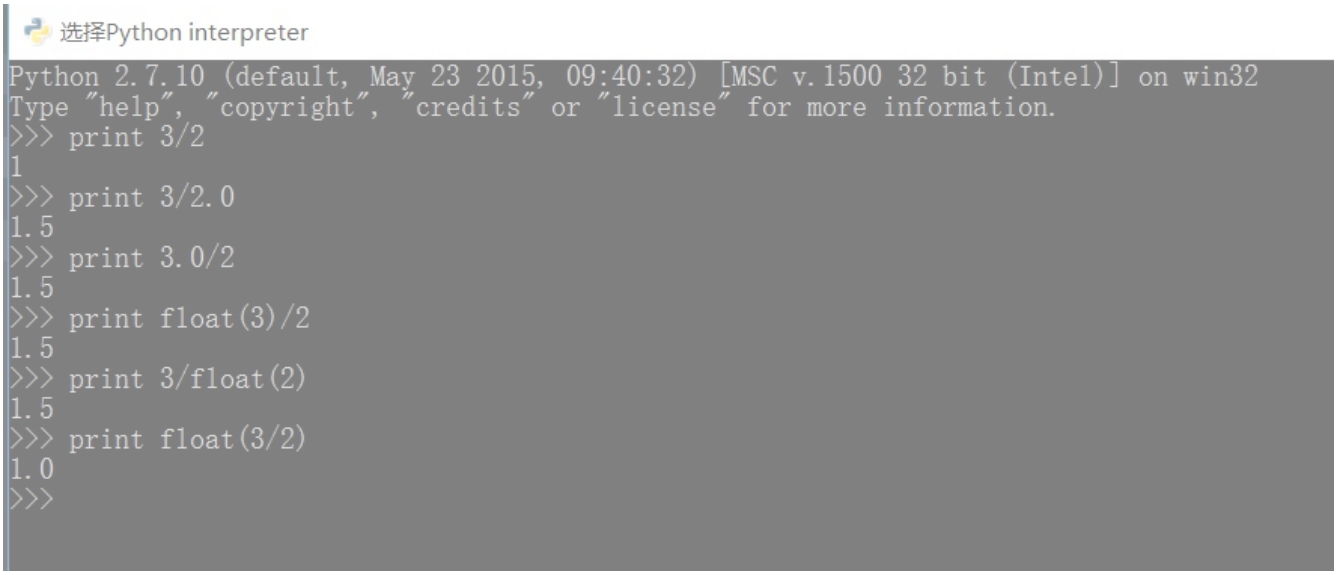

图2.2. 除法与浮点数

逻辑变量 (Booleans)

布尔逻辑变量 (Booleans)主要用于逻辑判断，它的值通常为True 或者False。读者朋友们请注意的是， Python 的变量是大小写敏感的。 True和False是逻辑变量值，而true和false不是逻辑变量的值。

常用的逻辑判断操作包括==（是否等于），！ =（不等于），>（大于），>=（不小于于），<（小于）,<=（不大于）， and 或者&（且），or 或者|（或）， ^ (异或,XOR, exclusive or), is, is not, not等。

进行逻辑判断时， 需要注意的是==与=的区别。 ==是“是否等于”的逻辑判断，而=是赋值操作。 我们写代码时经常的bug就是把==错写成了=。另外就是，不同的语言的不等于可能会有差别，Python的不等于是!= （Matlab的不等于是~=）。is和 is not分别相当于==和!=，但是更接近自然语言，增加了Python的易读性。not 是一种否定的操作。 如果a 是True, not a就是False。请读者朋友们在Python解释器中尝试下面的代码。

```
a=1
```

```
b=2
```

```
a==b
```

False

```
a is b
```

False

```
a is not b
```

True

```
not a
```

False

and , &, or 和| 用于支持多项逻辑操作。 and和&等价，只有所有的项为True时，整个逻辑操作才为真。 or和|等价，只要有一项为真时， 整个逻辑操作就为真。 我个人更偏好使用and 和or ， 因为它的易读性高于&和|。请读者朋友们在Python解释器中尝试下面的代码。

```
c=True
```

```
d=False
```

```
c and d
```

False

```
c & d
```

False

```
c or d
```

True

```
c | d
```

True

与or (|)相对比的是^（异或）。or 是只要有一个逻辑项为真时，整个逻辑运算就为真。 而^（异或）只有当两个变量不相同时，整个逻辑运算才算真。请大家尝试下面的代码。

```
e = True
```

```
f = False
```

```
g = True
```

```
e or f
```

True

```
e ^ f
```

True

```
e ^ g
```

False

对于其它语言来说，要说明是一个数的区间时， 我们通常需要两个逻辑判断，比如e>5 & e<8，而对于Python，可以更简洁地写为5<e<8。这样的表达，更接近我们通常的数学书写习惯。

```
e=6
```

```
e>5 & e<8
```

True

```
5<e<8
```

True

字符串 (Strings)

Python的字符串可以用单引号(')， 双引号(")和三引号(''' 或者""")定义。比如下面的代码，a,b和c都是相同的值的字符串。 Python 支持单引号、双引号和三引号的好处就是，在字符串中依然可以有引号。比如，我们引用谁说了什么话时，本来就需要在字符串中使用引号，如下面代码中的字符串d所示。此外，三引号"""的字符串经常用于函数的说明文档中。在比较字符串时，只与引号内的内容

有关系，而与它是单引号、双引号或者三引号无关。下面示例中的a,b和c都是等价的字符串。如果我们的字符串中本来就需要单引号或者双引号时，我们就应该用'''或者"""来定义这个字符串了。例如下面示例中的d和e变量。

```
a = 'I love Python'
b = "I love Python"
c = """I love Python"""
a == b
b == c
d = "Jibo said,'I love Python'."
e = '''She hails:"Smart is sexy. Programmers are
cool".'''
```

字符串中可以有转意字符，比如\t代码着制表符，相当于在文本中输入了一个键盘上的tab按钮。\n代码着换行字符。

```
a = "I love newlines.\nI wish I could marry
them.\n\nSigh..."
b = "TABS\tARE\tAWESOME\t!"
```

Python的优雅和简洁之处在于，她的字符串操作非常直观和简洁。其它语言需要很多代码才能完成的字符串操作，Python只需要+和*，以及[]这个切割操作符就可以完成。例如，下面的代码中使用c=a+b就可以将a和b两个字符串连接在一起。而使用*就可以将字符串复制10倍，得到c10，也就是说10遍"I love Python\n"

```
a = "I love"
b = "Python"
c = a+b
c10 = (a+" "+b+"\n")*10
```

字符串的索引与切割 []

在字符串操作中， 我们通常需要得到字符串中的一部分，这个通常叫做字符串的索引（index）或者切割（slicing）操作。在Python语言中， 使用[]完成字符串的索引与切割。[]里面可以有[start: end :step] 开始、步长和结尾三个参数，而且每一个参数都是可选的。当start参数缺失时，默认值为0。 step的默认值为1.end的默认值为-1或者字符串的结尾。而且，start和end参数可以是正数，也可以是负数。正数表示从左向右计数。负责表示从右向做计数。在编程语言中，我们是从0开始计数的。因此，字符串的第一个字符为0位置。最右侧的字符为-1位置。请读者朋友们在Python的解释器（interpretter）下面练习输入下面的代码， 熟悉如何通常Python进行索引。

```
sayLove = "I love Python"
```

```
sayLove[:]
```

'I love Python'

```
sayLove[0]
```

'I'

```
sayLove[-1]
```

'n'

```
sayLove[3]
```

'o'

当我们在[start: end :step]中有两个参数或者三个参数时，就是字符串的切割操作了。下面的示例代码就是觉的slicing操作。 Python让人着迷的地方在于，很多简单的操作，可以胜过其它语言烦杂的代码。例如sayLove[::-1]中，start和end都是默认参数，我们只需要将step赋值为-1，sayLove[::-1]就完成了将字符串从尾到头的转置。

```
sayLove = "I love Python"
```

```
sayLove[2:6]
```

'love'

```
sayLove[2:-1]
```

'love Pytho'

```
sayLove[2:]
```

'love Python'

```
sayLove[::-1]
```

'nohtyP evol I'

字符串的常用函数

Python提供了许多字符串的函数操作，让我们的使用更加便捷。包括lower(),upper(),capitalize(),center(),count(),replace(), split()等。建议读者朋友们逐一实验，了解每一个函数的作用和操作。

```
a = "I love Python"
a.lower()
a.upper()
a.capitalize()
a.center(10)
a.count('love')
a.replace('love', 'really love')
a.split()
```

```
>>> a = "I love Python"
>>> a.lower()
'i love python'
>>> a.upper()
'I LOVE PYTHON'
>>> a.capitalize()
'I love python'
>>> a.center(10)
'I love Python'
>>> a.count('love')
1
>>> a.replace('love', 'really love')
'I really love Python'
>>> a.split()
['I', 'love', 'Python']
>>>
```

图2.2.Python字符串常用函数操作结果

字符串的格式化输出

我们经常需要将字符串格式化输出到一个文本文件中，或者按一定的格式输出到终端。这时，我们可以使用%s,%d, %f, %7.2f这样的格式串来设置输出格式。其中，%s表示一个字符串或者字符，%d 表示以整型输出，%f格式以浮点型式输出，%7.2f表示浮点型的整数部分最多7位，小数部分最多两位数字。具体操作如下例所示：

```
a = "%s love Python"
print a%"Jibo"
b = "%s loves %s"
print b%("Jibo","Python")
print b%("Jack Ma","Java")
print "My final grade is %f to be exact, which is
%2.2f after round up to two decimals, or %d for
short."%(99.5672,99.5672,99.5672)
```

```
>>> a = "%s love Python"
>>> print a%"Jibo"
Jibo love Python
>>> b = "%s love %s"
>>> print b%("Jibo","Python")
Jibo love Python
>>> print b%("Jack Ma","Java")
Jack Ma love Java
>>> print "My final grade is %f to be exact, which is %2.2f after round up to two decimals, or %d for shor
t."%(99.5672,99.5672,99.5672)
My final grade is 99.567200 to be exact, which is 99.57 after round up to two decimals, or 99 for short.
>>>
```

图2.3 字符串的格式化输出

字符串的高级操作

Python比较省时和省事，因为很多复杂的操作可以叠加在一起用。比如上面的函数和切割操作[]可以结合在一起使用。 例如split()函数可以将一个字符串分割开来，得到一个列表。 我们可以直接在sayLove.split()的结果上用[]直接进行切割操作。例如，sayLove.split()[1]将得到sayLove的第二个单词，而sayLove.split()[1][0]将返回第二个单词的首字母。

```
sayLove = "I love Python"
```

```
sayLove.split()[1]
```

'love'

```
sayLove.split()[1][0]
```

'l'

集合 (Sets)

列表 (Lists)

元组 (Tuples)

数组 (NumPy arrays)

字典 (Dictionary)