# 实验2.Python基础知识

实验目的：

1.熟悉Python的数据类型、数值运算

2.了解常量与变量

3.掌握常用的运算符，掌握优先级及结合规律

4.掌握Python中的变量、数字、运算符、表达式和基本输入输出的用法。

实验平台：

Windows系列+Python3.X

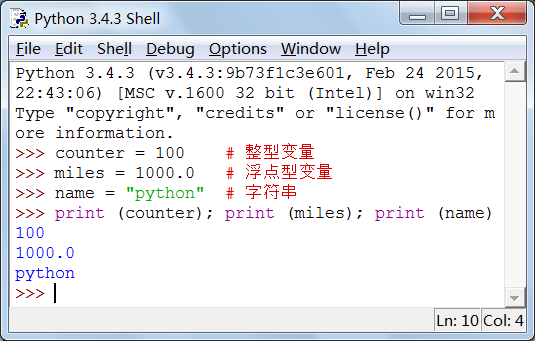
主要实验内容：

#### 一、数据类型

在Python 中，变量就是变量，它没有类型，我们所说的“类型”是变量所指的内存中对象的类型。

赋值运算符（=）用来给变量赋值。

赋值运算符左边必须是一个变量名,右边是存储在变量中的值。例如：

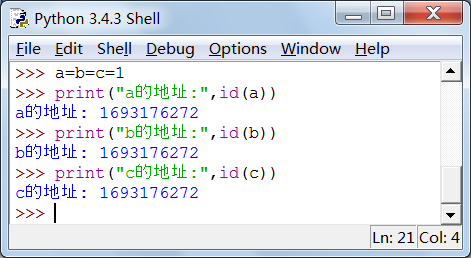


Python允许同时为多个变量赋值，例如：

a = b = c = 1

注意：

* 以上赋值语句创建了一个整型对象，值为1，三个变量被分配到相同的内存空间上。以下的运行结果可以证实之：



* Python中的一切事物皆为对象，并且规定参数的传递都是对象的引用，因此，对象的赋值实际上是对象的引用。

也可以为多个对象指定多个变量，例如：

a, b, c = 1, 2, "python"

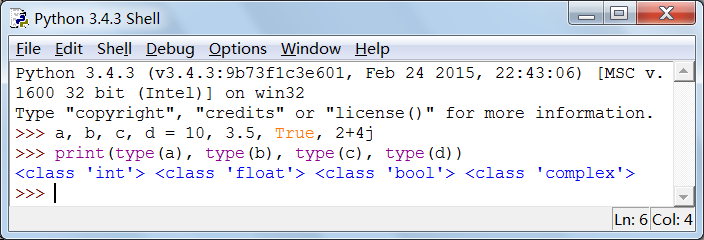
以上形式的赋值语句也称作复合赋值语句，两个整型对象1和2分配给变量a和b，字符串对象“python”分配给变量c。

Python 3中有六种标准的数据类型，它们是：Number（数值）、String（字符串）、List（列表）、Tuple（元组）、Dictionary（字典）、Sets（集合）。本节介绍数值类型，后续各节分别介绍其余的各种类型。

Python 3支持四种数值类型：int（整型）、float（浮点型）、bool（布尔型）、complex（复数型）。

在Python 3里，只有一种整数类型 int，表示为长整型，而不像python 2 那样区分标准整型与长整型。

与大多数编程语言一样，数值类型的赋值和计算都是很直观的。内建的函数type()可以用来查询变量所指的对象类型。



当指定一个值时，就创建了一个Number 对象：

var1 = 1

var2 = 2

同样，这里的var1和var2是两个对象引用。使用del语句可以删除对象引用。del语句的语法是：

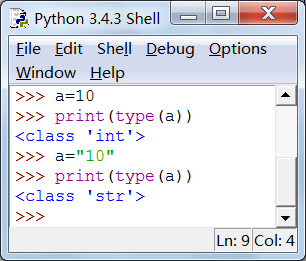
del var1[,var2[,var3[....,varN]]]

例如：

del var

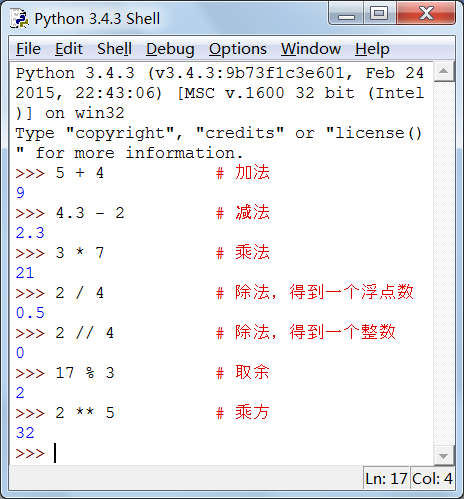
del var\_a, var\_b

一个变量可以通过赋值指向不同类型的对象。例如：



1．数值运算

像其他语言一样，Python的数值运算包括加、减、乘、除四则运算以及取余、乘方运算等。



注意：

* 除法运算符“/”总是返回一个浮点数，要返回整数应使用“//”运算符。
* 在混合运算时，Python把整型数转换成为浮点数。
* 除上述运算外，Python的整数还支持“位运算”，详见表2-1。

2．数值类型实例

Python 3支持的数值类型实例如下：

int型实例：10，100，-786，080，-0490，-0x260，0x69。

float型实例：0.0，15.20，-21.9，32.3e+18，-90.，-32.54e100，70.2e-12。

bool型实例：True，False。

Python还支持复数。复数由实部和虚部构成，可以表示为a + bj或者complex(a,b)，其中实部a和虚部b都是浮点型。

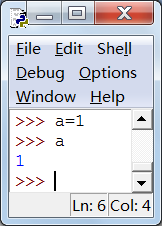
complex型实例：3.14j，45.j，9.322e-36j，.876j，-.6545+0j，3e+26j，4.53e-7j。

#### 二、 常量与变量

几乎所有的编程语言都有变量和常量的概念，它们和数学上的概念很相似。

1．变量

变量和数学函数中的变量一样。顾名思义，变量就是其值可以改变的量，只不过在程序中变量不仅仅可以是数字，还可以是字符串等等。变量名的命名方式严格遵循上一节提及的Python标识符命名规则，即由英文字母、Unicode字符（含汉字）、数字、下划线组成，且不能以数字开头。



上面的例子，通过赋值运算符把1赋值给a，这个a就是我们创建的一个变量。

a=1

b=2

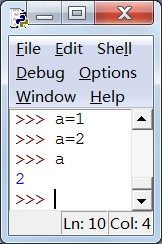
这里我们用一个字母来表示变量，通过赋值运算符来给变量赋值。

Python是动态语言，所以我们不需要像C语言那样提前声明一个变量的数据类型。C语言使用变量之前需要声明一下。例如：

int a=1;

而Python直接使用 a=1即可。这就是动态语言的好处，我们无需关心变量本身的数据类型，Python有自己的判断机制。

注意：给变量赋值的时候，Python只会记住最后一次所赋的值。例如：



先给a赋值1，再赋值2，最后输出a的时候显示的是最后一次所赋的值2。

2．常量

常量与变量相对应，就是程序运行过程中其值不可改变的量。例如：

PI=3.1415926

这时我们就认定PI是个常量，也就是说，PI始终代表3.1415926。实际上，Python中并没有严格意义的常量，因为Python中没有保证常量不会改变的机制。一般来讲，我们使用全部大写的字母来表示常量。尽管我们称PI为常量，但仍然可以给PI重新赋值。

换言之，Python的世界里本来就没有常量，编程时主动不修改的变量也就伪装成了常量。用大写字母来“注明”常量，只有提醒的意义，其实这个值还是可以改变的。

在编程过程中，我们可能会用到常量的概念，所以这里还是要提一下。

#### 三、运算符与优先级

1．Python的运算符

Python的运算符十分丰富，表2-1简要介绍了Python的运算符及其用法。

表2-1 Python运算符及其用法

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 名称 | 说明 | 例子 |
| + | 加 | 两个对象相加（对字符串则是连接） | 3 + 5 得到 8。 'a' + 'b' 得到 'ab'。 |
| - | 减/负号 | 得到一个负数，或是一个数减去另一个数 | -5.2 得到一个负数。 50 - 24 得到 26。 |
| \* | 乘 | 两个数相乘或是返回一个被重复若干次的字符串 | 2\*3 得到 6。 'la'\*3 得到 'lalala'。 |
| \*\* | 幂 | 指数运算，返回 x 的 y 次幂 | 3\*\*4 得到 81 （即 3\*3\*3\*3）。 |
| / | 除 | x 除以 y | 4/3 得到1.3333333333333333。 |
| // | 取商数 | 返回商的整数部分 | 4 // 3.0 得到 1.0。 |
| % | 取余数 | 返回除法的余数 | 8%3 得到 2。 -25.5%2.25 得到 1.5。 |
| << | 左移 | 把一个数的位向左移一定数目（每个数在内存中都表示为位或二进制数字，即 0 和 1） | 2 << 1 得到 4。因为 2 用二进制表示时为10B，往左移一位变成二进制的100B，二进制的100B用十进制表示则为4。 |
| >> | 右移 | 把一个数的位向右移一定数目 | 11 >> 1得到5。因为11用二进制表示时为1011B，向右移动1位后得到101B，即十进制的 5。 |
| & | 位和 (Bitwise AND) | 数的位和 | 5 & 3 得到 1。  (101B &11B得到1B) |
| | | 位或 (Bitwise OR) | 数的位或 | 5 | 3得到 7。  (101B | 11B 得到111B) |
| ~ | 位翻转 | 位翻转 | x 的位翻转是 -(x+1) ，~5(101B) 得到 -6(-110B)。 |
| < | 小于 | 比较运算符，返回 x 是否小于 y 的结果。返回 1 则表示真，返回 0 则表示假。Python 中也可以用特殊的变量True表示真，用False表示假。注意首字母为大写。 | 5 < 3 返回 0（即 False）；而3 < 5 返回1（即True）。也可以任意连接比较运算符：3 < 5 < 7 返回 True。 |
| > | 大于 | 返回 x 是否大于 y | 5 > 3 返回 True。如果两个运算子都是数字，它们会先被转换成相同的类型后才开始计算。如果 x 和 y 类型不同, 则会返回 False。 |
| <= | 小于等于 | 返回 x 是否小于等于 y | x = 3; y = 6; x <= y 返回 True。 |
| >= | 大于等于 | 返回 x 是否大于等于 y | x = 4; y = 3; x >= y 返回 True。 |
| == | 等于 | 比较两个对象是否相等 | x = 2; y = 2; x == y 返回 True。x = 'abc'; y = 'Abc'; x == y 返回 False。 |
| != | 不等于 | 比较两个对象是否不相等 | x = 2; y = 3; x != y 返回 True。 |
| not | 布尔「非」 | 如果 x 为 None (空), 0, False, 或空字符串。则 not x 会返回 True, 否则 not x 会返回 False。 | x = True; not x 返回 False。 x = False; not x 返回 True。 |
| and | 布尔「与」 | 如果 x 为 False，x and y 返回 False，否则它会返回 y 的计算值。 | x = False; y = True; x and y 的结果会返回 False。在这个例子里， Python 语言并不会计算 y 的值，因为 x 的值已经是 False, 所以这个表达式的值肯定是 False。这个现象称为短路计算。 |
| or | 布尔「或」 | 如果 x 为 True，则返回 True，否则返回 y 的计算值。 | x = True; y = False; x or y 的结果会返回 True。 短路计算在这里也同样适用。 |

2．Python运算符的优先级

表2-2给出Python运算符的优先级，其排列顺序是从最低的优先级（最松散地结合）到最高的优先级（最紧密地结合）。这意味着在计算一个表达式时，Python首先计算列在表下部的运算符，然后再计算列在表上部的运算符。

表2-2 Python运算符的优先级

|  |  |
| --- | --- |
| 运算符 | 描述 |
| lambda | Lambda 表达式 |
| or | 布尔「或」 |
| and | 布尔「与」 |
| not x | 布尔「非」 |
| in，not in | 成员测试 |
| is，is not | 同一性测试 |
| <，<=，>， >=，!=，== | 比较 |
| | | 按位或 |
| ^ | 按位异或 |
| & | 按位与 |
| <<，>> | 移位 |
| +，- | 加法与减法 |
| \*，/，% | 乘法、除法、取余数 |
| +x，-x | 正负号 |
| ~x | 按位翻转 |
| \*\* | 指数 |
| x.attribute | 属性参考 |
| x[index] | 下标 |
| x[index:index] | 寻址段 |
| f(arguments...) | 函数调用 |
| (experession,...) | 绑定或元组显示 |
| [expression,...] | 列表显示 |
| {key:datum,...} | 字典显示 |
| 'expression,...' | 字符串转换 |

上述的运算符优先级表决定了哪个运算符在别的运算符之前计算。然而，如果想要改变它们的计算顺序，可以使用圆括号。例如，想要在一个表达式中让加法在乘法之前计算，那么就要写成类似于(1 + 2) \* 3的形式。

3．Python运算符的结合规律

运算符通常自左向右结合，即具有相同优先级的运算符按照从左向右的顺序计算。例如，1 + 2 + 3被解释为(1 + 2) + 3。也有一些运算符是自右向左结合的，如赋值运算符。这样的运算符，其结合顺序正相反，即a = b = c被解释为a = (b = c)。

注意：合理使用括号可增强代码的可读性。在很多场合下使用括号都是一个好主意，而没用括号的话，可能会使程序得到错误的结果，或使代码的可读性降低，引起阅读者的困惑。括号在Python语言中不是必须存在的，不过为了获得良好的可读性，使用括号总是值得的。

#### 四、练习

1．变量类型练习：写出各条命令的执行结果。

a, b, c, d = 10.0, 10, False, 3.5+4.2j

print(type(a), type(b), type(c), type(d))

2．数值运算练习：写出各条命令的执行结果。

6 / 3

6 // 3

15 % 4

2\*2\*\*3

2\*\*3\*\*2

8+2<<3