Python变量和数据类型

Python使用对象（object）模型存储数据，因此所构造的任何类型的值都以对象形式存在。事实上，当我们在解释器中直接键入一个数字或者字符串并按下回车后，就创建了一个“对象”。

Python内置的4种最基本的数据类型，包括：

- **布尔型**（用来表示真假，仅包含`True`和`false`两种取值）
- **整型**（整数，例如`42`、`100000000`）
- **浮点型**（小数，例如`3.14159`、`1.0e8`、`100000000.0`）
- **字符串型**（字符序列）

对象和变量

Python中的一切都是对象。 所有的对象都具有三个特征：

- **身份**：每个对象唯一身份标识，可以使用内建函数 `id()` 查看，如 `id(43)`， `id(obj_name)`，所得到的值可以认为是该对象的内存地址。
- **类型**：决定了该对象可以保持什么类型的值，使用内建函数 `type(obj_name)` 可以查看，注意，该函数返回的是类型对象，而非字符串对象。对象类型决定了可以对它进行怎样的操作，还决定了其包装的 **值** 是否允许被修改。对象的类型无法改变，所以Python是**强类型的（strongly typed）**。
- **值**：对象表示的数据项

某些对象有属性、值和方法，和C++一样，可以使用 `.` 操作符访问。

对象值的比较

操作符：**<**, **<=**, **>**, **>=**, **==**, **!=**, **<>** 返回值：布尔值 `True`，`False`

```
2 == 2
3.14 <= 10
'welcome' > 'sssaf'
[3, 'abc'] != ['abc', 3]
3 < 4 < 7
4 > 3 == 3
```

从最后两个例子可以看出，Python中的表达式更加的灵活自然。

对象身份的比较

先谈一下引用计数：

```
foo1 = foo2 = 4.3
```

这条语句的实质是：一个值为`4.3`的数字对象被创建，`foo1`和`foo2`这两个变量名字共同指向了此对象（**变量**仅仅是一个名字，是对对象的**引用**而不是对象本身。），即`foo1`和`foo2`是同一个对象的两个引用：

```
foo1 = 4.3
foo2 = foo1
```

第一句话使得值为`4.3`的数字对象被创建，然后其引用被赋值给`foo1`，第二句话使得`foo2`借助`foo1`同样指向了值为`4.3`的对象，这里和上一个例子的实质是相同的。

```
foo2 = 1.3 + 3
```

值为`1.3`的数字对象和值为`3`的数字对象被创建，相加后，得到一个新的值为`4.3`的对象（**此对象与上面代码中的值`4.3`对象不是同一个！**），然后`foo2`指向了这个新对象。

现在，foo1和foo2指向了两个值相同，但身份不相同的数字对象。

```
foo1 = 4.3
foo2 = 4.3
```

同样，两个值相同，身份不同的数字对象被创建，分别由`foo1`和`foo2`指向。

很多时候如果你分不清楚的话，可以使用内建函数`id()`来进行判定，可以认为`id`返回的是**对象的内存地址**，即**指针**，这样的判定方法是最有效的。如：

```
foo1 = foo2 = 4.3
id(foo1) == id(foo2)  #返回值为True
```

```
bar1 = 4.3
bar2 = 4.3
id(bar1) == id(bar2)  #返回值为False
```

通常`id()`很少使用，`is`和`is not`操作符是判别身份的最佳方式：

```
foo1 = foo2 = 4.3
foo1 is foo2      # 返回True
foo1 is not foo2  # 返回False
```

特殊情况是存在的，这通常会令人迷惑不解，如：

```
a = 4
b = 4
a is b
```

```
c = 1000
d = 1000
c is d
```

第二个结果很好理解，两个`1000`是不同的对象嘛！但第一个怎么回事？因为 **小整型量通常会在程序代码中频繁使用，为了提升效率，Python会对-1~100（注意：这个范围是可以变化的）的整型对象进行缓存，即不会重返创建。**这就是上面例子中`a is b`返回`True`的原因。

任何一个对象都有一个内部的计数器，记录着其引用的数量，当引用为0时，该对象就会被系统给收回，这就是Python进行自主内存管理的基本原理之一。

使用`type()`返回对象的类型

用法：`type(obj_name)`

```
type(4)      # 返回int
type(4.0)    # 返回float
type('abc')  # 返回str

# type返回的不是字符串，而是类型对象，如
type('abc').__name__    # 返回 'str' ，__name__是返回对象的属性
type(type('abc'))   # 返回type，Python的内建元类
```

类（class）是对象的定义。

使用`isinstance()`检查一个对象是否是某类型的实例

```
def display_num_type(num):
    print(num, 'is', end=' ')
    if isinstance(num, (int, float, complex)):
        print('a number of type: ', type(num).__name__)
    else:
        print('not a number at all!')

display_num_type(-69)
display_num_type(98.6)
display_num_type(234+2j)
display_num_type('xxx')
```

变量赋值

```
# 普通赋值方式
int_example = 211
string_example = 'So easy!'

# 增量赋值，与C语言中的算数自反赋值运算一样
int_example *= 3
string_example += 'Try it!'
print(int_example)
print(string_example)

# 多重赋值
x = y = z = 1
print(x, y, z)
# 多元赋值，很有用，用起来效率很高，括号是可选的，但保留可以增强代码可读性
(x, y, z) = (1, 2, 'a string')
print(x, y, z)
(x, y) = (y, x)  # 对x，y的值做交换，不需要第三个辅助变量了
# 不要过多考虑顺序和优先级，注重功能逻辑
```

变量名（标志符）

1. **规则**

   和C语言相似，没有长度限制，具体如下：

   - **只能包含以下字符：**
     - 小写字母（a-z）
     - 大写字母（A-Z）
     - 数字（0-9）
     - 下滑线（_）
   - **不允许以数字开头。**
   - **Python中以下划线开头的名字有特殊的含义。**

2. **关键字**

3. **built-in**

   进入解释器时，`__builtin__`模块会被自动导入，这个模块中包含一些保留的名字集合，如`open`，`input`等，一般情况下，你定义的标识符最好不要和它们冲突。

```
dir(__builtin__)  # 查看模块中所有的内建名字
```

标识符命名应该使用固有的风格，不要随便命名，离标识符、保留名字、特权名字等远一些。

数字

数字可以直接访问，是不可更改并且不可分割的原子类型。不可更改意味着变更数字值的实质是新对象的创建。Python本身支持整数和浮点数，其整数类型可以存储任意大小的整数（所能表达的数字范围和计算机的虚拟内存大小有关），这使得Python非常适合大数计算。

数字对象的创建和赋值

```
# 像大多数脚本语言一样，无需指定类型
an_int = 1
a_float = 3.1415
a_complex = 1.2 + 3.3j
```

布尔型

布尔型只有两个值，`True`和`False`。事实上，布尔型是整型的子类，对应整型的1和0。使用内建函数`bool`返回布尔对象。

```
bool()       # 返回False
bool(1)      # 返回True
bool(0)      # 返回False
bool(True)   # 返回True
bool(False)  # 返回False
True + True  # 返回2，因bool值实质是整型
```

布尔运算

布尔运算符有三个：`and`，`or`，`not`。善于使用括号以避免优先级和结合性导致的问题。

优先级由高到低依次为：`not`，`and`，`or`。

复数

语法：`real + imag j`

实数部分和虚数部分都是浮点型，虚数部分结尾必须是`j`或`J`。

复数包含两个浮点属性：`real`(实数部分)，`imag`(虚数部分)，还有一个方法： *conjugate()* ，用以获取其共轭复数。

```
a_complex = 3.5 + 2.9j
a_complex             # 返回(3.5+2.9j)
a_complex.real        # 返回3.5
a_complex.imag        # 返回2.9
a_complex.conjugate() # 返回(3.5-2.9j)
```

更新数字对象（即重新赋值，注意其本质：新对象的创建）

```
an_int += 1
a_float = 3.1415926
```

“删除”数字对象

```
del an_int
```

注意：我们只是删除了对象的引用，而不是删除了对象本身（相当于使对象内部计数器的值减少1），这时`an_int`不引用任何对象。对象本身的删除是由Python内部的内存管理功能进行的。

Python支持的数学运算

运算符	描述	示例	结果
+	加法	5 + 8	13
-	减法	90 - 10	80
*	乘法	4 * 7	28
/	浮点数除法	7 / 2	3.5
//	整数除法	7 // 2	3
%	模（求余）	7 % 3	1
**	幂	3 ** 4	81

将运算过程与赋值过程合并

```
a = 95
a -= 3
a += 8
a *= 2
a /= 3
a //= 2
a
```

除法

- 使用`/`执行**浮点**除法（十进制小数）

  即使运算对象是两个整数，使用`/`仍会得到浮点型的结果。

- 使用`//`执行**整数**除法（整除）

```
1 / 2     # 0.5
1.0 / 2   # 0.5
1.0 // 2  # 0.0

9 / 5   # 1.8
9 // 5  # 1
```

如果除数为**0**，除法运算会产生`ZeroDivisionError`异常。

```
5 / 0
```

基数

除了十进制外，Python还支持以下三种进制的数字：

- `0b`或`0B`表示二进制（以2为底）
- `0o`或`0O`表示八进制（以8为底）
- `0x`或`0X`表示十六进制（以16为底）

```
0b10
0o10
0x10
```

类型转换

两个不同类型的数字对象进行运算时，Python就要对其中一个进行强制类型转换，继而进行运算，这个道理和C语言中的自动转化是相似的。基本规则： **整型转换为浮点型，非复数转换为复数。** 总之就是： **简单类型向复杂类型转换，不精确类型向更精确类型转换。**

类型转换失败会产生`ValueError`异常。

```
int('10a')
int('98.6')
```

运算优先级

参考附录F（p.380）

使用括号来保证运算顺序与期望的一致。

数学函数

转换工厂函数

注意：转换是表现，实质是创建新对象。

- `int()`
- `float()`
- `complex()`
- `bool()`

```
int(4.225)         # 返回4，实质是生产了一个int类型对象
float(4)           # 返回4.0
complex(11, 9.0)   # 返回(11+9j)
bool(0.000001)     # 返回True
```

功能函数

- `abs()`

返回绝对值，如果参数是整型，返回整型，如果是浮点型，返回浮点类型，同样也可用于复数绝对值的计算，即返回实部和虚部平方和的二次方根。

- `divmod()`

此函数将除法和求余结合起来，返回一个包含商和余数的元组：

```
divmod(10, 3)    # 返回(3, 1)
divmod(2.5, 10)  # 返回(0.0, 2.5)
```

- `pow()` 此函数的功能和 `**` 一样，实现幂运算：

```
pow(2, 5)   # 返回32
pow(5, 2)   # 返回25
```

- `round()`

`round()` 做真正的四舍五入！可以用第二个参数指定精确到小数点后第几位：

```
round(4.499)     # 返回4
round(4.499, 1)  # 返回4.5
round(4.5)       # 返回4
```

高级数学运算

参考教材附录C（p.320）。