Numbers —— 数字

整数

Python中的变量不需要声明类型

解释器会自己进行推断，也就导致了Python的运行效率不理想

Python中的数字有`int`和`float`两种

但是本质上`int`和`float`完全就是字面意思，整数和浮点数，其精度是解释器自己推断的

其他语言中的整数也许分为好多种，比如`long`,`short`,`long long`，但是python中的整数只有`int`

你可以在idle中尝试以下代码：

```
short_int = 1
int_ = 75569
int_long = 465746846467687687
print(type(short_int), type(int_), type(int_long))
```

你会发现不论数字的大小，你得到的`type()`打印出来的结果都是`int`

你不需要考虑太多，你只需要把所有的整数视为`int`即可

其他的语言我不是很清楚，但是Python给整数的进制转换提供了良好的支持，Python的每个模块都会自动导入`builtin`这个模块，进制转换的方法就在`builtin`模块中，这个机制暂时不需要非常详细的了解

现阶段您只需要知道，当想要对一个整数进行进制转换时，可以这样做

```
int_org = 256
# 转换为二进制数字
bin(int_org)

# 转换为八进制数字
oct(int_org)

# 转换为10进制数字
bin_int = 0b1010
int(bin_int)

# 转换为16进制
hex(int_org)
```

上述的几个函数`bin()`,`oct()`,`int()`,`hex()`

函数	功能
`bin()`	转换任意进制的整数为其2进制形式
`oct()`	转换任意进制的整数为其8进制形式
`int()`	转换任意进制的整数为其10进制形式
`hex()`	转换任意进制的整数为其16进制形式

可以接受不同种类的数字，你可以使用`oct()`来转化一个16进制的数字为八进制的数字，也可以用`hex()`来对二进制的数字进行转换

另外`int()`这个其实不算是一个**函数**，有点像是`int`类型的构造函数，这一部分面向对象的内容，我们会在后续的章节进行详细的阐述

另外在使用这样的函数时您可能会发现一些东西，就是`0b`，`0o`，`0x`

这样的前缀，这是一个通用的表示进制的前缀，`0b`表示二进制，因为`10`和`0b10`其实是两个数字，使用前缀的方法可以简洁高效的消除歧义

即

前缀	意义
`0b`	二进制数字
`0x`	十六进制数字
`0o`	八进制数字

如果你还不清楚什么是进制以及进制之间如何转换，您可以参考进制

如何转换进制并不在Python的教程范围内，您可以参考补充文档Play with Python 0x10

到这里为止，您应该明白了Python中的整数以及其进制转换

浮点数

您应该发现了，我用了两个节来阐述小数和浮点数，这里我想说，浮点数和小数应该是两个概念

您输入的`0.3`，在程序中会被解释为一个二进制的数字，因为其算法的问题，会出现

```
x = 0.3
print(0.1 + 0.2 == x) # 打印出 0.1+0.2是否等于变量x
```

输出结果为

```
False
```

的情况

亦或者

```
x = 0.1 + 0.2
print(x)
```

输出结果为

```
0.30000000000000004
```

的情况。

所以在这里我用两个小节来介绍在Python中的两种不同的小数表现形式

很多其他的语言中，浮点数有不同的精度，比如`float`单精度浮点数，`double`双精度浮点数，或者配合精度修饰符`long`之类的，在Python中您不需要考虑这些，这些精度上的解释和推断，都交给解释器处理

所以您只需要写

```
example_float = 0.41
```

即可

所有的浮点数在被判断`type`的时候，都会输出为`float`

另外，您也看到了，在上述的例子中出现了`0.1 + 0.2 = 0.30000000000000004`这样的情况，所以我们判断两个浮点数的大小是否相等时(并且需要一定精度支持时)，我们需要引入一个非常小的值，一般我们称他为`EPSILON`，一般取值为`1e-8`

您可能会疑问，这个`1e-8`是个什么意思，e和后面的数字，联合起来表示n次方

`1e-8`即，1的-8次方，通过判断两个值相减结果是否小于1的-8次方，来判断这两个值是否相等

like this

```
x = 0.1 + 0.2
y = 0.2 + 0.1
x - y < 1e-8
```

结果为

```
True
```

这个例子中的x和y，都等于`0.30000000000000004`，当然可能因为电脑的操作系统，电脑的位数，和python的版本之类的和我的不一致而导致您看到的值不完全等于我写出来的这个`0.30000000000000004`，不过没关系，思路是一致的

最后，浮点数没有办法像是整数一样被`bin()`一类的函数进行进制转换，您这样做会会触发

```
TypeError: 'float' object cannot be interpreted as an integer
'float' object cannot be interpreted as an integer
```

的错误，这也是您在本教程中遇到的第一个`Error`，叫做`TypeError`，表示类型错误，Python 的解释器的错误归类非常完善，也非常的友好，当您不知道怎么debug的时候，可以试着直接在google搜索这个错误的内容。

好啦，Python 的浮点数就是这样，非常简单。

小数

看到小数，你可能会想，文章刚刚不是介绍了“小数”吗？

此小数非彼小数，这个小数又叫做$Decimal$，刚刚的那个叫浮点数$FloatingPoint$

浮点数是一个处理小数的机制，但是因为浮点数并不能很好地表达小数，所以在要求精度（比如银行的系统里）时，我们需要一种更加精确的数字

为了搞清楚为什么浮点数表达不清楚小数，我们举一个例子

数字3转换成二进制后为0b11，如果写成4个字节长度的整数，就是

`0b0000 0000 0000 0011`

任何一个整数都可以用一种简单方式，转换为由若干个2的指数乘以某个系数相加的形式

比如

$$3 = 2^0 * 1 + 2^1 * 1$$

亦或者

$$7 = 2^0 * 1 + 2^1 * 1 + 2^2 * 1$$

具体的证明过程就不在这里阐述了

但是小数这里就很难了

0.3就不能像3一样，在转换3为二进制数字的时候，用的是2的正数次方，但是小数需要用二的负数次方

$$2^{-1} = 0.5$$
$$2^{-2} = 0.25$$
$$2^{-3} = 0.125$$

要想表示0.3，可以：

$$2^{-2} + 2^{-3}$$

不论你用怎样的组合，你组和的值只能无限接近于0.3而不能等于0.3

而且计算机储存小数是以约定好的字节数存储的，有限的空间里储存一个无线巨大的“组合”是不可能的，所以只能退而求其次

但是为了精度，人们发明了一种新的数据类型，叫做$Decimal$

在Python中，想要使用这种Decimal，你需要先把他从标准库中导入进来，导入操作的作用机制我们以后会论述，在这里就当作是一个约定，我们先写上这样一行特殊的代码：

```python
from decimal import *

# 简单地创建一个decimal变量:
exmaple_decimal1 = Decimal("0.3")
exmaple_decimal2 = Decimal("0.142857142857142857142857142857")

# 打印出两个变量相加地值
print(exmaple_decimal1 + exmaple_decimal2)
```

decimal 模块旨在支持“无偏差，精确无舍入的十进制算术（有时称为定点数算术）和有舍入的浮点数算术”。 —— 摘自 decimal 算术规范说明

到此为止，初学阶段涉及的Decimal内容就结束了，但是Decimal不仅仅是刚刚展示地这样简单，可以参考这个

Python3.8.5标准库介绍

创建一个Decimal变量涉及到了一些面向对象都内容，但是不要紧，在现阶段你可以这样创建，等你有了更多地Python基础，我们可以再回来阅读接下来地内容

更丰富的Decimal介绍