Python基础教程

第2章 变量和基本数据类型

1.1 标识符

标识符是计算机语言中允许作为名字的有效字符串集合。

（1）有一部分是关键字，构成语言的标识符；

---保留字（不能用于其他用途）--> 语法错误（**SyntaxError**异常）

（2）“内建”（**built-in**）的标识符集合

---不是保留字，但不推荐使用这些特别的名字

（3）合法的标识符

---字符串规则和其他大部分用C编写的高级语言相似

1.1.1 合法的标识符

- 在Python中，所有标识符可以包括字母、数字以及下划线（_），但不能以数字开头。
- Python中的标识符是区分大小写的。
  - V = 1
  - V3 = 2
  - 3v = 3

1.1.1 合法的标识符

- 以双下划线开头的（__foo）代表类的私有成员；
- 以双下划线开头和结尾的（__foo__）代表Python里特殊方法专用的标识，如 __init__( ) 代表类的构造函数。
- 注意：避免用“下划线_”作为变量名的开始

1.1.2 关键字(保留字符)

这些标识符在Python程序中具有特定用途，不能用作常规变量名或函数名，或任何其他标识符名称。

除了True、False和None这三个以外，其他的保留字都是小写单词。

```
>>> import keyword
>>> keyword.kwlist
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

1.2 行和缩进

Python与其他语言最大的区别就是：

代码块不使用大括号（{ }）来控制类，函数以及其他逻辑判断，而是使用缩进来写模块。

缩进的空白数量是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个必须严格执行。

缩进相同（建议4个空格）的一组语句构成一个代码块，称之代码组。

1.2 行和缩进

像if、while、def和class复合语句，以关键字开始，以冒号（：）结束，之后的一行或多行构成代码组。

首行及后面的代码组称为一个字句（clause）。

```
1  #以TAB键作为缩进
2  def foo(args):
3      if args is not None:
4          print args
5
6  foo("Heelo,World!")
```

```
1  #以单个空格键作为缩进
2  def foo(args):
3   if args is not None:
4    print args
5
6  foo("Heelo,World!")
```

1.3 多行语句

Python语句中一般以新行作为语句的结束符，但是可以使用斜杠（\）将一行的语句分为多行显示，如：

```
>>> string = "Hello, \
World!"
>>> count = 1 + \
        2 + \
        3
>>> print (string, count)
```

```
Hello, World! 6
```

1.3 多行语句

但是，对于语句中包含[ ], { } 或 ( ) 括号就不需要使用多行连接符：

```
13  days = {
14          'Monday':'星期一',
15          'Friday':'星期五'
16          }
```

1.4 同一行显示多条语句

Python可以在同一行中使用多条语句，语句之间使用分号（;）分割：

```
>>> print ("Hello");a = 1 + 1
Hello
>>> a
2
>>> print ("Hello"); print ("Hello")
Hello
Hello
```

注意：同一行上书写多个语句会大大降低代码的可读性，不提倡！

1.5 注释

Python中单行注释采用“ # ” 开头，后面接注释语句。

Python没有块注释，所以多行注释也是采用“ # ”。当然也可以使用三引号(“”“”) 或者(’’’)把多行语句当做字符串，从而达到类似注释的效果。

```
2  print "Hello,World!" #单行注释
3  #多行注释
4  #多行注释
5
6  '''
7  三引号括起来的字符串：达到类似多行注释的效果
8  print "Hello,World!"
9  '''
```

2.1 变量赋值

Python中等号（=）是主要的赋值操作符，如：

```
>>> aInt = -12
>>> aString = "string"
>>> aFloat = -.23424
>>> aList = [1, 2, 'A', (1, 'B')]
```

```
>>> aInt
-12
>>> aString
'string'
>>> aFloat
-0.23424
>>> aList
[1, 2, 'A', (1, 'B')]
```

赋值与引用

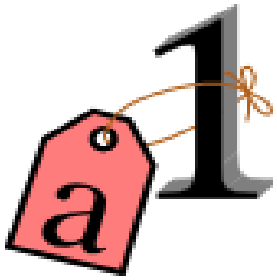
int a = 1;

在C中，变量的内存地址是不变的。

在Python中，对象是通过引用或标

a = 1

id(a)




```
>>> a = 1
>>> id( a )
140719373358752
>>> a = 10
>>> id( a )
140719373359040
>>> b = 1
>>> id( b )
140719373358752
```

a = 10

id(a)

对Python，变量自身是没有地址的

指向的数据在内存中的地址。

赋值与引用

请问：把a的第一个元素修改，会不会影响b？

```
>>> a = [1, 2, 3]
>>> id( a )
172196494912O
>>> b = a
>>> id( b )
172196494912O
```

2.2 增量赋值

等号可以和一个算术操作符组合在一起，将计算结果重新赋值给左边的变量，这被称为“增量赋值”。

如：x = x + 1 现在可以写成： x += 1

增量赋值通过使用赋值操作符，将数学运算隐藏在赋值过程中。

类似的增量赋值操作符还有：

+= -= *= /= **=

<<= >>= &= ^= |=

注意：Python不支持前/后置自增/减运算，比如：x++, --x

2.3 多目标赋值

- 一个对象的引用被赋值给多个变量。如：**x = y = z = 1**

```
>>> x = y = z = 1
>>> x
1
>>> y
1
>>> z
1
```

- **y = x = x + 1**　　（之前有**x =1**）

```
>>> x = 1
>>> y = x = x + 1
>>> x, y
(2, 2)
>>> x = 1
>>> x = x + 1
>>> y = x
>>> x, y
(2, 2)
```

2.5 多元赋值

- 将多个变量同时赋值的方法称为“多元赋值”，采用这种方式赋值时，等号右边的对象是序列。如：

```
>>> x, y, z = 1, 2, 'string'
>>> x
1
>>> y
2
>>> z
'string'
>>> x, y = y, x
>>> x
2
>>> y
1
```

3. 标准数据类型

Python主要有如下的标准数据类型：

- **Numbers（数字）**
- **String（字符串）**
- **List（列表）**
- **Tuple（元组）**
- **Dictionary（字典）**
- **Set（集合）**

3.1 数值类型

Python支持三种不同的数值类型：

- **int**（有符号整型）

  Python3整型理论上是没有限制大小的

- **float**（浮点型）

- **complex**（复数）

  复数由实数部分和虚数部分构成，可以用**a + bj**表示

3.1 数值类型

- 以0b开头的整数表示二进制整数：如：0b1000即8，各数字不能超过2，否则会出现错误。

```
>>> bin( 32 )
'0b100000'
```

- 以0o开头的整数表示八进制整数：如：0o41即33。
- 以0x开头的整数代表16进制整数：如0x61即97，同理各类似：0x10HI等也会出错。

```
>>> a = 0x61
>>> print( a )
97
```

3.1 数值类型

- **Python支持复数，复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示。**
  - **复数的实部a和虚部b都是浮点型。**
- **浮点数可以省略开头的0，如：0.1234 等价于 .1234**

```
>>> a = .1234
>>> a
0.1234
>>> a = 0.1234
>>> a
0.1234
```

练习：整数的操作–除法

```
>>>3/2

>>>3/2.0

>>>10/3

>>>10/3.0

>>>1/3.0

>>>1/3
```

```
>>> 3/2
1.5
>>> 3/2.0
1.5
>>> 10/3
3.3333333333333335
>>> 10/3.0
3.3333333333333335
>>> 1/3
0.3333333333333333
>>> 1/3.0
0.3333333333333333
```

3.2 字符串定义

Pyhton中的字符串被定义为引号之间的字符集合。

Python支持使用成对的单引号或者双引号，三引号（三个连续的单引号或者双引号）。

3.2.1 字符串定义

如果字符中已经包含了单引号，此时就不能再使用单引号来表示字符串，例如：

```
>>> '"Hello,world!" she said'
'"Hello,world!" she said'

>>> 'Let's go!'
SyntaxError: invalid syntax

>>> "Let's go!"
"Let's go!"
```

```
>>> 'Let\'s go!'
"Let's go!"
```

同样，如果字符串中已经有了双引号，则只能使用单引号来括起来。

此外，还可以使用反斜线（\）来对字符串的引号进行转义。

（\）起到转义作用，表示反斜线后面的单引号不是表示字符串结尾。

3.2.1 字符串定义-长字符串

Python中如果一个字符串包含很多行，则可以使用长字符串表示：长字符串使用三个引号（单双引号均可）代替普通引号。

```
>>> print( '''
        This is a very long string.
        It continues here.
        And it's not over yet.
        "Hello World!"
        Still here.''' )

        This is a very long string.
        It continues here.
        And it's not over yet.
        "Hello World!"
        Still here.
>>>
```

注意：长字符串中的字符可以直接使用单双引号而不用转义。

3.2.2 字符串操作–拼接字符串

- 一般连接字符串可以直接使用'+'号，例如：

```
>>> "Hello," + "World"
'Hello,World'
```

- 星号（*）可用于字符串重复

  Delim = '-' * 80

```
>>> Delim = '-' * 80
>>> Delim
'--------------------------------------------------------------------------------'
```

3.2.2 字符串操作–索引

字符串有其特有的索引规则：

第一个字符的索引是0，最后一个字符的索引是-1。

y = 'Hello World'

y[0]

y[-1]

```
>>> y = 'Hello World'
>>> y[ 0 ]
'H'
>>> y[ -1 ]
'd'
```

3.2.2 字符串操作-切片操作符

切片操作符（[开始索引 : 结束索引]）可以得到子字符串。其中“索引”是从0开始算起，可以是正数或负数，注意取值范围是一个半闭半开区间。

```
y = list(range(10))

y[1:5]

y[6:]       #结束索引为空表示取到尾

y[:-2]      #开始索引为空表示从头取
```

```
>>> y = list( range( 10 ))
>>> y
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> y[1 : 5]
[1, 2, 3, 4]
>>> y[6 : ]
[6, 7, 8, 9]
>>> y[ : -2]
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
>>> tag = '<a href="http://www.python.org">Python web site</a>'
>>> tag[9:30]
'http://www.python.org'
>>> tag[32:-4]
'Python web site'
```

3.2.2 字符串操作–成员操作符

字符串可用**in**或**not in**运算符来测试一个字符或一个子串是不是属于一个字符串。

'bc' in 'abcd'

'n' in 'abcd'

'nm' not in 'abcd'

```
>>> 'bc' in 'abcd'
True
>>> 'n' in 'abcd'
False
>>> 'nm' not in 'abcd'
True
```

3.2.4 字符串的格式化(%操作符)

如同C中的printf函数一样，可以用“%”来格式化字符串：

```
"I'm %d year old" % 19

"I'm %s year old" % 19   #自动转换

" I‘m %s. I’m %d year old" %  ('Vamei', 19)
```

其中，%s和%d部分称为转换说明符，标记了需要插入转换值的位置和格式。

3.2.3 字符串的格式化

格式	描述	%X	无符号整数(十六进制大写字符)
%%	百分号标记	%e	浮点数字(科学计数法)
%c	字符及其ASCII码	%E	浮点数字(科学计数法，用E代替e)
%s	字符串	%f	浮点数字(用小数点符号)
%d	有符号整数(十进制)	%g	浮点数字(根据值的大小采用%e或%f)
%u	无符号整数(十进制)	%G	浮点数字(类似于%g)
%o	无符号整数(八进制)	%p	指针(用十六进制打印值的内存地址)
%x	无符号整数(十六进制)	%n	存储输出字符的数量放进参数列表的下一个变量中

3.2.4 字符串的格式化

- 负号指示数字应该是左对齐的；
- “0”指示用前导0填充数字；
- 正号指示数字总是显示它的正负(+，-)符号，即使数字是正数也不例外；
- 可指定最小的字段宽度，如：“%5d” % 2；
- 可用句点符指定附加的精度，如：“%.3d” % 3；

3.2.4 字符串的格式化

```
from math import pi
print (pi)

print ('%f' % pi)

print ('%20f' % pi)
```

```
3.141592653589793
3.141593
            3.141593
3.141593
3.14159265358979311599796346854418516159057617187500000000000000
Guido
```

```
print ('%.5s' % 'Guido van Rossum')
```

3.2.5 字符串转义

在字符中使用特殊字符时，需要用反斜杠(\ )转义字符。

转义字符	描述	转义字符	描述
\(在行尾时)	续行符	\n	换行
\\	反斜杠符号	\v	纵向制表符
\'	单引号	\t	横向制表符
\"	双引号	\r	回车
\a	响铃	\f	换页
\b	退格(Backspace)	\oyy	八进制数yy代表的字符，例如：\o12代表换行
\e	转义	\xyy	十进制数yy代表的字符，例如：\x0a代表换行
\000	空	\other	其它的字符以普通格式输出

3.2.6 Unicode字符串

Unicode（统一码、万国码、单一码）是一种在计算机上使用的字符编码，为解决传统的字符编码方案的局限而产生的，它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

Unicode标准也在不断发展，但**最常用的是用两个字节表示一个字符**（如果要用到非常偏僻的字符，就需要**4**个字节）。现代操作系统和大多数编程语言都直接支持**Unicode**。

在Python3中，所有的字符串都是Unicode字符串。

3.2.6 Unicode字符串

对于单个字符的编码，Python提供了ord( )函数获取字符的整数表示，chr( )函数把编码转换为对应的字符。

```
>>> len('包含中文的str')
>>> ord('燚')
>>> hex(ord('瀞'))
>>> chr(25991)
```

```
>>> len('包含中文的str')
8
>>> ord('燚')
29146
>>> hex(ord('瀞'))
'0x7023'
>>> chr(25991)
'文'
```

3.2.7 原始字符串

不想让转义字符生效，只想显示字符串原来的意思，这就要用**r**和**R**来定义原始字符串（使用**r**/**R**作为前缀）。

```
>>> print( 'C:\now' )
C:
ow
>>> print( r'C:\now')
C:\now
```

原始字符串是一种特殊的字符串，它会将字符串中的内容原样保持，而不会进行任何的转义或者修改。

3.2.8 字符串表示 str、repr

一个值被转换成字符串有二种机制：**str( )**与**repr( )**。

str函数会把值转换为字符串，方便打印和查看。

repr函数会创建一个字符串，以**Python**表达式的形式表示值，便于开发者使用。

```
>>> temp=30    #要求：print('温度是:‘?’temp)
>>> print( '温度是:' + str(temp) )
温度是:30
>>> print('温度是:%d' % temp)
温度是:30
>>> print( '温度是:' + repr(temp) )
温度是:30
```

```
>>> import datetime
>>> today = datetime.date.today( )
>>> str( today )
'2020-09-14'
>>> repr(today)
'datetime.date(2020, 9, 14)'
>>> datetime.date(2020, 9, 15)
datetime.date(2020, 9, 15)
```

#返回值（去掉引号）直接复制到命令上，这种方法是可以直接执行的。

思考

- 变量赋值有几种方式?
- 标准数据类型有哪些?
- 字符串的特点?
  - 字符串类型函数以及内建函数