Python变量和数据类型

字符串

Python中对字符串的定义：

> Textual data in Python is handled with `str` objects, or **strings**. Strings are immutable sequences of Unicode code points.

Python中的文本数据是通过`str`对象或字符串来处理的，字符串是由一系列Unicode码位（code point）所组成的**不可变序列**。

```
('S' 'T' 'R' 'I' 'N' 'G')
```

Unicode 暂时可以看作一张非常大的地图，这张地图里面记录了世界上所有的符号，而码位则是每个符号所对应的坐标。

```
s = '春分'
print(s)
print(len(s))
print(s.encode())
```

使用内建函数`len()`可以获得字符串的长度。

不可变是指无法对字符串本身进行更改操作：

```
s = 'Hello'
print(s[3])
s[3] = 'o'
```

而**序列（sequence）**则是指字符串继承序列类型（`list/tuple/range`）的通用操作：

```
[i.upper() for i in 'Hello']
```

序列是容器类型，“成员”们站成了有序的队列，我们从0开始进行对每个成员进行标记，0, 1, 2, 3…，这样，便可以通过**下标**访问序列的一个或几个成员，就像C语言中的数组一样。接下来，我们先来了解一下序列。

序列类型操作符

注：以下操作符对所有序列类型都适用。

成员关系操作符（`in`、`not in`）

```
'x' in 'china'                            # 返回 False
'e' not in 'pity'                         # 返回 True
12 in [13, 32, 4, 0]                      # 返回 False
'green' not in ['red', 'yellow', 'white']  # 返回 True
```

连接操作符（`+`）

注：只可用于同种类型序列连接。

```
str1 = 'aaa'
str2 = 'bbb'
str2 = str1 + str2
str2              # 返回'aaabbb'，此时str2所指向的对象是新创建的对象
                  # 因字符串是不可更新的标量，可以用id()测试
```

```
num_list = [1, 3, 5]
num_list += [7, 9]
num_list          # 返回[1, 3, 5, 7, 9]，此时的num_list指向的对象还是
                  # 原始对象，因其是可更改的容器，可以用id()测试
```

```
(1, 3) + (5, 7)  # 返回(1, 3, 5, 7)，注意元组是不可更改的容器
```

重复操作符（`*`）

用法：`s * n` 或 `n * s`

`*`用以将序列重复指定次数，如：

```
astr = 'hello'
astr *= 3
astr               # 返回'hellohellohello'
```

```
alpha_list = ['a', 'b', 'c']
alpha_list *= 2
alpha_list         # 返回['a', 'b', 'c', 'a', 'b', 'c']
```

```
('ha', 'ya') * 3  # 返回('ha', 'ya', 'ha', 'ya', 'ha', 'ya')
```

当`n`的值小于0的时候都按照`n = 0`对待（结果将返回一个和`s`类型相同的空序列）。

```
'a' * -2
```

另外需要注意的是序列`s`中的元素并没有被复制，它们只是被引用了多次。这个需要Python初学者特别注意，比如：

```
lists = [[]] * 3
lists
```

```
lists[0].append(3)
lists
```

`[[]]`是一个单元素列表，包含一个空的列表，所以`[[]] * 3`中的3个元素都引用这个空的列表。修改其中的任意一个元素都会修改这个空的列表。你可以用下面的方式创建一个不同列表构成的列表：

```
lists = [[] for i in range(3)]
lists[0].append(3)
lists[1].append(5)
lists[2].append(7)
lists
```

更多的解释可以参考Python官方文档：How do I create a multidimensional list?

切片操作符（`[]`、`[:]`、`[::]`）

通过切片功能可以访问序列的一个或者多个成员。和C一样，在访问单个成员时你要保证你访问下标的成员是存在的，否则会引发`IndexError`异常（C中叫做数组越界）。

索引——访问单个成员`[]`

```
astr = 'Python'
astr[0]
astr[3]
```

```
astr[6]
```

```
astr[-1]  # 'n'
astr[-6]  # 'P'
```

注意，因为`-0`等于`0`，负数的索引从`-1`开始。

切片——访问连续的多个成员`[starting_index : ending_index]`

切片索引有默认值，默认的起始索引是`0`，默认的终止索引是所要切片的字符串的长度。

```
astr[:]     # 'Python'
astr[0:]    # 'Python'
astr[:6]    # 'Python'
astr[:5]    # 'Pytho'
astr[5:]    # 'n'
astr[:-1]   # 'Pytho'
astr[1:-1]  # 'ytho'
```

注意起始索引是包含进来的，终止索引是排除在外的。所以，`s[:i] + s[i:]`**永远等于**`s`。

```
astr[:2] + astr[2:]
astr[:4] + astr[4:]
```

记住切片如何工作的一种方法是将索引看作是字符间的点，第一个字符的左侧的位置为`0`，最后一个字符的右侧的位置为字符的长度。比如：

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

另外需要注意的是，当使用切片访问连续的多个成员时超出索引范围将被很好的处理。

以等差数列形式的下标进行访问`[starting_index : ending_index : step_length]`

```
(1, 2, 3, 4, 5, 6)[0:6:2]  # 返回 (1, 3, 5)
bstr = "abcdefg"
bstr[::-1]    # 返回'gfedcba'，瞬间反转，未添加的参数默认为开始和结束
bstr[::]      # 返回'abcdefg'，未添加的参数都使用默认值
```

用于序列的内建函数

- `max()` 返回序列中的最大值
- `min()` 返回序列中最小值
- `sum()` 返回列表的元素之和
- `enumerate(iter)` 接受一个可迭代对象，返回一个enumerate对象，该对象生成iter的每个成员的index值和item值构成的数组
- `reversed(seq)` 返回一个序列的逆向迭代器
- `sorted()` 对一个序列，排序，返回排好序的列表，可以指定排序方法
- `zip()` 返回一个zip对象，其成员为元组

```
list_demo = [1, 43, 4, 54]
max(list_demo)             # 54
min(list_demo)             # 1
sum(list_demo)             # 102
list(reversed(list_demo))  # [54, 4, 43, 1]
sorted(list_demo)          # [1, 4, 43, 54]
list(zip(list_demo, list_demo[::-1]))  # [(1, 54), (43, 4), (4, 43), (54, 1)]
```

上面是对序列简单的介绍，接着我们回到字符串上来。

字符串的创建

3种方式创建字符串字面量：

1. 单引号：`'allows embedded "double" quotes'`
2. 双引号：`"allows embedded 'single' quotes"`
3. 三引号：`'''Three single quotes'''` `"""Three double quotes"""`

其中，三引号创建的字符串可以跨越多行，其中的空白（例如每行的换行符以及行首或行末的空格）会被包含进所创建的字符串字面量。

Python允许空字符串`''`，它不包含任何字符但完全合法。空字符串是其他任何字符串的子串。

字符串字面量是一个单独的表达式，如果多个字符串字面量中间仅包含空白，则它们将被隐性地转换为一个单一的字符串字面量。所以，`("spam" "eggs") == "spameggs"`。

```
url_str = ('http://'       # protocol
           'localhost'     # hostname
           ':8080'         # port
           '/index.html')  # file
url_str  # 'http://localhost:8080/index.html'
```

另外，你还可以使用`str`构造器将其它对象转换为字符串。

```
str(98.6)  # '98.6'
str(True)  # 'True'
```

使用`\`转义

常见的转义符：`\n`（换行符）、`\t`（Tab制表符）、`\r`（回车）、`\'`（单引号）、`\"`（双引号）、`\\`（反斜线）

字符串操作符

参考上面**序列类型操作符**，不再赘述。

用于字符串的内建函数

- `input()` 获取用户输入，返回一个字符串

```
name = input("What's your name: ")
print("Your name is %s." % name)
```

```
ord(chr(64))
```

```

dir(str)
```

- `chr()` 接受一个整数，返回对应的Unicode字符

- `ord()` 功能与`chr()` 相反

字符串方法

用法：`string_object.method(arguments)`

字符串方法比较多，可以通过`help(str)`或者`dir(str)`获取帮助。

以下是一些常用的方法：

- `split()` 基于**分隔符**将字符串分割成由若干子串组成的列表，如果不指定分割符，默认使用空白字符进行分割。
- `join()` 与`split()`功能相反，将包含若干子串的列表分解，并将这些子串通过指定的**粘合用的字符串**合并成一个完整的大的字符串。
- `find()` 查找返回字符串中第一次出现子串的位置（偏移量），失败时返回`-1`。
- `index()` 与`find()`类似，但是查找失败时将触发`ValueError`异常。
- `rfind()` 与`find()`类似，但返回最后一次子串出现的位置。
- `startswith()` 判断字符串是否以特定前缀开头。
- `endswith()` 判断字符串是否以特定后缀结尾。
- `count()` 统计子串在字符串中出现的次数。
- `is*` 判断字符串中字符是否符合某种类型或者规则。
- `strip()` 返回移除开始和结尾空白字符的字符串，如果指定参数，将在字符串的开始和结尾移除参数中所包含的字符。
- `upper()` `lower()` `swapcase()` 分别将字符串所有字母转换成大写、转换成小写、大小写转换。
- `title()` 将字符串中所有单词的开头字母变成大写。
- `capitalize()` 将字符串首字母变成大写。
- `center()` `ljust()` `rjust()` 分别将字符串根据指定长度居中对齐、左对齐、右对齐。
- `replace()` 进行简单的子串替换，需要传入的参数：需要被替换的子串，用于替换的新子串，以及需要替换多少处。

```
s = "Hello, world!"
s.split()  # ['Hello,', 'world!']
' '.join(s.split())  # s.split()
s.find('world')  # 7
#s.index('~')  # will cause ValueError
s.is
```

```
s.rfind('o')  # 8
s.startswith('m')  # False
s.endswith('!')  # True

s.isalnum()  # False
s.isdigit()  # False
s.islower()  # False
s.istitle()  # False
s.isalpha()  # False

s.strip('!d')  # 'Hello, worl'

s.upper()     # 'HELLO, WORLD!'
s.lower()     # 'hello, world!'
s.swapcase()  # 'hELLO, WORLD!'

s.title()     # 'Hello, World!'
s.capitalize() # 'Hello, world!'

s.center(20)  # '   Hello, world!    '
s.ljust(20)   # 'Hello, world!       '
s.rjust(20)   # '       Hello, world!'

s.replace('o', 'O', 1)  # 'HellO, world!'
```