Python 程序设计 - 序列（sequences）

李正华

November 28, 2016

1 序列类型

序列：按顺序、依次存储的数据类型，可以根据下标访问。

0, ..., N-1

-N, ..., -1

适用于序列的操作符：

- 成员关系（in, not in）
- 连接操作符（+）
- 重复操作符（*）
- 元素访问操作符（[]），返回元素

- 切片操作符 ([:],[::]), 返回序列

```
s = `abcdefgh'

print(s[::-1])

for i in range(-1, -len(s), -
                        1):

  print s[:i]
```

如何打印所有的元素呢？

几个概念：

- 序列类型
- 可迭代对象（提供迭代器方法）

类型转换（本质上是工厂函数）：

- list(iter)：如果把一个列表对象传递给list(),那么会创建这个对象的“浅拷贝”，然后将其插入到新的列表中（这句话什么意思？好好理解，尝试一下。此处的

浅拷贝是什么意思?)。对于连接操作(+)和重复操作(*),也是类似的。(怎么理解?)

- str(obj)
- tuple(iter)

可操作 BIF:

- enumerate(iter)
- len(seq)
- max(iter, key=None) 或 max(arg0, arg1,

..., key=None)：key 必须是一个可以传给 sort 方法的，用于比较的回调函数。

- min(...)
- sum(seq, init=0)
- reversed(seq)：返回一个迭代器，如：for i in reversed([0,1,2]): print(i)
- sorted(iterable, key=None, reverse=False)：返回一个有序的列表
- zip(iter1 [, iter2 [...]])：返回一个列表，其中第一个元素是由 iter1, iter2 等这些对

象第一个元素构成的元组，...

2 字符串

- 字符串创建和赋值
- 访问字符串和子串（下标和切片：返回对象类型?)
- 如何改变字符串?
- 如何删除字符和字符串

操作符：

- 赋值、增量赋值
- 关系运算符：==, <
- 下标，切片
- 成员判断 in, not in？判断什么呢？

- 连接（concatenation）：运行时，编译时
- 重复
- 格式化操作符（%）：format_string % (arguments_to_convert)

```
%c

%s

%d or %i

%o

%x or %X
```

```
%e or %E

%f or %F

%g or %G

%%

*

-
```

```
+

<sp>

#

0

(var)

m.n
```

```
print('%x' % 108)

print('%X' % 108)

print('%#X' % 108)

print('%#x' % 108)



print('%f' % 1234.567890)
```

```
print('%.2f' % 1234.567890)

print('%E' % 1234.567890)

print('%e' % 1234.567890)

print('%G' % 1234.567890)

print('%g' % 1234.567890)

print('%g' %
```

```
                    1111111111111111111
                    )


print('%+d' % 4)

print('%+d' % -4)
```

```
print('MM/DD/YY = %02d/%02d/%
                    02d' % (2, 15,
                      67))


print('There are %(howmany)d
                     %(lang)s
```

```
                Quotation
                Symbols' % {'
                lang': 'Python
                ', 'howmany' :
                 3})
```

字符串方法

```
string.capitalize()

string.center(width) # 空格填
                       充两边

string.count(str, beg = 0,
                    end = len(
                    string))
```

```
string.decode(encoding='UTF-8
                        ', errors='
                        strict')
string.encode(encoding='UTF-8
                        ', errors='
                        strict')
```

```
string.endswith(suffix, beg=0
                      , end=len(
                      string))
string.find(str, beg=0, end=
                      len(string)) #
                        不存在则返回-
```

```
                    1

string.index(str, beg=0, end=
                    len(string)) #
                      不存在则抛出
                      异常
string.isalnum() # 至少一个字
```

符，且所有字符

都是字母或数字

```
string.isalpha()

string.isdecimal()

string.isdigit()

string.islower()
```

```
string.isnumeric()

string.isspace()

string.isupper()

string.join(seq)

string.ljust(width) # 左对
                    齐，空格填充
```

```
string.lower()
string.lstrip([chars]) # 删除
                        左边的空格（还
                        是空白符？）
string.repace(str1, str2, num
                        =string.count(
```

```
                    str1))
string.rfind(str, beg=0, end=
                    len(string)) #
                      从右查找
string.rindex(str, beg=0, end
                    =len(string))
```

```
string.rjust(width)

string.rstrip([chars])

string.split(str='', num=
                    string.count(
                    str))

string.splitlines(num=string.
```

```
                    count('\n'))
string.startswith(prefix, beg
                    =0, end=len(
                    string))
string.strip([chars])
string.swapcase()
```

```
string.upper()

string.zfill(width) # 右对
                    齐，前面填充0
```

3 Lists

How to Create and Assign Lists

How to Access Values in Lists

How to Update Lists

.append()

How to Remove List Elements and Lists

del 语句, .remove(), .pop()

```
x = ['a', 23, 'b']

del x[0]

x.pop()

x.pop(0)

x.remove(23)
```

```
del x
```

Operators

- Standard Type Operators: comparison
- Sequence Type Operators: slicing, membership, concatenation, repetition
- List Type Operators and List Comprehensions

```
[ i * 2 for i in [8, -2, 5]
```

```
                    ]

[ i for i in range(8) if i

                    % 2 == 0 ]
```

Built-in Functions

- Standard Type Functions
- Sequence Type Functions: len(), max(), min(), sorted(), reversed(), enumerated(),

zip(), sum(), list(), tuple()

- List Type Built-in Functions

List Type Built-in Methods

```
list.append(obj)

list.count(obj)

list.extend(seq)

list.index(obj, i=0, j=len(
```

```
                    list))

list.insert(index, obj)

list.pop(index=-1)

list.remove(obj)

list.reverse()

list.sort(func=None, key=None
```

```
, reverse=
False) #
optional
comparison
function; key
is a callback
```

```
when
extracting
elements for
sorting
```

4 Tuples

How to Create and Assign Tuples

How to Access Values in Tuples

How to Update Tuples

How to Remove Tuple Elements and Tuples

del 语句

Tuple Operators and Built-in Functions

- Standard and Sequence Type Operators and Built-in Functions
  - Creation, Repetition, Concatenation
  - Membership, Slicing
  - str(), max(), min(), list()
  - Comparison
- Tuple Type Operators and Built-in Functions and Methods
- .count(), .index()

How Are Tuples Affected by Immutability?

- How to understand that tuples are immutable?
- Immutability does not necessarily mean bad news.

Tuples Are Not Quite So "Immutable"

- Although tuple objects themselves are immutable, this fact does not pre- clude tuples from containing mutable objects that can be changed.

Tuples Are The Default Collection Type

```
def foo1(): :

  return obj1, obj2, obj3

def foo2(): :

  return [obj1, obj2, obj3]

def foo3(): :
```

```
return (obj1, obj2, obj3)
```

Single-Element Tuples

```
x = ['abc']

x = ('abc')

x = ('abc', )

x = 'abc',
```

5 Copying Python Objects and Shallow and Deep Copies

```
person = ['name', ['savings',
                   100.00]]
```

```
hubby = person[:] # slice
                  copy
wifey = list(person) # fac
                     func copy
[id(x) for x in person, hubby
                      , wifey]
```

```
hubby[0] = 'joe'

wifey[0] = 'jane'

print(hubby, wifey)


hubby[1][1]  = = 50.00
```

```
print(hubby, wifey)


[id(x) for x in hubby]

[id(x) for x in wifey]


hubby = person
```

```
import copy
wifey = copy.deepcopy(person)
[id(x) for x in person, hubby
                      , wifey]
hubby[0] = 'joe'
```

```
wifey[0] = 'jane'

print(hubby, wifey)



hubby[1][1]  = = 50.00

print(hubby, wifey)
```

6 Summary of Sequences

Sequence Type Operators, Built-in Functions and Methods

```
[]

()

'  ', "  ", ''' '''
```

```
capitalize(), center(),
                endswith(),
                startswith(),
                isdecimal(),
                isspace(),
```

```
ljust(), lower
(), strip(),
replace(),
split(),
splitlines(),
swapcase()...
```

```
decode(), encode()

hex(), chr(), oct(), ord()

find(), index()

append(), pop(), remove(),
                 reverse(),
                 sort(),
```

```
count()

extend()

insert()

len()

list(), tuple(), str(), type
                            ()
```

```
max(), min(),

input()

. (attributes)

[] (slice)

[:]

*
```

```
%

+

in, not in
```