字符串

```
s = 'string'

s = s + '123'
# 字符串连接 'string123'

s = s * 3
# 字符串复制'string123string123string123'
```

字符串类型是不可变类型，一旦定义不能更改，这意味着不能通过所谓对某一位置进行赋值而改变字符串。

```
s[0]='a'
```

> 报错
>
> TypeError: 'str' object does not support item assignment

对字符串的单个位置赋值会报错，但我们仍能通过其他办法来实现目的。

```
s = 'a' + s[1:]
# 'atring123string123string123'
```

实际上就是定义了一个与原来字符串名字相同的新字符串（原来的字符串自动被删除了）。

字符串对象的find方法提供了寻找子串首次出现位置的功能。

```
s.find('tr')
# 1
```

字符串对象的replace方法提供了查找替换的功能。

```
s.replace('g','G')
# 'atrinG123strinG123strinG123'
```

字符串对象的split方法提供了按分隔符分割字符串的功能。

```
line = 'a,b,C,D'
line.split(',')
# ['a', 'b', 'C', 'D']
```

字符串的upper和lower方提供了将字符串转为全为大写和小写的功能。

```
line.upper()
# 'A,B,C,D'

line[0]
# 'a'

line.lower()
# 'a,b,c,d'

line[4]
# 'C'
```

特别注意的是转换大小写的过程是生成了一个新字符串，原字符串是不改变的（因为字符串的不可变性），所以line[0]与line[4]依然和原来一样。

如果想知道更多关于字符串对象的属性与方法，可以使用如下语句：

```
dir(s)

# ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',……]
```

语句执行之后将会以list的形式列出所有的属性方法。

如果想知道某个属性或者方法具体有哪些参数，如何使用，可以使用如下语句（以split为例）：

```
help(s.split)
```

```
split(...) method of builtins.str instance
    S.split(sep=None, maxsplit=-1) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
    removed from the result.
```

列表

列表比其他语言的数组更强大，因为其可以存储不同类型的数据。

```
L1 = [123,'qwer',12.3]
```

列表索引比其他语言强大之处在于支持逆向索引。

```
L1[0]
# 123

L1[-1]
# 12.3

L1[-2]
# 'qwer'
```

列表常见操作与方法

```
L2 = [2,1,3]
L3 = L1 + L2
# [123, 'qwer', 12.3, 2, 1, 3]

L3.append(('zxcv'))
# [123, 'qwer', 12.3, 2, 1, 3, 'zxcv']

L3.pop((5))
# 返回删除值3，删除后L3=[123, 'qwer', 12.3, 2, 1, 'zxcv']

L2.sort()
# [1,2,3]

L3.reverse()
# [3, 2, 1]

L3.remove('zxcv')
# [123, 'qwer', 12.3, 1, 2]

L3.insert(5,'insert')
# [123, 'qwer', 12.3, 1, 2, 'insert']
```

注意： append方法一次只能在列表最后添加一个元素，如果传入一个列表或者其他维度的数据，会被当作一个。

```
L1.append(([1,2,33]))
# [123, 'qwer', 12.3, [1, 2, 33]]
```

由于列表是可变对象，append方法是对列表进行直接修改，而不是产生一个复制。所以不要再进行如下所示的错误赋值语句（append方法会返回一个None对象，赋值之后列表就会变为None对象）

```
L1 = L1.append(([1,2,33]))
# 错误语句
```

另外不要忘记少写append(())的括号，四个括号必不可少。

pop方法是按照索引进行删除，L3.pop(5)表示删除L3列表的第6个值；remove方法是按照值进行删除，L3.remove('zxcv')是删除列表中值为'zxcv'的元素。

sort方法默认按照升序排列，若要降序排列，置参数reverse为True，即：L2.sort(reverse=True)。

insert方法第一个参数为插入的位置索引，第二个参数为插入值。

列表嵌套构造矩阵

```
L4 = [[1,2,3],[2,3,4],[3,4,5]]
```

此处列表中两个元素均为列表，每个列表又包含了3个元素。

嵌套列表如何索引

```
L4[1][2]
# 4

L4[1,2]
# 错误
```

此处留心与以后学习的numpy矩阵的索引进行对比（numpy支持两种索引，list仅支持前者）

列表解析

```
col2 = [row[1] for row in L4]
# [2,3,4]
```

代码能获取列表的第2列。表达式可以这样理解：将L4的每一行(row)的第2个元素放到一个新列表里。 事实上，列表解析可以更简单地实现复杂的功能。

```
L5 = [row[1] + 1 for row in L4]
# [3,4,5]

L6 = [row[1] for row in L4 if row[1] % 2 == 0]
# [2,4]
diag = [L4[i][i] for i in [0,1,2]]
# [1, 3, 5]
```

```
doubles = [c * 2 for c in 'student']
# ['ss', 'tt', 'uu', 'dd', 'ee', 'nn', 'tt']
```

L5是由L4第2列的所有元素+1得到；L6是过滤掉L4第2列中的奇数得到；diag是取出L4的对角线元素；doubles是将字符列表['s','t','u','d','e','n','t']每个字符数量变成两倍得到。

字典

字典是一种映射，反映一种一一对应关系。当你查字典时（根据键查找），会根据这种一一对应关系返回相应的值。字典数据用{}包围。

创建字典

方法一

```
Dict = {'name':'Richard','age':21,'sex':'male'}
# 创建字典

Dict['name']
# Richard

Dict['age']
# 21
```

方法二

```
keylist = ['name','age','sex']
valuelist = ['Rice',22,'female']
Dict = dict(zip(keylist,valuelist))

Dict['name']
# Rice

Dict['age']
# 212
```

字典常用方法

```
D.keys()
# 查询字典所有键

D.values()
# 查询字典所有值

D.items()
# 查询字典所有键值对
```

```
D.copy()
# 得到字典的副本

'name' in D
# 查询键是否存在于字典中

D1.update(D2)
# 合并

D.pop(key)
# 删除

len(D)
# 长度

del D[key]
# 根据键删除条目
```

字典解析

```
D = {x:x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

D = {'student%d' % x:x**2 for x in range(5)}
# {'student0': 0, 'student1': 1, 'student2': 4, 'student3': 9, 'student4': 16}
```

字典的属性

```

```

- 无序集合

字典输出结果与键值对的顺序没有关系。

```
D = {'age':21,'name':'Richard','sex':'male'}
D
# {'age': 21, 'name': 'Richard', 'sex': 'male'}

D = {'name':'Richard','age':21,'sex':'male'}
D
# {'age': 21, 'name': 'Richard', 'sex': 'male'}
```

如果采用print(D)的形式，将会原样输出字典，这意味着这种情形下输出结果是与键值对的顺序有关的。

- 支持嵌套

这意味着键值对中的“值”可以是其他数据类型，比如列表，字符串，字典等。

```
D = {'name':['Richard','Rice'],'age':[21,22],'sex':['male','female']}
```

修改字典

按照赋值的思想去修改即可。

```
D = {'name':'Richard','age':21,'sex':'male'}
D['name'] = 'Rice'
D
# {'age': 21, 'name': 'Rice', 'sex': 'male'}
```

字典的妙用

字典本身是用来索引的，这里谈一些其他的用处。

- 模拟列表

列表的索引不能超出索引范围，所以我们的程序经常会出这个bug。但是字典自动为新索引添加键值对，从而免去错误。

```
D = {}
D[99] = 'student'
# {99: 'student'}
```

此处有几个细节：（1）字典的键不一定是字符串，可以是任何不可变的数据类型。所以当使用整数作为键时，使其在引用时与列表引用很相似。（2）如果开始D的数据类型是列表，那么程序就会报超出索引范围的错误。而字典会自动添加为新键添加

- 字典用于稀疏结构

```
matrix = {}
matrix[(1,2)] = 1
matrix[(3,3)] = 1
matrix[(2,3)] = 1
```

4 x 4 的矩阵仅有三个元素，不妨采用字典来刻画。

- 避免 missing-key 的情况

```
# 方式1

if (1,2) in matrix:
```

```
    print(matrix[(1,2)])
else:
    print(0)

# 方式2

try:
    print(matrix[(1,2)]
except KeyError:
    print(0)

# 方式3

print(matrix.get([1,2],0))
```

方式