NLP 南京大学 自然语言处理研究组 Natural Language Processing Group

南京大學 NANJING UNIVERSITY

Python复合数据类型

黄书剑

复杂（复合）数据类型

- **列表 list**
  - `[1, 2, 3]  ['Monday','Tuesday','Thursday']`
- **元组 tuple**
  - `('Tim','Peters') ('Nanjing', 'Qixia', 'Xianlin Avenue', '163')`
- **range对象**
  - range(1,5) ：1到4组成的序列
- **字典 dict**
  - `{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}`
- **集合 set/frozenset**
  - `{1, 2, 3}`

列表

列表

- **按照顺序排列的元素组成的数据类型**
  - languages=[‘C’, ‘C++’, ‘Java’, ‘Python’]
  - num = list(range(1, 5))   #range还能生成更复杂的序列
- **可以按照次序访问对应的元素（下标、索引）,每个元素也可以看做是一个变量**
  - **下标为负数**表示列表结尾开始进行索引
  - languages[0], languages[**-1**]

列表

- **列表中元素的类型不一定相同**

```
>>> langFamilies = [ ['C', 'C++'], 'Java', 'Python']
>>> langFamilies[0][1]
C++
```

- **列表的元素仍然可以为列表（二维列表、高维列表）**
  - [[1,2,3,4], [5,6,7,8], [9,10,11,12]]
  - matrix[0] :   [1,2,3,4]   matrix[0][1]:   2

- **列表方便地把不同的值组合成一个序列（对值的索引）**
  - 与数组不同（元素类型可以不同、下标访问更为自由）

列表操作

- 定义列表
- 修改列表元素
- 使用列表
  - 列表比较、遍历列表
- 创建新的列表
  - 列表解析、列表切片、range对象等

列表元素操作

- **修改**
  - languages[0] = ‘Perl’
- **添加**
  - languages.append(‘Go’)
  - languages.insert(2, ‘Scheme’)
- **删除**
  - languages.pop()
  - languages.pop(1)   del langauges[1]
  - languages.remove(‘Go’)

列表的比较 （对象比较 v.s. 值比较）

- 列表的值比较结果为其逐元素值比较的结果

```
>>> y = [1,2,3,4]
>>> x = [1,2,3,4]
>>> x == y
True
>>> x is y
False
```

```
>>> id(y)
140502903818592
>>> id(x)
140502903496944
```

```
>>> x = [1,2,3,4]
>>> y = [1,2,3,5]
>>> x < y
True
```

```
>>> x < 5
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: '<' not supported
between instances of 'list' and
'int'
```

列表整体操作

- **列表自带功能函数 sort() reverse() extend() 等**

改变列表的值

```
>>> x = [3, 9, 1, 7]
>>> x.sort()
>>> x
[1, 3, 7, 9]
```

- **可以用build-in函数操作列表 print() len() sorted() reversed()**

生成新的列表

```
>>> sorted([3, 9, 1, 7])
[1, 3, 7, 9]
```

```
>>> reversed([3, 9, 1, 7])
<list_reverseiterator object at 0x7fc961b154d0>
>>> list(reversed([3, 9, 1, 7]))
[7, 1, 9, 3]
```

列表参与运算

- 成员判断（in）

```
>>> 3 in [1, 3, 4]
True
```

- 序列连接（+）

```
>>> [2, 8 ,6] + [1, 3, 4]
[2, 8, 6, 1, 3, 4]
```

- 序列重复（*）

```
>>> [1, 3, 4] * 3
[1, 3, 4, 1, 3, 4, 1, 3, 4]
>>> 3 * [1, 3, 4]
[1, 3, 4, 1, 3, 4, 1, 3, 4]
```

生成新的列表

列表整体操作

- 对数字列表做简单的统计

```
>>> digits = [1,3,16,5,9]
>>> min(digits)
1
```

```
>>> sum(digits)
34
```

- 对字母、字符串也可以进行类似操作

```
>>> max(["c++","java","python"])
'python'
```

遍历列表元素

- **for循环语句**
  - 注意不是按照下标进行遍历，而是由解释器自行遍历

```
>>> for x in list(range(1,5)):
...     print(x*x)
...
1
4
9
16
```

```
>>> y = list(range(1,5))
>>> for i in range(0,4):
...     print(y[i]**2)
...
1
4
9
16
```

声明式风格v.s. 命令式风格

遍历列表元素

- **for循环语句**
  - 注意不是按照下标进行遍历，而是由解释器自行遍历

```
>>> for lan in languages:
...     print(f"Please learn {lan} with patience!")
...
Please learn ['C', 'C++'] with patience!
Please learn Java with patience!
Please learn Python with patience!
```

f"string1 {exp} string2" 是一种格式化字符串操作，在字符串求值时对exp表达式求值

列表解析（list comprehension）

- **创建一个新列表的方法：对已有元素序列依次进行操作**
  - 比循环更简洁的使用方式

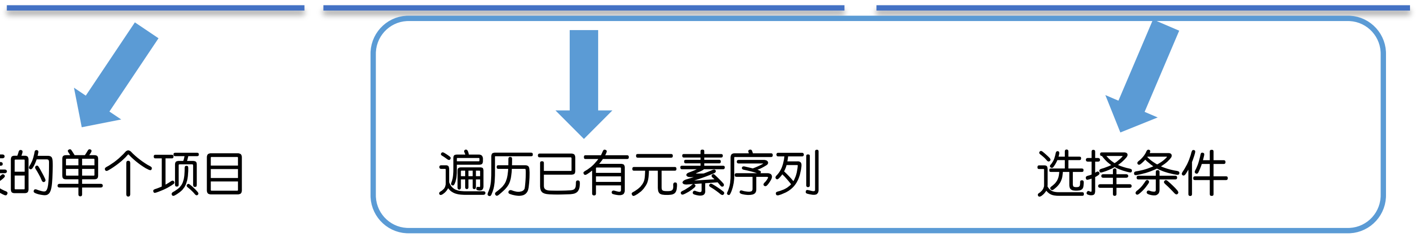
```
>>> squares = [ value**2 for value in [1,2,3,4] ]
>>> squares
[1, 4, 9, 16]
```

```
>>> squares = [ value**2 for value in range(1,5) if value%2==0 ]
>>> squares
[4, 16]
```

列表解析（list comprehension）

- **语法说明：**

```
newlist = [expression for item in iterable if condition == True]
```



列表解析（list comprehension）

- 组合打印两个列表[1,2,3]，[3,1,4]:

```
>>> [print(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

```
>>> z = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

  – 循环版本：

```
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...             if x!= y:
...                     print(x,y)
...
```

```
1 3
1 4
2 3
2 1
2 4
3 1
3 4
```

列表解析（list comprehension）

- 可以有多个for语句，从左到右依次执行

```
>>> matrix
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> [num for row in matrix for num in row]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

- 可以有复杂的条件判断表达式

```
>>> ["Even" if i%2==0 else "Odd" for i in range(6)]
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

```
>>> [y for y in range(100) if y % 2 == 0 if y % 7 == 0]
[0, 14, 28, 42, 56, 70, 84, 98]
```

嵌套的List Comprehension

- **操作高维列表(矩阵转置)**

```
>>> matrix = [ [1,2,3,4], [5,6,7,8], [9,10,11,12] ]
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```
>>> transposed = []
>>> for i in range(4):
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
```

```
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
```

```
>>> list(zip(*matrix))
```

*

- **zip函数**
  - help(zip)

```
class zip(object)
 |  zip(*iterables) --> A zip object yielding tuples until an input is exhausted.
 |
 |     >>> list(zip('abcdefg', range(3), range(4)))
 |     [('a', 0, 0), ('b', 1, 1), ('c', 2, 2)]
 |
 |  The zip object yields n-length tuples, where n is the number of iterables
 |  passed as positional arguments to zip().  The i-th element in every tuple
 |  comes from the i-th iterable argument to zip().  This continues until the
 |  shortest argument is exhausted.
```

*星号内容为补充内容，不作为此处课程要求，后同

*

- *** 操作符 和 unpacking**
  - *操作符用来标识和unpacking相关的操作

```
>>> matrix
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>> [1, *matrix, 2]
[1, [1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], 2]
```

  - 关于packing/unpacking在元组部分还会介绍
  - 更详细的介绍可参考：https://stackabuse.com/unpacking-in-python-beyond-parallel-assignment/

列表切片（slice）

- 按照下标从列表中取出多个元素，构成新的列表

```
>>> listA = ['a','b','c','d','e','f','g']
>>> print(listA[2:3], listA[1:6], listA[3:])
['c'] ['b', 'c', 'd', 'e', 'f'] ['d', 'e', 'f', 'g']
```

列表切片（slice）

- **语法说明**

```
newlist = listA[start : end : step]
```

开始位置　　结束位置　　步长

- 结束位置元素**不在**结果列表中
- 步长用于跳过部分元素
- start、end 可以省略，分别表示从列表开始、直到列表结束
- step可以省略，表示默认步长为1

列表切片（slice）

- **省略 start 或 end**

```
>>> listA[-2:]
['f', 'g']
>>> listA[:-2]
['a', 'b', 'c', 'd', 'e’]
>>> listA[::]
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

```
>>> id(listA[::])
140502904181856
>>> id(listA)
140502903496784
```

- **步长为负数表示倒序**

```
>>> listA[::-1]
['g', 'f', 'e', 'd', 'c', 'b', 'a']
>>> listA[-2::-1]
['f', 'e', 'd', 'c', 'b', 'a']
```

列表切片（slice）

- **切片结果由start、end和step共同决定**
  - 所选区域内无元素则范围空列表

```
>>> listA[-2:-5]
[]
>>> listA[-2:-5:-1]
['f', 'e', 'd']
```

列表的复制

- **copy**

```
>>> a1 = [1, 2, 3]
>>> b = a1.copy()
>>> b
[1, 2, 3]
```

```
>>> a2 = [1, 2, [3, 4]]
>>> b = a2.copy()
>>> a2 is b
False
>>> a2[2] is b[2]
True
```

- **deep copy**

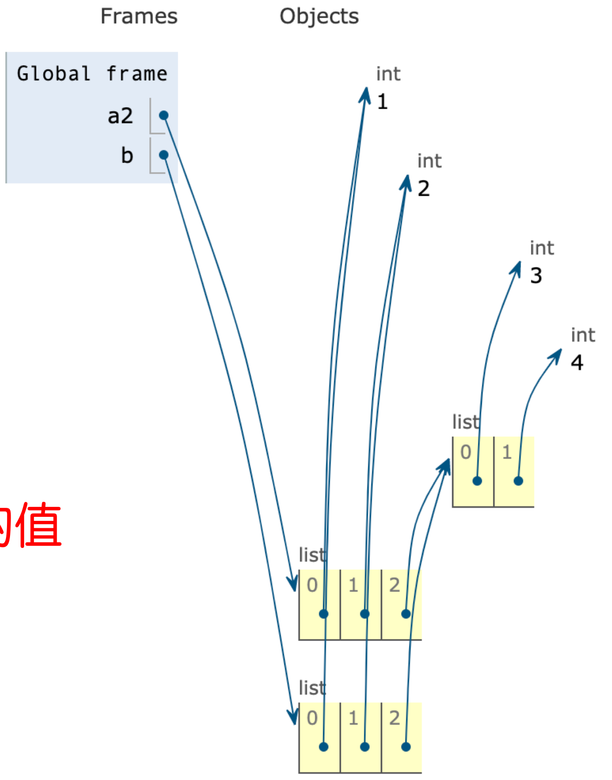
```
>>> from copy import deepcopy
>>> c = deepcopy(a2)
>>> c is a2
False
>>> c[2] is a2[2]
False
```

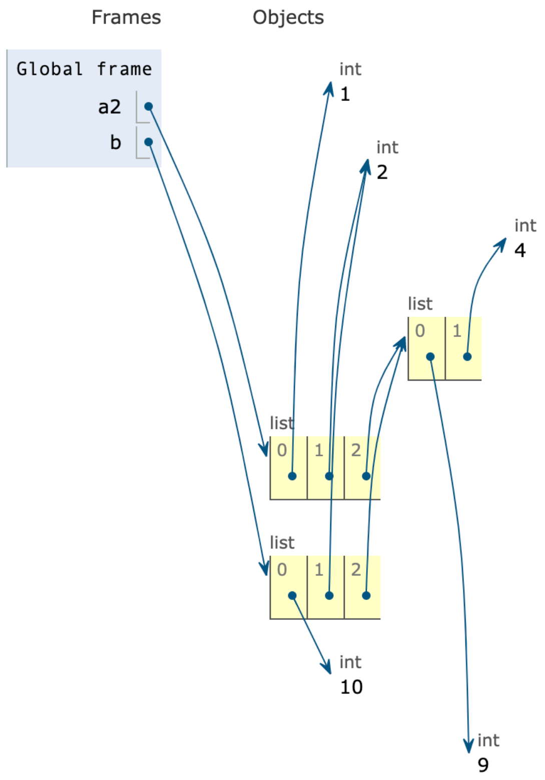
Shallow copy v.s. Deep copy

- **Shallow copy**

```
a2 = [1, 2, [3, 4]]
b = a2.copy()
b[0] = 10
b[2][0] = 9
```

b[2][0]的修改会影响a[2][0]的值





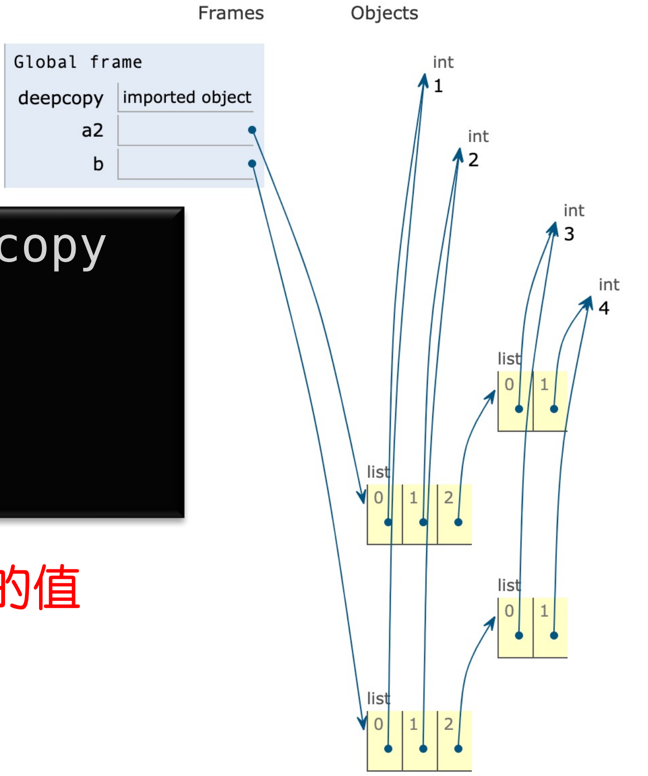
http://pythontutor.com/

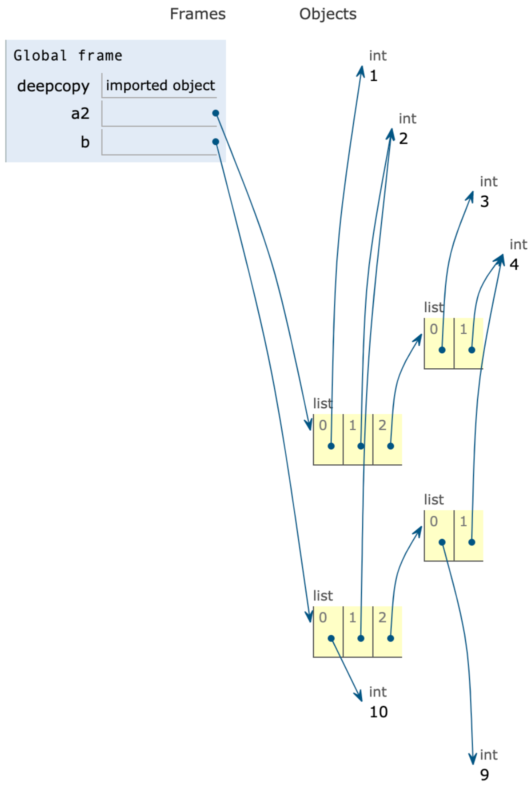
Shallow copy v.s. Deep copy

- **Deep copy**

```
from copy import deepcopy
a2 = [1, 2, [3, 4]]
b = deepcopy(a2)
b[0] = 10
b[2][0] = 9
```

b[2][0]的修改不再影响a[2][0]的值





http://pythontutor.com/

从其他对象构建列表

- **从range对象创建**

```
>>> list(range(11, 20))
[11, 12, 13, 14, 15, 16, 17, 18, 19]
```

- **从字符串创建**

```
>>> list("Hello World")
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

- **从迭代器创建**

```
>>> reversed([3, 9, 1, 7])
<list_reverseiterator object at 0x7fc961b154d0>
>>> list(reversed([3, 9, 1, 7]))
[7, 1, 9, 3]
```

range对象

- 用于对数字序列进行处理的一种内建对象
- 使用range函数创建，参数与切片类似
  - start end step

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1, 5))
[1, 2, 3, 4]
>>> list(range(1, 5, -1))
[]
>>> list(range(5, 1, -1))
[5, 4, 3, 2]
>>> list(range(5, -5, -1))
[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
```

列表综合应用

- **按分数段统计得分情况，并排序输出**

```
>>> scores = [97, 65, 84, 83, 68, 72, 77, 60, 73, 95]
>>> [s for s in scores if 60<=s<70]
[65, 68, 60]
>>> sorted([s for s in scores if 60<=s<70])
[60, 65, 68]
>>> [ sorted([s for s in scores if i<=s<i+10]) for i
in range(60,100,10)]
[[60, 65, 68], [72, 73, 77], [83, 84], [95, 97]]
```

字符串及其常用操作

字符串

- **用于表示字符序列的类型**
  - 单引号、双引号标识
  - 三引号可以跨行引用

- **列表是可变的！**
- **字符串是一种值不可以发生改变的对象！**

Mutable vs Immutable Objects 后续将详细讨论

```
>>> type("string2")
<class 'str'>
>>> type('string1')
<class 'str'>
>>> type('str"i"ng1')
<class 'str’>
>>> print("str'i'ng1")
str'i’ng1

>>> print('''this is a very long
... sentence, so i have to break
... into several lines''')
this is a very long
sentence, so i have to break
into several lines
```

字符串的访问

- 字符串可以看做是字符的列表，适用许多列表操作

```
>>> list("Hello World")
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

- 下标访问、切片，循环访问元素、列表解析等
- 列表的+、*、len、sorted、reversed等

```
>>> aStr = 'The Boeing Company'
>>> hStr = aStr[:4] + 'IBM' + aStr[-8:]
>>> hStr
'The IBM Company'
```

Python提供了常用字符串功能

- 这些功能会返回新的对象
  - upper(), lower()
  - split()
  - strip(), lstrip(), rstrip()

```
>>> "Guido van Rossum".upper()
'GUIDO VAN ROSSUM'

>>> "Guido van Rossum".split()
['Guido', 'van', 'Rossum']

>>> "  Guido ".lstrip()
'Guido '
```

- 原有字符串的值不能修改

```
>>> "hello"[0]
'h'
>>> "hello"[0]='H'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

字符串常用方法

capitalize()	center()	**count()**	**encode()**	endswith()	**find()**
format()	**index()**	isalnum()	isalpha()	isdigit()	**islower()**
isspace()	istitle()	isupper()	**join()**	ljust()	**lower()**
lstrip()	maketrans()	partition()	**replace()**	rfind()	rindex()
rjust()	rpartition()	rstrip()	**split()**	splitlines()	startswith()
strip()	swapcase()	title()	translate()	upper()	zfill()

- **查找计数**
  - count, find, rfind, index, rindex, startswith, endswith
- **切分**
  - split, partition, rpartition
- **替换**
  - replace, translate
- **判断调整大小写**
  - istitle, isupper, islower; title, swapcase, upper, lower
- **调整缩进、空格、对齐**
  - strip, lstrip, rstrip, ljust, rjust, zfill
- **……**

- **Join方法**

```
>>>help(str.join)
```

```
join(self, iterable, /)
    Concatenate any number of strings.

    The string whose method is called is inserted in
between each given string.
    The result is returned as a new string.

    Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
```

```
>>> langs = ("C++", "Python", "Java", "Scheme")
>>> " ".join(langs)
'C++ Python Java Scheme'
```

- **find方法**

```
find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

• index方法

```
index(...)
    S.index(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.
```

• format方法

```
>>> "{} beat {} yesterday!".format("Lakers","Pacers")
'Lakers beat Pacers yesterday!'
>>> "{1} beat {0} yesterday!".format("Lakers","Pacers")
'Pacers beat Lakers yesterday!'
```

```
>>> "Print float : {0:.0f}".format(3.1415)
'Print float : 3'
>>> "Print float : {0:.2f}".format(3.1415)
'Print float : 3.14'
```

字符串中的转义字符

- 表示缩进、换行、换页

```
>>> x = "Hello, \t my friend, \n \t\t Greetings!"
>>> print(x)
Hello,   my friend,
  Greetings!
```

- 直接使用ascii码

```
>>> print("\x64", "\x34")
d 4
```

简便的字符串处理功能

- **f-string**

```
>>> name = "Python"
>>> f"Hello, dear {name}"
'Hello, dear Python'
```

- **%**

```
>>> "This is %s" % ("Python")
'This is Python'
```

- **r-string: 阻止解释转义字符**

```
>>> print(r"Hello, \t my friend, \n \t\t Greetings!")
Hello, \t my friend, \n \t\t Greetings!
>>> print(r"\x64", r"\x34")
\x64 \x34
```

字符串应用示例:

- **取出文本中的引用内容:**

```
>>> s = 'What do you think of this saying "No pain, No gain"?'
>>> index = s.find("\"")
>>> index2 = s.rfind("\"")
>>> s[index+1:index2]
'No pain, No gain'
```

```
s.split("\"")[1]
```

元组

元组及其创建

- 一组具有序列关系的元素的组合

```
myTuple = "Mon.", "Tue.", "Wed.", "Thu.", "Fri."
myTuple = ("Mon.", "Tue.", "Wed.", "Thu.", "Fri.")
```

注意，没有括号标识也是元组（by default）。

- 元组的元素可以不同
  - 可以方便的利用元组将一组数据组合成一个对象(packing)

```
myTuple = "Mon.", [1, 2, 3]
```

  - 也可以再把元组赋值给多个变量（unpacking)

```
x, y = myTuple
```

- 元组对象是不可变对象（immutable)

元组的操作

- **因为元素具有序列关系，元组具有跟序列相关的操作**
  - 下标访问、循环访问元素、用于列表解析
  - len, 加、乘运算符
- **构建新的元组：**
  - 切片, sorted, reversed, 利用tuple()转换

```
>>> tuple([1, 2, 3, 4])
(1, 2, 3, 4)
```

- **因为元组对象是immutable，所以不能增加、删除、“修改”其元素**
  - 如：append、pop、insert等

```
>>> myTuple[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

元组的用途

- 显式标识对象不应该再被修改
- 多重赋值，函数参数、返回值处理等
  - 利用packing和unpacking
- packing示例：参数传递

将多个元素组合成一个元素，比如元组

```
>>> "{} beat {} yesterday!".format("Lakers","Pacers")
```

- unpacking示例：从控制台获取多个输入

```
>>> x, y = input().split()
23 16
>>> x, y = eval(x), eval(y)
```

将一个元素拆分成多个元素

```
>>> x, y = [ eval(t) for t in input().split() ]
```

packing和unpacking

- **元素数量一致的unpacking**
  - unpacking的对象可以是list、tuple、set等

```
>>> a, b = 1, 2
>>> a, b = b, a
>>> a, b
(2, 1)
```

```
>>> a, b, c = 1, 2, 3
>>> a, b = 1, 2, 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
>>> a, b, c = 1, 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
```

packing和unpacking

- **元素数量可变的unpacking情形（星号操作符*）**
  - *标识的变量将unpacking剩余的元素组合为列表

```
>>> a, *b = 1, 2, 3
>>> a
1
>>> b
[2, 3]
```

```
>>> *b, = 1, 2, 3
>>> b
[1, 2, 3]
```

```
>>> first, *middle, last = [1, 2, 3, 4, 5, 6]
>>> first, middle, last
(1, [2, 3, 4, 5], 6)
```

packing和unpacking

- **特殊的packing**
  - 将一个序列对象展开为其元素的序列（一组对象）

```
>>> x = (1, 2, 3)
>>> [0, x, 4]
[0, (1, 2, 3), 4]
>>> [0, *x, 4]
[0, 1, 2, 3, 4]
```

```
>>> list(zip(*matrix))
```

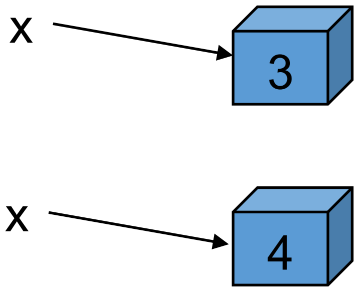
可以看做是unpacking之后再进行packing

MUTABLE V.S. IMMUTABLE

Mutable v.s. Immutable

- **Mutable 可变对象:**
  - list, dict, set, 大部分自定义类型的对象
- **Immutable 不可变对象:**
  - int, bool, float, tuple, str, frozenset

```
>>> mylist = [0,1,2,3]
>>> id(mylist)
140525391579408
>>> mylist[0] = 100
>>> id(mylist)
140525391579408
>>> mylist.append(36)
>>> id(mylist)
140525391579408
```



```
>>> x = 3
>>> id(x)
4314154192
>>> x = x + 1
>>> id(x)
4314154256
```

Python里其实无法改变一个int的值！！！

mutable/immutable是指对象状态（值）是否可变

int也是immutable！！！

Immutable中的Mutable

```
>>> myTuple = ("test", [0, 1])
>>> myTuple[0] = "Test"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
>>> myTuple[1] = [1, 2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

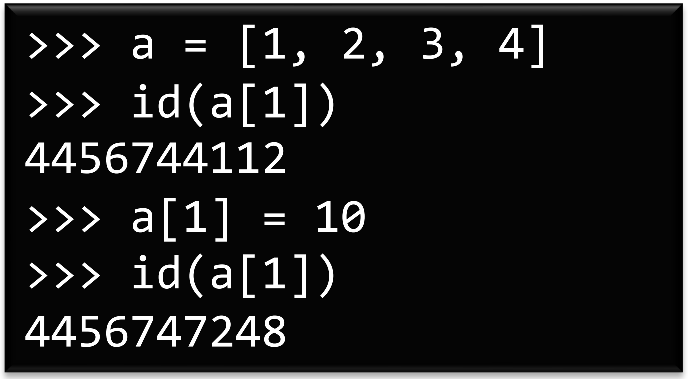
```
>>> myTuple[1][0] = [1, 2]
>>> myTuple
('test', [[1, 2], 1])
```

理解Immutable的复合数据类型

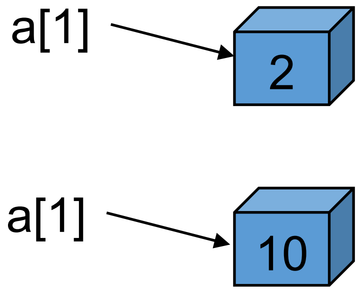
- 元组是一种元素的组合，其中：
- 不可变的是每一个元组元素到实际对象（值）的引用/绑定关系
- 如果某个元组元素本身是可变的，则其值仍然可以发生变化

```
>>> id(myTuple)
140525391406336
>>> id(myTuple[0])
140525393085808
>>> id(myTuple[1])
140525390835296
```

Mutable中的Immutable



```
>>> a = [1, 2, 3, 4]
>>> id(a[1])
4456744112
>>> a[1] = 10
>>> id(a[1])
4456747248
```



- 在发生赋值时：
  - 2这个int对象的值并没有发生变化（也不能变）
  - a[1]被关联到了一个新的int对象，其值为10
- **list对象是mutable，是指列表中每个元素到实际值/对象的引用关系是可变的；列表的元素仍然可以是immutable**

对运行效率的可能影响(例如：构造数字字符串"0 1 2 3 4")

```
>>> string_builder = ""
>>> for i in range(5):
...     string_builder += (str(i) + " ")
...
>>> string_builder.strip()
```

反复修改一个immutable，需要不断创建新的对象。

```
>>> string_list = []
>>> for i in range(5):
...     string_list.append(str(i))
...
>>> " ".join(string_list)
```

构建一个mutable的list，再利用内建方法join进行组合（交给解释器去优化）

```
" ".join([str(i) for i in range(5)])
```

```
" ".join(map(str, range(5)))
```

*map用于将操作应用于一系列对象（函数式编程）

对运行效率的可能影响

- **元组（mutable） v.s. 列表（immutable)**
  - 更高的效率、更少的空间

```
>>> import timeit
>>> timeit.timeit("[1,2,3,4,5]")
0.0457108104222685
>>> timeit.timeit("(1,2,3,4,5)")
0.007636267924681306
```

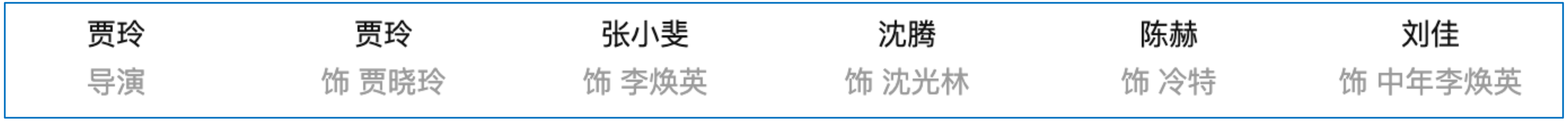
```
>>> from sys import getsizeof
>>> getsizeof([1,2,3,4,5])
112
>>> getsizeof((1,2,3,4,5))
96
```

  - 更重要的是immutable可以作为参数传递而不用担心被修改，在多线程、多对象共享环境下都有更好的安全性
  - *元组可以作为字典的键值或集合元素，列表不行
- **仅有三种mutable的容器可以用解析方法创建（list, set, dict)**

字典

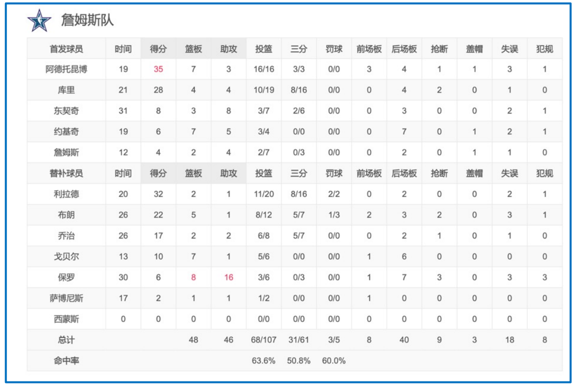
字典应用场景

- 用于保存成对信息的一种数据结构



贾玲	贾玲	张小斐	沈腾	陈赫	刘佳
导演	饰 贾晓玲	饰 李焕英	饰 沈光林	饰 冷特	饰 中年李焕英

Key	Value
'Mayue'	3000
'Lilin'	4500
'Wuyun'	8000



詹姆斯队

首发球员	时间	得分	篮板	助攻	投篮	三分	罚球	前场板	后场板	抢断	盖帽	失误	犯规
阿德托昆博	19	35	7	3	16/16	3/3	0/0	3	4	1	1	3	1
库里	21	28	4	4	10/19	8/16	0/0	0	4	2	0	1	0
东契奇	31	8	3	8	3/7	2/6	0/0	0	3	0	0	2	1
约基奇	19	6	7	5	3/4	0/0	0/0	0	7	0	1	2	1
詹姆斯	12	4	2	4	2/7	0/3	0/0	0	2	0	1	1	0
替补球员	时间	得分	篮板	助攻	投篮	三分	罚球	前场板	后场板	抢断	盖帽	失误	犯规
利拉德	20	32	2	1	11/20	8/16	2/2	0	2	0	0	2	1
布朗	26	22	5	1	8/12	5/7	1/3	2	3	2	0	3	1
乔治	26	17	2	2	6/8	5/7	0/0	0	2	1	0	1	0
戈贝尔	13	10	7	1	5/6	0/0	0/0	1	6	0	0	0	0
保罗	30	6	8	16	3/6	0/3	0/0	1	7	3	0	3	3
萨博尼斯	17	2	1	1	1/2	0/0	0/0	1	0	0	0	0	0
西蒙斯	0	0	0	0	0/0	0/0	0/0	0	0	0	0	0	0
总计			48	46	68/107	31/61	3/5	8	40	9	3	18	8
命中率					63.6%	50.8%	60.0%						

- 实际需要保存的是Key 到 Value的映射关系

- 可以利用多个有序结构存储成对信息：

```
>>> names = ['Mayue', 'Lilin', 'Wuyun']
>>> salaries = [3000, 4500, 8000]
>>> salaries[names.index('Lilin')]
4500
```

Key	Value
'Mayue'	3000
'Lilin'	4500
'Wuyun'	8000

- 或者利用元组的packing功能

```
>>> salaries = [('Mayue', 3000), ('Lilin', 4500),
('Wuyun', 8000)]
>>> [s[1] for s in salaries if s[0]=='Lilin']
```

- 可是多个列表信息不容易维护，元组不能修改且查询不便

字典

- **用于存储和查询成对元素的数据结构**

```
>>> salariesDict = {'Mayue': 3000, 'Lilin': 4500,
'Wuyun': 8000}
>>> salariesDict['Lilin']
4500
```

- **元素修改（字典对象是可变对象mutable）**

```
>>> salariesDict['Lilin'] = 5000
>>> print(salariesDict)
{'Mayue': 3000, 'Lilin': 5000, 'Wuyun': 8000}
```

字典创建

- **从key value列表创建**

```
>>> salaries = [3000, 4500, 8000]
>>> names = ['Mayue', 'Lilin', 'Wuyun']
>>> sd1 = dict(zip(names, salaries))
>>> print(sd1)
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

- **从tuple列表创建**

```
>>> salariesList = [('Mayue', 3000), ('Lilin', 4500),
('Wuyun', 8000)]
>>> sd2 = dict(salaries)
>>> print(sd2)
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

- **利用字典解析创建新字典（dictionary comprehension)**

```
newdict = {expr : expr for item in iterable if condition == True}
```

```
>>> dict1 = { chr(ord('a') + i) : i + 1 for i in range(5)}
>>> dict1
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

  - chr()和ord()用于转换字符及其编码（ascii)

```
>>> double_dict1 = { k:v*2 for (k,v) in dict1.items()}
>>> double_dict1
{'a': 2, 'b': 4, 'c': 6, 'd': 8, 'e': 10}
```

字典中获取元素信息

- 可转化为元素序列（iterable），用于循环遍历等用途

```
>>> sd1 = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> sd1.keys()
dict_keys(['Mayue', 'Lilin', 'Wuyun'])
>>> sd1.values()
dict_values([3000, 4500, 8000])
>>> sd1.items()
dict_items([('Mayue', 3000), ('Lilin', 4500), ('Wuyun',
8000)])
```

因为用于索引value，所以keys中的值不能重复。

集合

集合的应用场景

- **集合用于描述一系列无序不重复的元素的组合**
  - 可变集合（set)
  - 不可变集合（frozenset)

```
>>> names = ['Mayue', 'Lilin', 'Wanqi', 'Mayue', 'Lilin']
>>> nameset = set(names)
>>> print(nameset)
{'Mayue', 'Wanqi', 'Lilin'}
```

集合

- **创建集合**

```
>>> nameset = {'Mayue', 'Lilin', 'Wanqi', 'Mayue', 'Lilin'}
>>> print(nameset)
{'Mayue', 'Wanqi', 'Lilin'}
```

- **集合基本操作**
  - 添加删除元素（set）
  - 元素判断 in / not in
  - 合交并补差 等运算

issubset(t)
issuperset(t)
union(t)
intersection(t)
difference(t)
symmetric_difference(t)
copy()

• 用集合解析创建集合（set comprehension)

```
newlist = {expression for item in iterable if condition == True}
```

```
>>> sentence = "The cat in the hat had two sidekicks, thing one and thing two."
>>> words = sentence.lower().replace('.', '').replace(',', '').split()
>>> words
['the', 'cat', 'in', 'the', 'hat', 'had', 'two', 'sidekicks', 'thing', 'one',
'and', 'thing', 'two']
>>> unique_short_words = {word for word in words if len(word) <= 3}
>>> unique_short_words
{'had', 'two', 'one', 'in', 'hat', 'cat', 'and', 'the'}
```

回顾

- **Python中的列表、字符串、元组等数据结构**
- **不同数据结构的创建、使用、转换**

- **命令式 v.s. 声明式**
- **Mutable v.s. Immutable**

- **一些可能有用的函数**
  - timeit, getsizeof
  - chr, ord
  - join, format, f-string

本周任务

- 熟悉列表、字符串等各种数据结构的使用
- 完成OJ在线编程作业

- 阅读：
- https://stackoverflow.com/questions/626759/whats-the-difference-between-lists-and-tuples
- 更多练习：
- http://coolpython.net/python_primary/data_type/list_exercises.html