

Lesson 9 - Lists

(more than 1 value)

List → a kind of Collection

e.g.: `>>> print([1, [5,6], 7])`

{ Algorithms → set of rules/steps
Data structures → way of organising data
e.g.: `friends = ["Joseph", "Glenn", "Sally"]`

e.g.: `>>> for i in [5,4,3,2,1]:
 print(i)`

- ① Looking inside lists
→ index (from 0)

Joseph	G Glenn	Sally
0	1	2

- ② Lists are mutable
(可變的)

Lists are Mutable

- Strings are "immutable" - we cannot change the contents of a string - we must make a new string to make any change
- Lists are "mutable" - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'  
>>> fruit[0] = 'b'  
Traceback (most recent call last):  
  File "", line 1  
    fruit[0] = 'b'  
TypeError: 'str' object does not support item assignment  
>>> x = fruit.lower()  
>>> print(x)  
banana  
>>> lotto = [2, 14, 26, 41, 63]  
>>> print(lotto)  
[2, 14, 26, 41, 63]  
>>> lotto[2] = 28  
>>> print(lotto)  
[2, 14, 28, 41, 63]
```

- ③ How long is a list? → `len()`

- ④ Range function → returns a list of numbers from 0 to one less than the parameter

range()
↑
a number

Using the range function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
>>> print(range(4))  
[0, 1, 2, 3]  
>>> friends = ['Joseph', 'Glenn', 'Sally']  
>>> print(len(friends))  
3  
>>> print(range(len(friends)))  
[0, 1, 2]  
>>>
```

A tale of two loops...

```
friends = ['Joseph', 'Glenn', 'Sally']  
  
for friend in friends : ↗  
    print('Happy New Year:', friend)  
  
for i in range(len(friends)) : ↗  
    friend = friends[i]  
    print('Happy New Year:', friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']  
>>> print(len(friends))  
3  
>>> print(range(len(friends)))  
[0, 1, 2]  
>>>  
  
Happy New Year: Joseph  
Happy New Year: Glenn  
Happy New Year: Sally
```

⑤ Concatenating lists using + (串聯)

e.g.:
`>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]`

⑥ Slicing lists

Lists can be sliced using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is "up to but not including"

"up to but not including"

⑦ List methods use `dir()`

List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
 'pop', 'remove', 'reverse', 'sort']
>>>
```

⑧ Building a list from scratch → `X = list()` ← Build an empty list `X.append()` ← Fill in

- We can create an empty list and then add elements using the `append` method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

R.X. `X.append()` can be written directly as a line to run

⑨ Is something in a list? "in" & "not in"

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return `True` or `False`
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>> .
```

⑩ Lists are in Order → can use `sort()`

- A list can hold many items and keeps those items in the order until we do something to change the order
- A list can be sorted (i.e., change its order)
- The `sort` method (unlike in strings) means "sort yourself"

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> friends.sort()
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[0])
Joseph
>>>
```

← `X.sort()` can be written directly as a line to run

⑪ Built-in functions and Lists → `len()`; `max()`; `min()`; `sum()`; `sum() / len()`

e.g. (compared with previous)

previous →
method
(less memory but
no storage)

```

total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print('Average:', average)

numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)

```

↑ use "list" (more memory)

⑫ Strings & Lists

`X.split()`

These "white spaces" can be omitted through `h.split()`

Best Friends: Strings and Lists

```

>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With

```

`Split` breaks a string into parts and produces a list of strings. We think of these as words. We can `access` a particular word or `loop` through all the words.

- When you do not specify a delimiter, multiple spaces are treated like one delimiter
- You can specify what delimiter character to use in the `splitting`

Note: `X.split()` only recognises
"white spaces"

`X.split(a)` recognises `a` as
splitting point

```

>>> line = 'A lot           of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
3
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>

```

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```

fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])

```

Sat
Fri
Fri
Fri
...

```

>>> line = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>

```

↑ handle .txt with `split()`

The Double Split Pattern

```

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

words = line.split()                      stephen.marquard@uct.ac.za
email = words[1]                           ('stephen.marquard', 'uct.ac.za')
pieces = email.split('@')                 '@uct.ac.za'
print(pieces[1])

```

↑ use `double split()` to find @XXX

⑬ Guardian pattern → Lists & Files

13.1 Tricky point: Blank lines in a file may cause traceback, when try to recognise an element in the list after splitting

Method to guard: 13.1.1 if $\text{len}(\text{wds}) < 1$:
continue wds is a splitted list

13.1.2 if line == "":
continue a line in the file

13.2 Extended trick: if the line(s) is $< a$, then $\text{wds}[b]$ will cause error
then we need guardian pattern
↳ if $\text{len}(\text{wds}) < a$:
continue

⑭ " $!=$ " → means "Not equal to"

e.g.: if $\text{wds}[0] \neq \text{"From"}$:
continue

Exercise 8.4

8.4 Open the file `romeo.txt` and read it line by line. For each line, split the line into a list of words using the `split()` method. The program should build a list of words. For each word on each line check to see if the word is already in the list and if not append it to the list. When the program completes, sort and print the resulting words in alphabetical order.

You can download the sample data at
<http://www.py4e.com/code3/romeo.txt>

Check Code Reset Code Grade updated on server.

```
1 fname = input("Enter file name: ")
2 fh = open(fname)
3 lst = []
4 for line in fh:
5     line = line.rstrip()
6     words = line.split()
7     for word in words:
8         if word not in lst:
9             lst.append(word)
10
11 lst.sort()
12 print(lst)
13
```

Your Output
['Arise', 'But', 'It', 'Juliet', 'Who', 'already', 'and', 'breaks', 'east', 'envious', 'fair', 'grief', 'is', 'kill', 'light', 'moon', 'p ale', 'sick', 'soft', 'sun', 'the', 'through', 'what', 'window', 'wit h', 'yonder']

Desired Output
['Arise', 'But', 'It', 'Juliet', 'Who', 'already', 'and', 'breaks', 'east', 'envious', 'fair', 'grief', 'is', 'kill', 'light', 'moon', 'p ale', 'sick', 'soft', 'sun', 'the', 'through', 'what', 'window', 'wit h', 'yonder']

Lesson 10 - Dictionaries

(very powerful tool)

little in-memory database

a second kind of Collections (more than 1 value in)

- List → a linear collection of values that stay in order
- Dictionaries → a "bag" of values, each with its own label (key)

Dictionaries

- Lists index their entries based on the position in the list
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a "lookup tag"

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

← dict()

(dictionary is bracketed with { })

② Lists vs. Dictionaries

Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

List		
Key	Value	lst
[0]	21	
[1]	183	

Dictionary		
Key	Value	ddd
['course']	182	
['age']	21	

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of key : value pairs
- You can make an empty dictionary using empty curly braces

```
>>> jjj = { 'chuck' : 1 , 'fred' : 40, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 40}
>>> ooo = {}
>>> print(ooo)
{}>>>
```

↑ "`= { }`" is equal to "`= dict()`"

③ Error to refer a non-existing key

Dictionary Tracebacks

- It is an error to reference a key which is not in the dictionary
- We can use the in operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

④ New keys & Dictionary

When we see a new name

When we encounter a new name, we need to add a new entry in the **dictionary** and if this the second or later time we have seen the **name**, we simply add one to the count in the **dictionary** under that **name**

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print(counts)
```

{'csev': 1, 'zqian': 1, 'cwen': 2, 'mengwud': 1}

⑤ Check to see key & add new

`x.get(a, b)`

variable of
the label default value
for new key

The get method for dictionaries

The pattern of checking to see if a **key** is already in a dictionary and assuming a default value if the **key** is not there is so common that there is a **method** called **get()** that does this for us

```
if name in counts:
    x = counts[name]
else:
    x = 0
```

Default value if key does not exist
(and no Traceback).

`x = counts.get(name, 0)`

{'csev': 2, 'zqian': 1, 'cwen': 2}

⑥ Idiom -

Simplified counting with `get()`

`Counts[name] = Counts.get(name, 0) + 1`

Simplified counting with `get()`

We can use `get()` and provide a **default value of zero** when the **key** is not yet in the **dictionary** - and then just add one

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

Default {'csev': 2, 'zqian': 1, 'cwen': 2}

⑦ Counting in a file through dictionary

```
python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran
into the tent and the tent fell down on the
clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']
Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3,
'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell':
1, 'the': 7, 'tent': 2}
```

⑧ Definite Loops and Dictionaries

```
e.g.: >>> counts = {'chunk': 1, 'fred': 42, 'jan': 100}
>>> for key in counts:
           print(key, counts[key])
jan 100
chunk 1
fred 42
>>>
```

(9) Retrieving lists of Keys and Values → `list(D)`; `D.keys()`; `D.values()`

Retrieving lists of Keys and Values

You can get a list of **keys**, **values**, or **items** (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a "tuple"? - coming soon...

`D.items()`

(10) Two iteration variables & `.items()` (Very elegant only seen in Python)

Bonus: Two Iteration Variables!

- We loop through the **key-value** pairs in a dictionary using ***two*** iteration variables
- Each iteration, the first variable is the **key** and the second variable is the corresponding **value** for the key

Two iterate simultaneously

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
        aaa   bbb
jan 100      [jan] 100
chuck 1      [chuck] 1
fred 42      [fred] 42
```

(11) Example : using two nested loops

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

`python words.py`
Enter file: words.txt
to 16

`python words.py`
Enter file: clown.txt
the 7

Using two nested loops

Lesson 11 - Tuples (元组)

(XXX, XXX, ...)

- ① Tuples are very similar with Lists, but is bracketed by "()" instead of "[]"
(XXX, XXX, XXX, ...)

e.g.:
 >>> y = (1, 9, 2)
 >>> print(max(y))
 9

- ② Difference with Lists - • Immutable

For efficiency purpose,
Less storage, quicker to access

but... Tuples are "immutable"

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]      >>> y = 'ABC'      >>> z = (5, 4, 3)
>>> x[2] = 6          >>> y[2] = 'D'        >>> z[2] = 0
>>> print(x)          Traceback: str'     Traceback: tuple'
>>> [9, 8, 6]          object does       object does
                           not support item   not support item
                           Assignment           Assignment
                           >>>
```

- Tuples → cannot use X.sort(); X.append(); X.reverse()

•
 >>> dir(List())
 ['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

 >>> dir(tuple())
 ['count', 'index']

- ③ Why Tuples? — More efficient, used when no need modification & saving memory & better performance (use & throw away)

- ④ Assignments in Tuples can be easier (put a tuple on the left-hand side)

e.g.:
 >>> (x, y) = (4, 'Fred')
 >>> print(y)
 fred
 >>> (a, b) = (99, 98)
 >>> print(a)
 99

- ⑤ Tuples & Dictionaries

Tuples and Dictionaries

The items() method in dictionaries returns a list of (key, value) tuples

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k, v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

* This returns same results from following statements:
for k, v in d.items():

⑥ Comparable (True/False) only compare the first different elements (or first letters) found

Tuples are Comparable

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

Not scan further for comparison

⑦ Sorting Lists of Tuples → sorted()

Sorting Lists of Tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary
- First we sort the dictionary by the key using the items() method and sorted() function

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
dict_items([(‘a’, 10), (‘c’, 22), (‘b’, 1)])
>>> sorted(d.items())
[(‘a’, 10), (‘b’, 1), (‘c’, 22)]
```

(elegant in Python)

Using sorted()

We can do this even more directly using the built-in function `sorted` that takes a sequence as a parameter and returns a sorted sequence

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[(‘a’, 10), (‘b’, 1), (‘c’, 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

⑧ Sort by values instead of key (flip things around)

Sort by values instead of key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items():
...     tmp.append((v, k)) ← Tuples in a List
...
>>> print(tmp)
[(10, ‘a’), (22, ‘c’), (1, ‘b’)]
>>> tmp = sorted(tmp, reverse=True) ← reverse=True gives reversed order results
>>> print(tmp)
[(22, ‘c’), (10, ‘a’), (1, ‘b’)]
```

Dictionary double loops simultaneously

Tuples in a List

← reverse=True gives reversed order results

⑨ Idiom → e.g. Find the top 10 most common words in a procedural way vs. in a closed form (classic) (Lambdas) (elegant in Python)

```
fhand = open(‘romeo.txt’)
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = list()
for key, val in counts.items():
    newtuple = (val, key) >>> Tuples as elements in a list
    lst.append(newtuple)

lst = sorted(lst, reverse=True)

for val, key in lst[:10]:
    print(key, val) ←
```

The top 10 most common words

Even Shorter Version

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print(sorted([(v,k) for k,v in c.items()]))
```

[(1, ‘b’), (10, ‘a’), (22, ‘c’)]

List comprehension creates a dynamic list. In this case, we make a list of reversed tuples and then sort it.

<http://wiki.python.org/moin/HowTo/Sorting>

↑ do everything in one line

* Summary:

3 different foundational data structures/collections

{ Lesson 9 Lists
Lesson 10 Dictionaries
Lesson 11 Tuples

They are related & we combine those 3 in different ways

tend not to use List, if we can take away with Tuples

+ 1 foundational format for each line read → Lesson 7 - string

e.g. (using Lessons 7-11 knowledge)

The screenshot shows a code editor interface with two panes: 'Your Output' and 'Desired Output'.
Your Output:
A list of hour counts:
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
Desired Output:
The same list of hour counts:
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
Code:
A Python script to read 'mbox-short.txt' and count messages by hour.

```
1 name = input("Enter file:")
2 # to name 'mbox-short' by default(when no input)
3 if len(name) < 1 : name = "mbox-short.txt"
4
5 handle = open(name)
6
7 time = dict()
8 for line in handle:
9     if line.startswith('From '):
10         words = line.split()
11         word & hour => string
12
13         for line in handle:
14             line = line.rstrip()
15             if line.startswith('From '):
16                 words = line.split()
17                 for word in words:
18                     if ':' in word:
19                         hour = word[:2]
20                         # print(hour)
21                         time[hour] = time.get(hour, 0) + 1
22
23         print(time)
24
25         for k,v in sorted(time.items()):
26             print(k,v)
```