Lesson 9 – Lists

(more than 1 value)

List → a kind of Collection

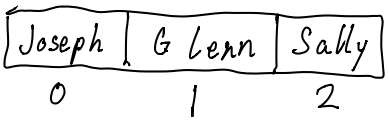
e.g.: >>> print([1, [5,6], 7])

{ Algorithms → set of rules/steps
Data structures → way of organising data

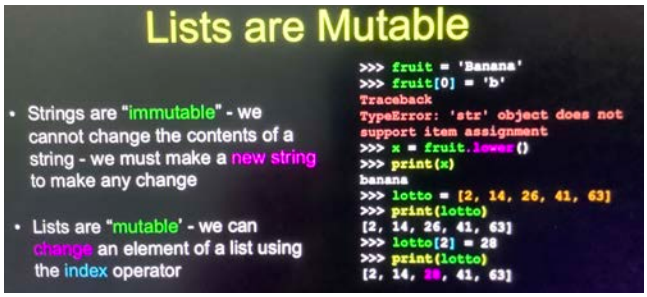
e.g.: friends = ["Joseph", "Glenn", "Sally"]

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

① Looking inside lists – index (from 0)



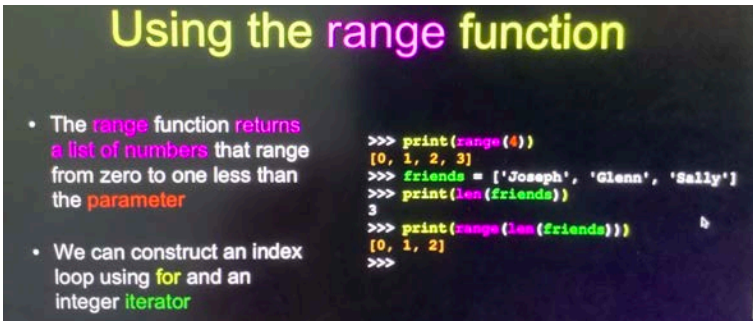
② Lists are mutable (可变的)

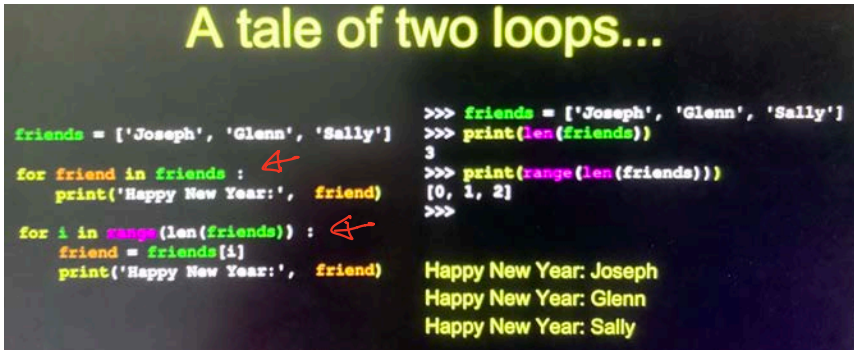


③ How long is a list? → len()

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

range( )
↑ a number

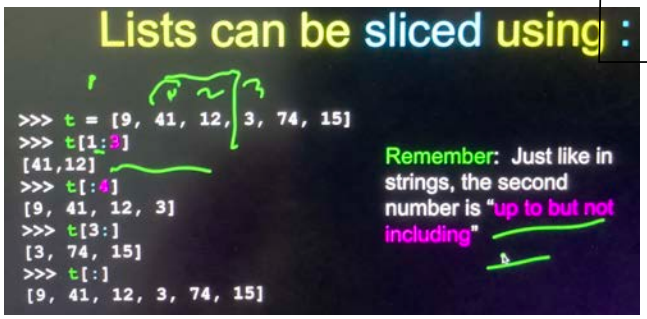




⑤ Concatenating lists using + (串联)

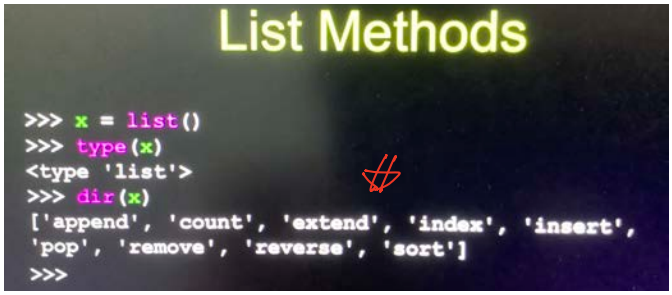
e.g.:
```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a+b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

⑥ Slicing lists



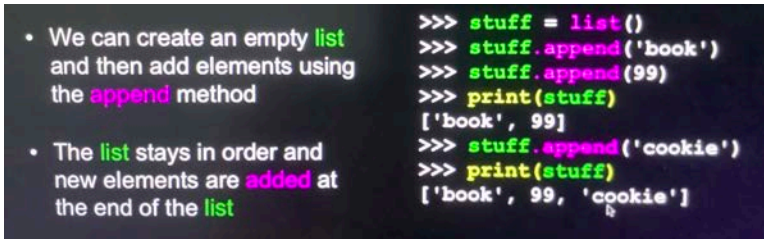
"up to but not including"

⑦ List methods use dir()

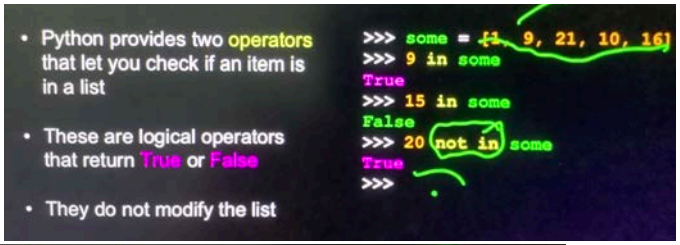


⑧ Building a list from Scratch → { X = list() ← Build an empty list ; X.append() ← Fill in

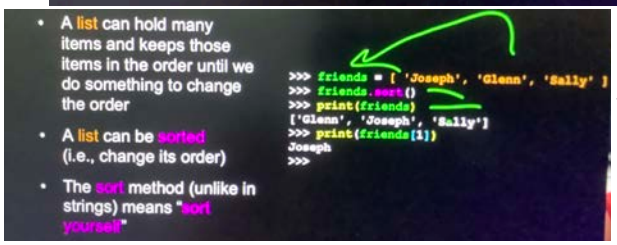
※ X.append() can be written directly as a line to run



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



⑩ Lists are in Order → can use sort()



← 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)
```

```
Enter a number: 3
Enter a number: 9
Enter a number: 5
Enter a number: done
Average: 5.66666666667
```

```
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()

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
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

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.

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

```
>>> 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))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

- 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 spliting point

```
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()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

↑ use double split() to find @XXX

⑬ Guardian pattern → Lists & Files

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

Method to guard: 13.1.1
```
if len(wds) < 1 :
    continue
```
wds is a splited list

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

13.2 Extended trick: if the line(s) is < a, then wds[b] will cause error

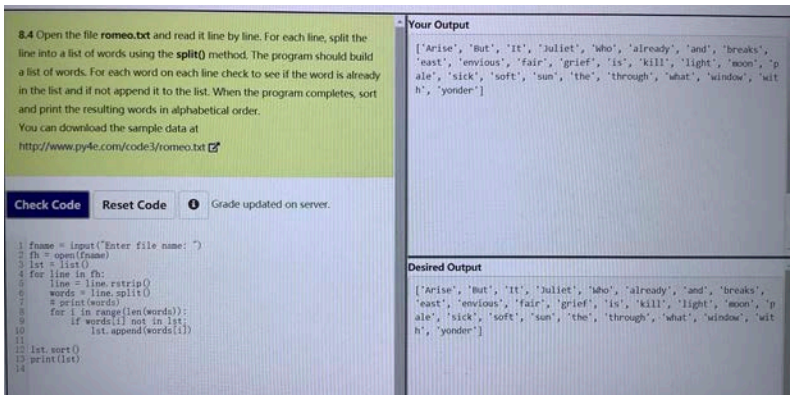
wds[b]: wds is a splited list; b ≥ a

then we need guardian pattern
```
↳ if len(wds) < a :
      continue
```

⑭ "!=" → means "Not equal to"

e.g.:
```
if wds[0] != "From" :
    continue
```

Exercise 8.4



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)

①

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 { })

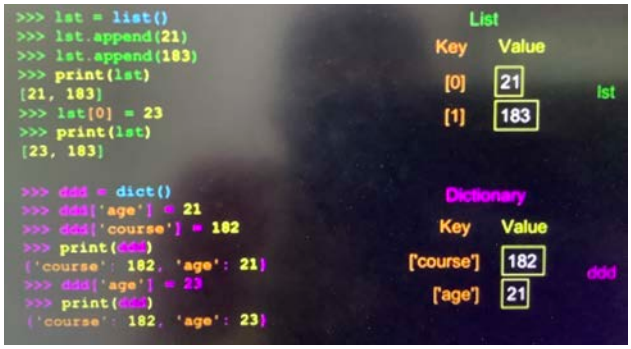
② Lists vs. Dictionaries

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}
```



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' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{}
>>>
```

↑ "= { }" is equal to "= dict()"

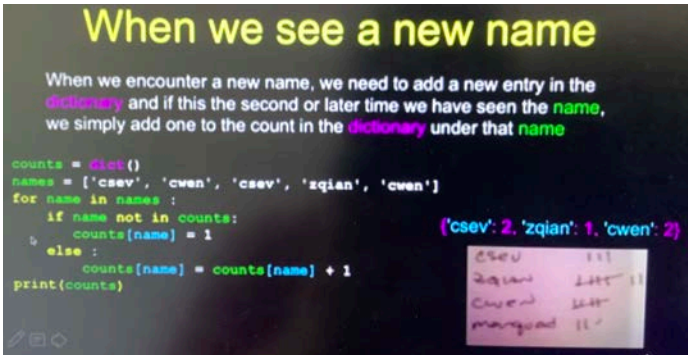
③ Error to refer a non-existing key

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
```

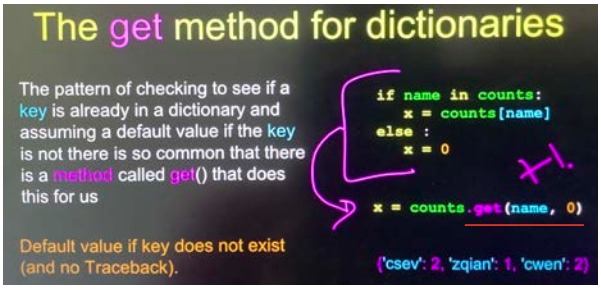
④ New keys & Dictionary



⑤ Check to see key & add new

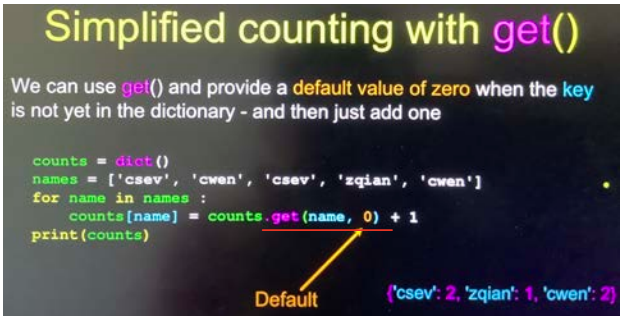
x.get(a, b)

variable of the label — default value for new key

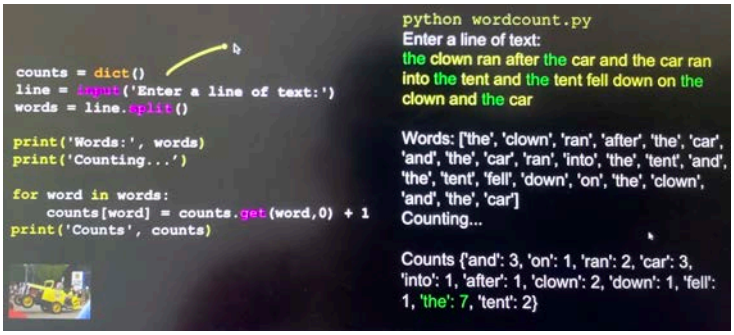


⑥ Idiom – Simplified counting with get()

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



⑦ Counting in a file through dictionary



⑧ Definite Loops and Dictionaries

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

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

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...

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

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

```
jan 100
chuck 1
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

Two iterate simultaneously

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

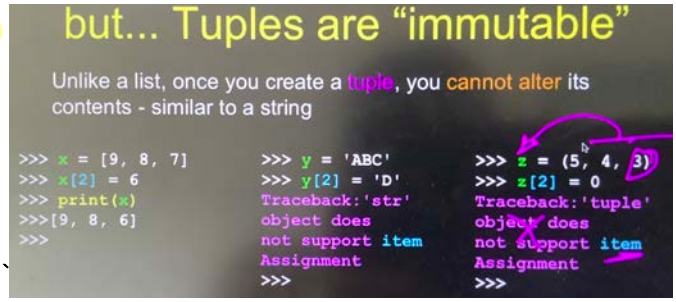
Lesson 11 - Tuples (元组) (XXX, XXX, ···)

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



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

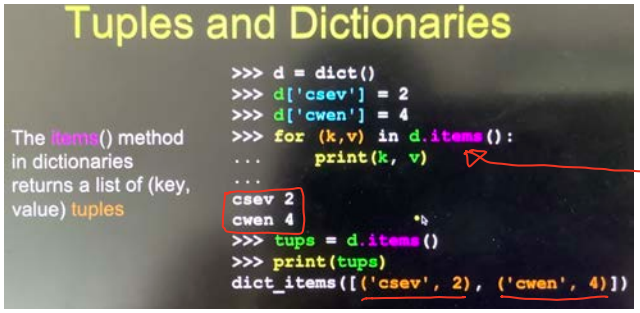
• >>> dir (list())
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> dir (tuple())
['count', 'index']

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

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



* 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

Not scan further for comparation

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
```

---

⑦ Sorting Lists of Tuples → sorted() (elegant in Python)

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)]
```

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) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> 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 (Tuples)

~~in procedural way vs. in a closed form~~

(classic) (Lambdas) (elegant in Python)

The top 10 most common words

```
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():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)

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

← Tuples as elements in a list

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:

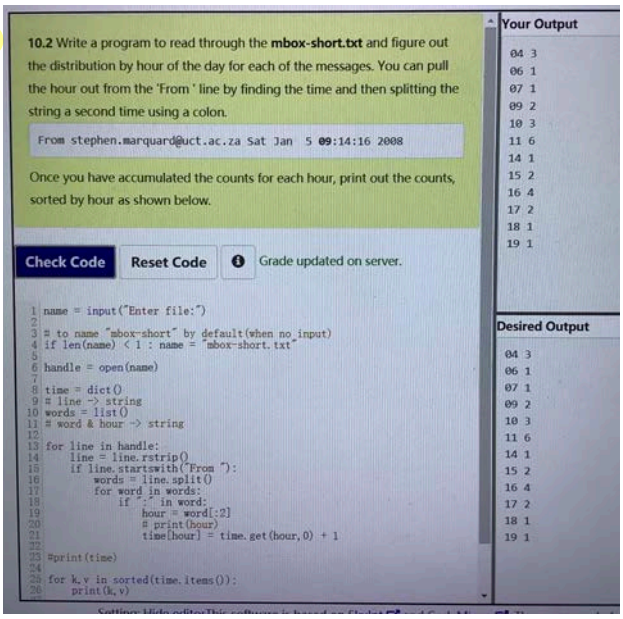
3 different foundational data structures/collections
- Lesson 9 Lists
- Lesson 10 Dictionaries
- Lesson 11 Tuples

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)



10.2 Write a program to read through the **mbox-short.txt** and figure out the distribution by hour of the day for each of the messages. You can pull the hour out from the 'From ' line by finding the time and then splitting the string a second time using a colon.

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

Once you have accumulated the counts for each hour, print out the counts, sorted by hour as shown below.

Check Code | Reset Code | Grade updated on server.

```
name = input("Enter file:")

# to name "mbox-short" by default(when no input)
if len(name) < 1 : name = "mbox-short.txt"

handle = open(name)

time = dict()
# line -> string
words = list()
# word & hour -> string

for line in handle:
    line = line.rstrip()
    if line.startswith("From "):
        words = line.split()
        for word in words:
            if ":" in word:
                hour = word[:2]
                # print(hour)
                time[hour] = time.get(hour,0) + 1

#print(time)

for k,v in sorted(time.items()):
    print(k,v)
```

Your Output

```
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

```
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```