Lists and Tuples

Section 4

Chapter 2

Quiz 4

Lists

The **`list`** Object

- A list is an ordered sequence of Python objects
  - A list is constructed by writing items enclosed in square brackets. Items in a list are separated by commas.

```
lst1=[26, 34, 3893]
lst2=['a', "bc", "BG@#HJG$^%"]
```

  - Objects in one list can be of different types.

```
school = ["CityU", 1984]
```

List indexing

- The indexing of lists is similar to that of strings.

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]
>>>
>>> lst[0]
'CityU'
>>> lst[1]
1984
>>> lst[2]
83
>>> lst[3]
'Tat Chee Ave'
```

```
>>> lst[-1]
'Tat Chee Ave'
>>> lst[-2]
83
>>> lst[-3]
1984
>>> lst[-4]
'CityU'
```

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]
>>>
>>> lst[0][1]  #This returns the second character of the first item ("CityU") in the list.
'i'
```

List indexing

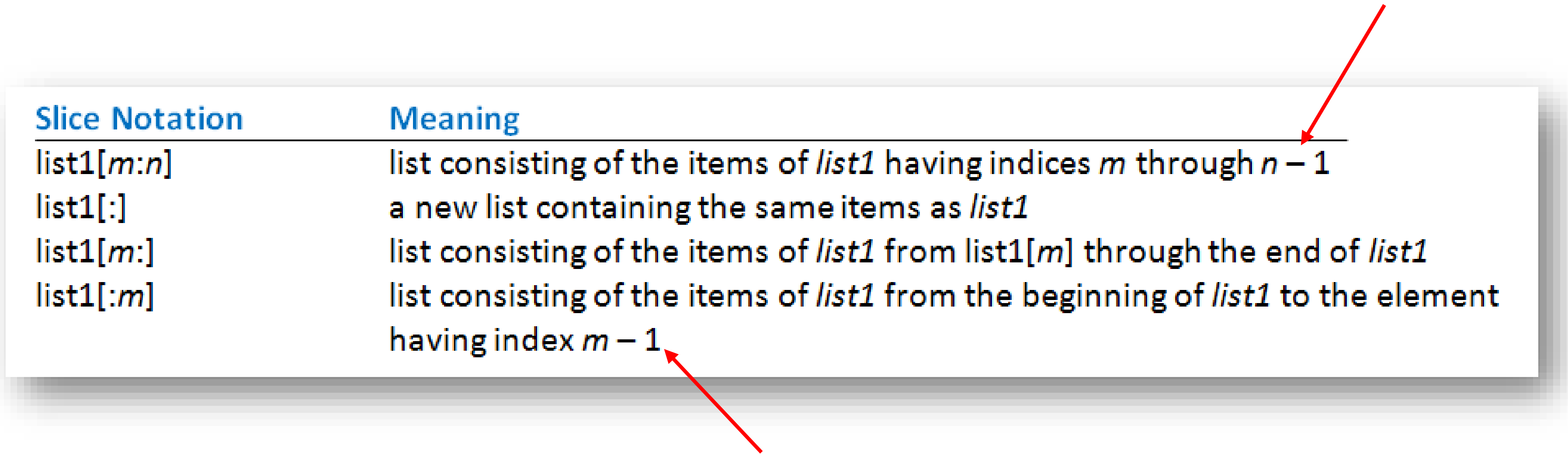
- The .find() method only works with strings.

- Lists have a similar method: .index()

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]
>>>
>>> lst.find(83)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    lst.find(83)
AttributeError: 'list' object has no attribute 'find'
>>>
>>> lst.index(83)
2
>>> lst.index("CityU")
0
```

List slicing

- A *slice* of a list is a sublist specified with colon notation
  - Analogous to a slice of a string



Slice Notation	Meaning
list1[*m*:*n*]	list consisting of the items of *list1* having indices *m* through *n* – 1
list1[:]	a new list containing the same items as *list1*
list1[*m*:]	list consisting of the items of *list1* from list1[*m*] through the end of *list1*
list1[:*m*]	list consisting of the items of *list1* from the beginning of *list1* to the element having index *m* – 1

List slicing

```
>>> lst = ['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> lst[1:3]
['b', 'c']
>>> lst[-5:-3]
['b', 'c']

>>> lst[1:3][0] #this returns the first item of the sublist.
'b'

>>> lst[:4]
['a', 'b', 'c', 'd']
>>> lst[4:]
['e', 'f']
>>> lst[:]
['a', 'b', 'c', 'd', 'e', 'f']

>>> lst[3:2] #when the left index > right index, the slicing returns the empty list.
[]
>>> lst[-2: -3]
[]
```

List concatenations

```
>>> ["CityU", 1984, 83, "Tat Chee Ave"] + [32, 5, 8, 9]
['CityU', 1984, 83, 'Tat Chee Ave', 32, 5, 8, 9]
>>>
>>> ['a', 34] * 3
['a', 34, 'a', 34, 'a', 34]
```

List operations

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]


>>> len(lst)    #len() returns the number of items in the list.
4

>>> lst.count(83) #.count() returns the number of occurrences of an item.
1


>>> lst.reverse()  #.reverse() reserves the order of the items.
>>> print(lst)
['Tat Chee Ave', 83, 1984, 'CityU']
```

List operations: adding items

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]
```

```
>>> lst.append("Kowloon Tong")  #.append() inserts an object at the end of the list.
>>> print(lst)
['CityU', 1984, 83, 'Tat Chee Ave', 'Kowloon Tong']
```

```
>>> lst.extend(["Kowloon", "Hong Kong"])  #.extend() inserts new list's items at the end
>>> print(lst)
['CityU', 1984, 83, 'Tat Chee Ave', 'Kowloon Tong', 'Kowloon', 'Hong Kong']
```

```
>>> lst.insert(2, "Tai Man") #.insert() inserts new item before the item of the stated index.
>>> print(lst)
    ['CityU', 1984, 'Tai Man', 83, 'Tat Chee Ave', 'Kowloon Tong', 'Kowloon', 'Hong Kong']
```

List operations: removing items

```
>>> lst=["CityU", 1984, 83, "Tat Chee Ave"]


>>> del lst[1]  #del removes the item with the stated index.
>>> print(lst)
['CityU', 83, 'Tat Chee Ave']


>>> lst.remove(83)  #.romove() removes the first occurence of an item
>>> print(lst)
['CityU', 'Tat Chee Ave']


>>> lst.clear()  #.clear() clears the list.
>>> print(lst)
[]
```

List operations: max, min, sum, sorted, .sort

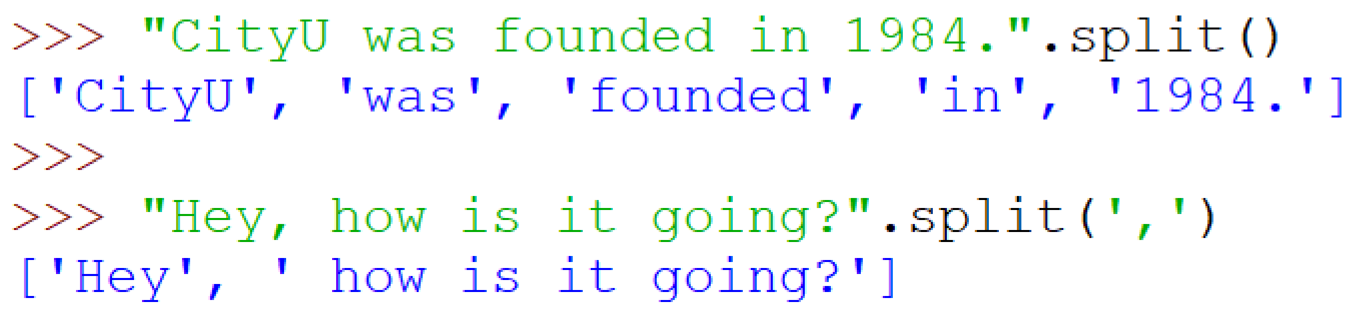
```
>>> lstNum=[32, 5, 8, 9]
>>> max(lstNum)
32
>>> min(lstNum)
5
>>> sum(lstNum)
54
```

```
>>> sorted(lstNum)
[5, 8, 9, 32]
>>> sorted(lstNum, reverse=True)
[32, 9, 8, 5]
```

```
>>> sorted(lstNum)
[5, 8, 9, 32]
>>> print(lstNum) #sorted() does not modify the original list.
[32, 5, 8, 9]
>>>
>>> lstNum.sort()
>>> print(lstNum) #.sort() modifies the original list.
[5, 8, 9, 32]
```

The **`split`** method

- The split method turns a single string into a list of substrings



```
>>> "CityU was founded in 1984.".split()
['CityU', 'was', 'founded', 'in', '1984.']
>>>
>>> "Hey, how is it going?".split(',')
['Hey', ' how is it going?']
```

The **`join`** method

- Join method is the reverse of the split method - turns a list of strings into a single string.

```
>>> " ".join(['CityU', 'was', 'founded', 'in', '1984.'])
'CityU was founded in 1984.'
>>> ",".join(['Hey', ' how is it going?'])
'Hey, how is it going?'
```

Tuple

The **`tuple`** Object

- Tuples, like lists, are ordered sequences of items.
- Tuples are written as comma-separated sequences enclosed in parentheses.
  - Tuples can also be written without parentheses.

```
>>> tp1 = ("CityU", 1984, 83, "Tat Chee Ave")
>>> tp2 = "CityU", 1984, 83, "Tat Chee Ave"
```

```
>>> tp1 == tp2  #== checks whether two objects are equal
True
```

The **`list`** and **`tuple`** functions

- Lists and tuples can be converted to one another.

```
>>> list( (2, 23, 6, 9, 8) )
[2, 23, 6, 9, 8]
>>> tuple( [2, 23, 6, 9, 8] )
(2, 23, 6, 9, 8)
```

```
>>> list("CityU")
['C', 'i', 't', 'y', 'U']
```

```
>>> list(1984)
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    list(1984)
TypeError: 'int' object is not iterable
```

Similarities between **`tuple`** and **`list`**

- Lists and tuples share some similar operations.
  - Items can be accessed by indices
  - Tuples can be sliced, concatenated, and repeated

```
>>> tp=(2, 23, 6, 9, 8)

>>> # Each item of a tuple can be accessed by its positive and negative indices
>>> tp[0]
2
>>> tp[2]
6
>>> tp[-1]
8
>>> # tuple slicing
>>> tp[2:4]
(6, 9)
>>> tp[:3]
(2, 23, 6)
>>> tp[1:]
(23, 6, 9, 8)
>>> tp[:]
(2, 23, 6, 9, 8)
```

```
>>> len(tp)
5
>>> tp.count(6)
1
>>> max(tp)
23
>>> min(tp)
2
>>> sum(tp)
48
```

```
>>> " ".join(('a', 'b'))
'a b'
>>> ",".join(('a', 'b'))
'a,b'
```

```
>>> (9, 3, 2) + ('a', 7)
(9, 3, 2, 'a', 7)
>>> ('a', 9)*3
('a', 9, 'a', 9, 'a', 9)
```

Similarities between **`tuple`** and **`list`**

- Similar to strings, Python does not allow out of bound indices for individual items in a list/tuple but does allow out of bound indices with slicing.

```
>>> lst = ['a', 'b', 'c']
>>> lst[5]
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    lst[5]
IndexError: list index out of range
>>>
>>> lst[ : 5]
['a', 'b', 'c']
```

```
>>> tp = ('a', 'b', 'c')
>>> tp[5]
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    tp[5]
IndexError: tuple index out of range
>>>
>>> tp[ : 5]
('a', 'b', 'c')
```

Differences between **`tuple`** and **`list`**

- Tuples cannot be modified in place
  - No **`append`**, **`extend`**, or **`insert`** methods for tuples.
  - Items of tuples cannot be directly deleted, sorted, or altered.

```
>>> tp = (2, 23, 6, 9, 8)
>>> lst = [2, 23, 6, 9, 8]
```

```
>>> lst[3]="sdlqwejrt" #Lists can be modified.
>>> print(lst)
[2, 23, 6, 'sdlqwejrt', 8]
```

```
>>> tp[3]="sdlqwejrt"
Traceback (most recent call last):
  File "<pyshell#146>", line 1, in <module>
    tp[3]="sdlqwejrt"
TypeError: 'tuple' object does not support item assignment
```

```
>>> school = "CityU"
>>> school[3] = "Q"  #BTW, strings can't be modified, either.
Traceback (most recent call last):
  File "<pyshell#149>", line 1, in <module>
    school[3] = "Q"  #BTW, strings can't be modified, either.
TypeError: 'str' object does not support item assignment
```

Differences between `tuple` and `list`

```
>>> tp = (2, 23, 6, 9, 8)
>>> lst = [2, 23, 6, 9, 8]
```

```
>>> lst.append(53)
>>> print(lst)
[2, 23, 6, 9, 8, 53]
>>> del lst[-1]
>>> print(lst)
[2, 23, 6, 9, 8]
```

```
>>> tp.append(53)
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    tp.append(53)
AttributeError: 'tuple' object has no attribute 'append'
>>>
>>> del tp[-1]
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    del tp[-1]
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> lst.sort() #altering the original list.
>>> print(lst)
[2, 6, 8, 9, 23]
>>> sorted(lst) #creating a new sorted list
[2, 6, 8, 9, 23]
```

```
>>> tp.sort()
Traceback (most recent call last):
  File "<pyshell#169>", line 1, in <module>
    tp.sort()
AttributeError: 'tuple' object has no attribute 'sort'
>>>
>>> sorted(tp)
[2, 6, 8, 9, 23]
>>> #sorted() also creates a new sorted listed when applied on tuples.
```

Immutable and Mutable Objects

- An **object** is an entity that holds data and has operations and/or methods that can manipulate the data.
    - E.g., numbers, strings, tuples, lists

- Objects that can be changed in place are **mutable**. Otherwise, they are **immutable**.

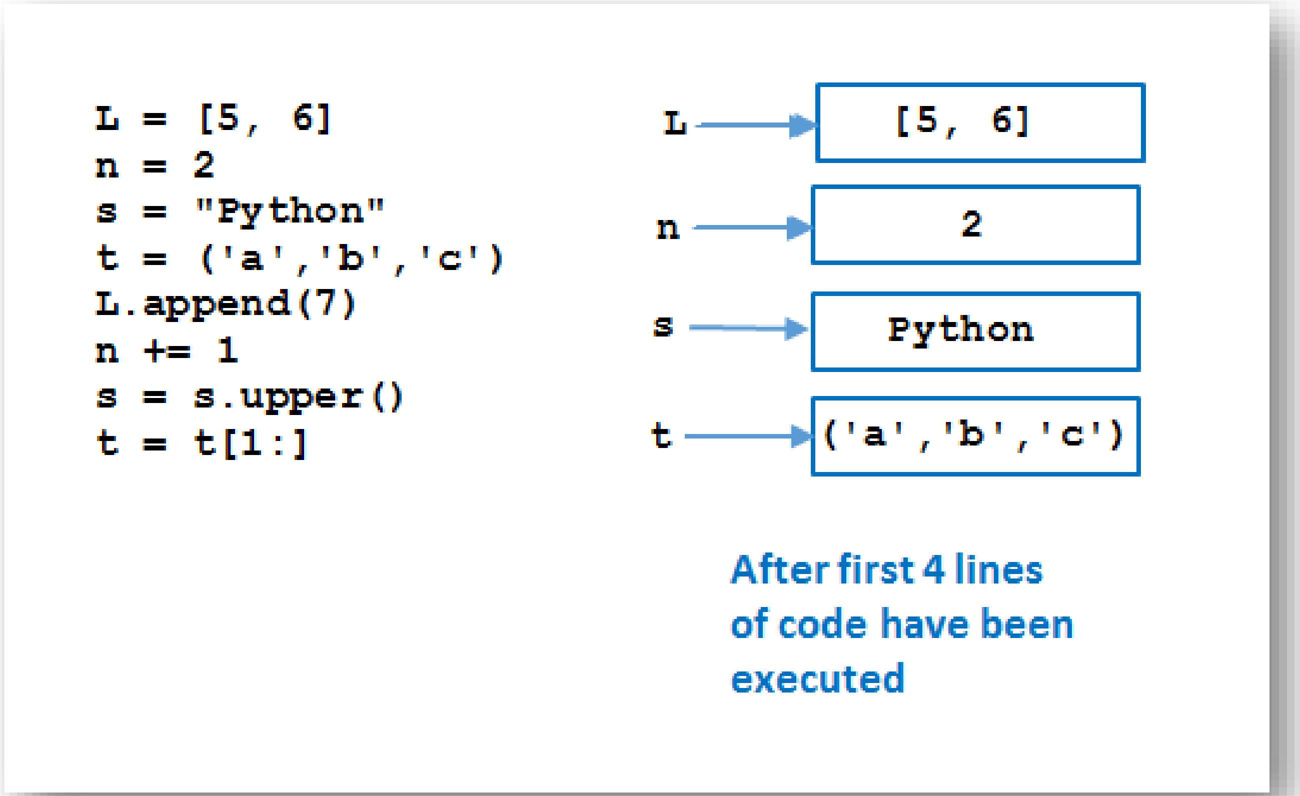
- **Lists are mutable. Numbers, strings and tuples are immutables.**

Immutable and Mutable Objects

- When a list is altered
  - Changes made to the object is in the list's memory location

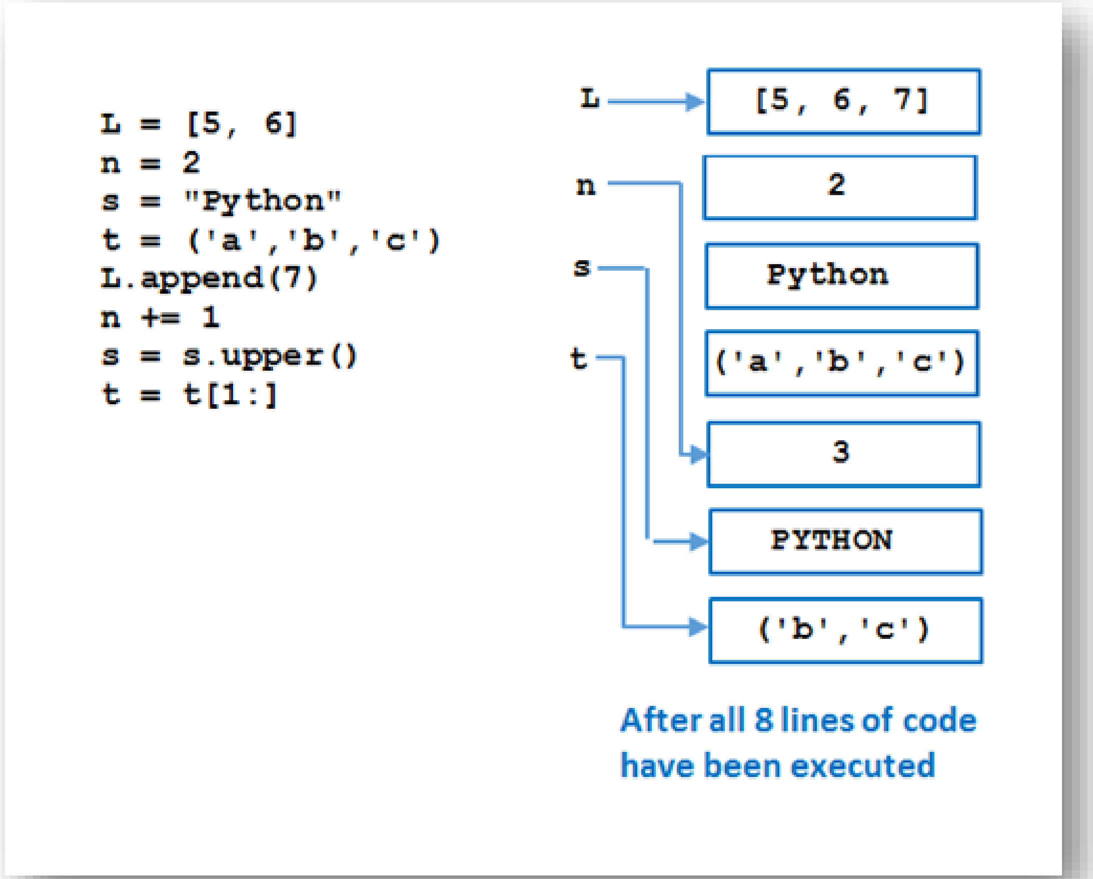
- When a number, string, or tuple is changed
  - Python designates a new memory location to hold the new value
  - The variable references that new object

Immutable and Mutable Objects



```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```

After first 4 lines of code have been executed



```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```

After all 8 lines of code have been executed

Copying Lists

- When copying lists, modifying the copied list will change the original list as well.

```
>>> #Lists are mutable!
>>> lst1=['a', 'b']
>>> lst2=lst1 #lst2 points to the same memory location as lst1.
>>> lst2[1]='c' #changing the value of the second item in lst2.
>>> print(lst1) #lst1 is also altered!!
['a', 'c']
```

- All because *lists are mutable*

Copying Lists

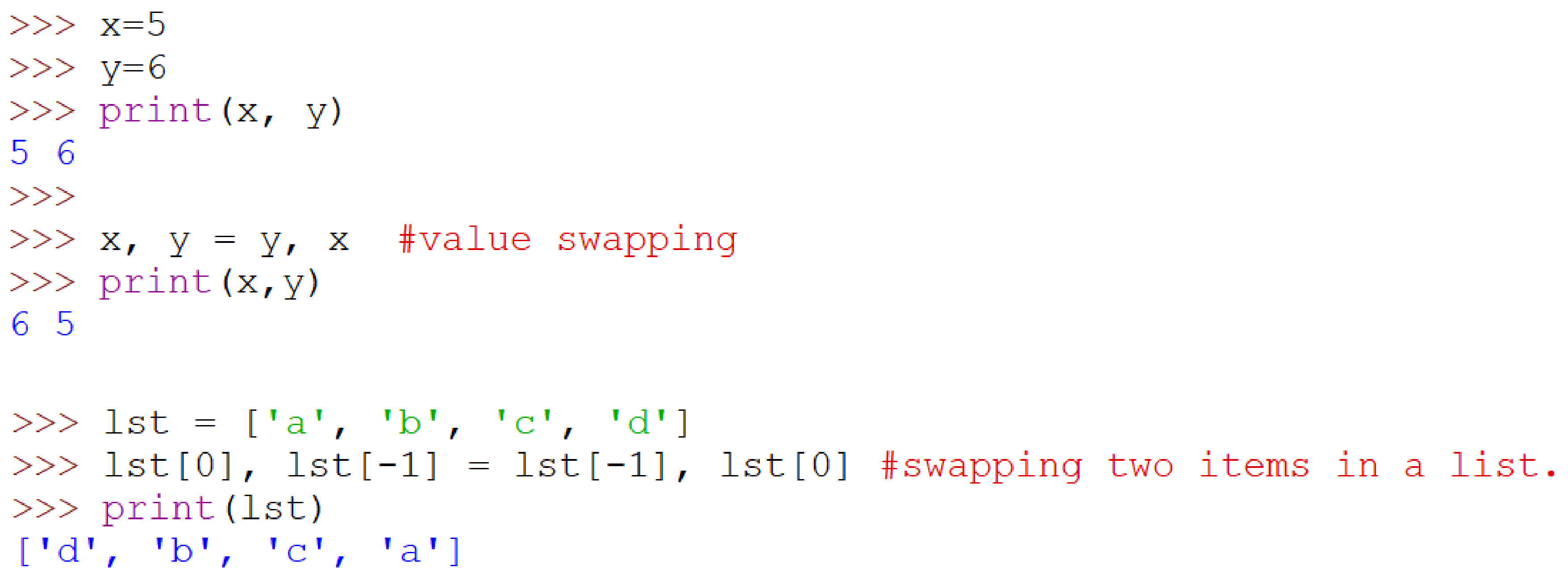
- Now note the change in line 2

```
>>> lst1=['a', 'b']
>>> lst2=list(lst1)  #lst2 now points to a different memory location.
>>> lst2[1]='c' #changing the value of the second item in lst2.
>>> print(lst1) #lst1 is unchanged!
['a', 'b']
```

- The 3rd line of the code will not affect the memory location occupied by *lst1*.

Value swapping

- Traditionally, swapping values for two variables is tedious - would involve an intermediate variable. But Python makes it incredibly easy.



```
>>> x=5
>>> y=6
>>> print(x, y)
5 6
>>>
>>> x, y = y, x  #value swapping
>>> print(x,y)
6 5

>>> lst = ['a', 'b', 'c', 'd']
>>> lst[0], lst[-1] = lst[-1], lst[0] #swapping two items in a list.
>>> print(lst)
['d', 'b', 'c', 'a']
```

Nested Lists

- Apart from numbers or strings, items in a list can be lists or tuples themselves.

```
>>> # A list of tuples
>>> lst = [("CityU", 1984), ("HKU", 1911), ("CUHK", 1963), ("HKUST", 1991)]
>>> lst[0][1] #The second item in the first item (tuple) of the list.
1984

>>> # A list of lists: a 2-dimensional array
>>> lst = [[1, 2, 3], [4, 5, 6], [8, 7, 9]]
>>> lst[1]
[4, 5, 6]
>>> lst[1][2] #The third item in the second list.
6
```

Summary: String, List, Tuple

- Lists, tuples and strings are ordered sequences of objects. They all have indices.

- Unlike strings that contain only characters, lists and tuples can contain any type of objects.

- Numbers, tuples, and strings are immutables. Lists are mutables - can be modified.

Lab 4

Example A

- Given a list of months.

```
month = ["January", "February", "March"]
```

- Output a list of abbreviations for the months.

```
['Jan', 'Feb', 'Mar']
```

Example B: acrostic poem

- Display the first letters of an acrostic poem.

- poem = `"Cuddly and fluffy \nAlways cheeky \nThey make us laugh \nSuperb at climbing"`

```
Cuddly and fluffy
Always cheeky
They make us laugh
Superb at climbing

CATS
```

Example C: nested list

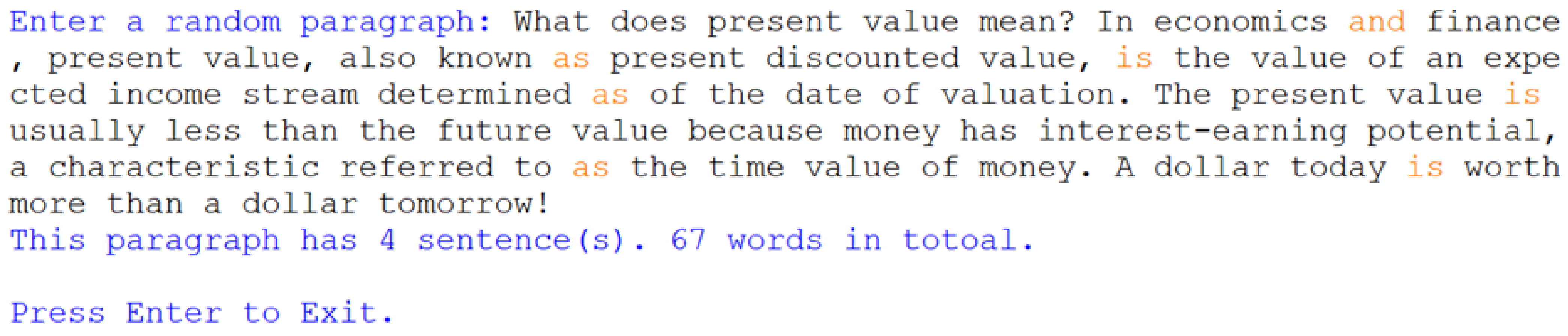
- `lst=["Hong Kong", ["Hong Kong Island", ["Kowloon",["&&&&", "Tat Chee Ave"]],"New Territories"]]`

- Replace ‘&&&&’ by “CityU” in the list and insert the street number.

```
['Hong Kong', ['Hong Kong Island', ['Kowloon', ['&&&&', 'Tat Chee Ave']], 'New Territories']]
['Hong Kong', ['Hong Kong Island', ['Kowloon', ['CityU', 'Tat Chee Ave']], 'New Territories']]
['Hong Kong', ['Hong Kong Island', ['Kowloon', ['CityU', 83, 'Tat Chee Ave']], 'New Territories']]
```

Classwork4

- Write a Python program that counts the number of sentences and the number of words in a paragraph entered by the user. Assume the end of each sentence contains only a single period, an exclaimation, or a question mark (.!?), and words are separated by single spaces only. Words connected by “-” are counted as one word. Exclude any spaces from either end of the paragraph. The output should resemble the following. Include the exit line. Upload the .py file and the output screenshot on Canvas.



```
Enter a random paragraph: What does present value mean? In economics and finance
, present value, also known as present discounted value, is the value of an expe
cted income stream determined as of the date of valuation. The present value is
usually less than the future value because money has interest-earning potential,
a characteristic referred to as the time value of money. A dollar today is worth
more than a dollar tomorrow!
This paragraph has 4 sentence(s). 67 words in totoal.

Press Enter to Exit.
```