

Programming Foundations in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 5



List, Tuple, and Dictionary Data Types

Lecture 5

Prepared by
Dr. Hamada I. AbdulWakel

Lecture

05

RECAP: LECTURE 4

- String data type.
- String features.
- String functions.

Lecture

05

LECTURE MAIN POINTS

- The need for new data types.
- The use of list, tuple and dictionary.
- The methods in each of these new data types.

Lecture

05

NEW PYTHON DATA TYPES

- A **list** allows the programmer to manipulate a **dynamic** sequence of data values of any types.
- A **tuple** is a special case of lists which allows the programmer to manipulate a **fixed** sequence of data values of any types.
- A **dictionary** organizes **dynamic** data values by association with other data values rather than by sequential position.

Lecture

05

LIST DATA TYPE/STRUCTURE

- **List** : Sequence of data values (items or elements).
- Some examples:
 - Shopping list for the grocery store.
 - To-do list.
 - Guest list for a wedding.
 - Text document, which is a list of lines.

Lecture

05

LIST FEATURES

- A list literal consists of a comma-separated list of items, enclosed in a pair of square brackets [].
- Each item in a list has a **unique index** that specifies its position.
 - Zero-based indexing: from 0 to length – 1
- Lists are mutable, means it can be changed after they are created.
- List item values can be duplicated and any data types.

Lecture

05

CREATING LISTS

- Empty list

```
L1 = [ ]
```

```
L2 = list()
```

- Same data type.

```
L3 = [1, 2, 3]
```

- Different data types.

```
L4 = [1, True, "ESU"]
```

```
L5 = list( [1, True, "ESU"] )
```

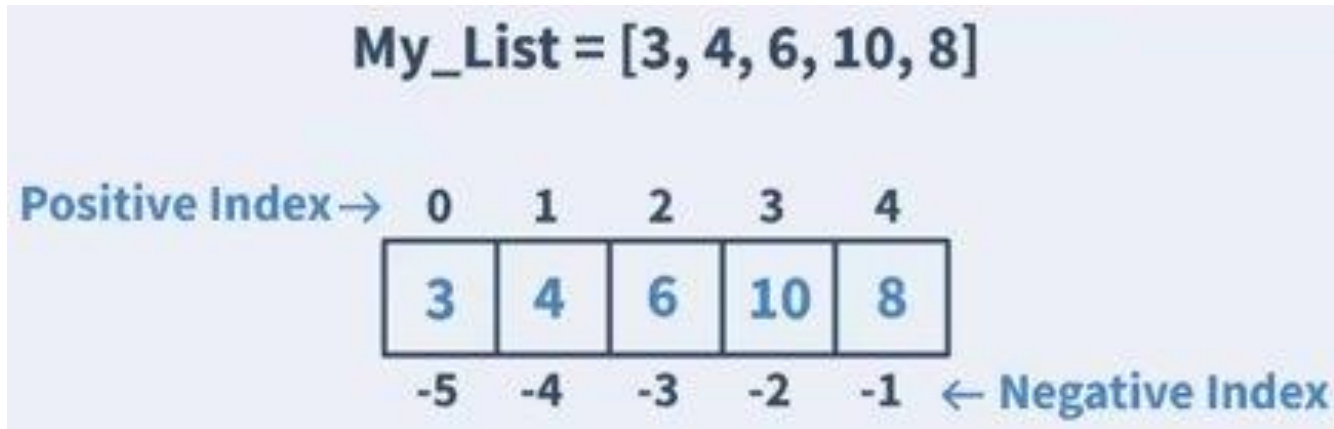
```
L6 = list( "ESU" ) #What about this?
```

Lecture

05

ACCESSING LIST ITEMS

- Using the index operator/subscript operator [].
- The subscript operator is used to reference the element in the list.



Lecture

05

ACCESSING LIST ITEMS

Cont'd

- Access items example:

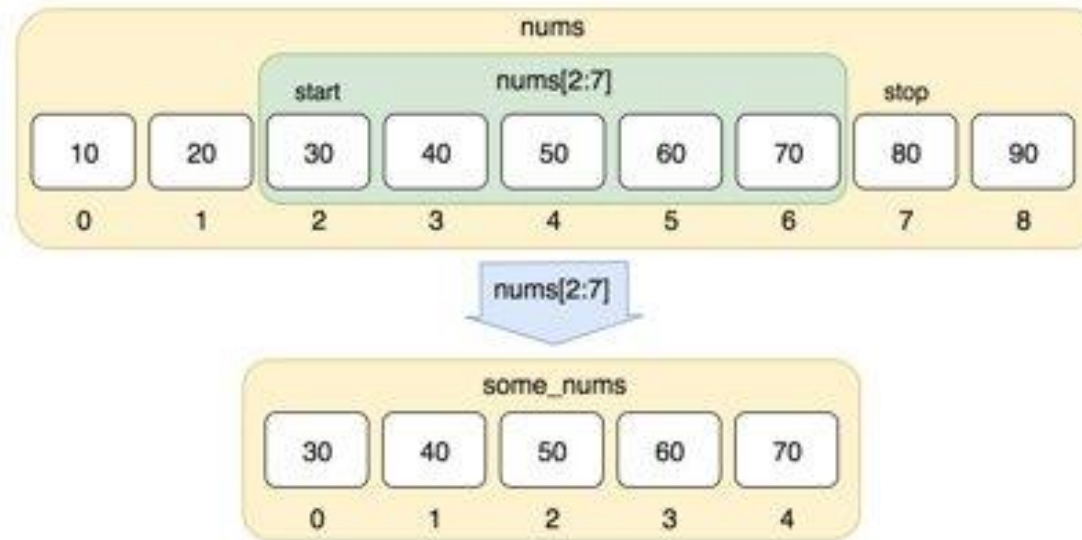
```
Numbers = [1, 2, 3, 4, 5]  
print( Numbers[5] )  
print( Numbers[-1] )  
print( Numbers[10] )  
Numbers[2] = 15  
print( Numbers[2] )  
Numbers[ 0 : 2 ] = [ 6, 7 ]  
print( Numbers )
```

Lecture

05

LIST SLICING

- Using the index operator/subscript operator `[]`.
 - Using this form `[ start : stop ]`



Lecture

05

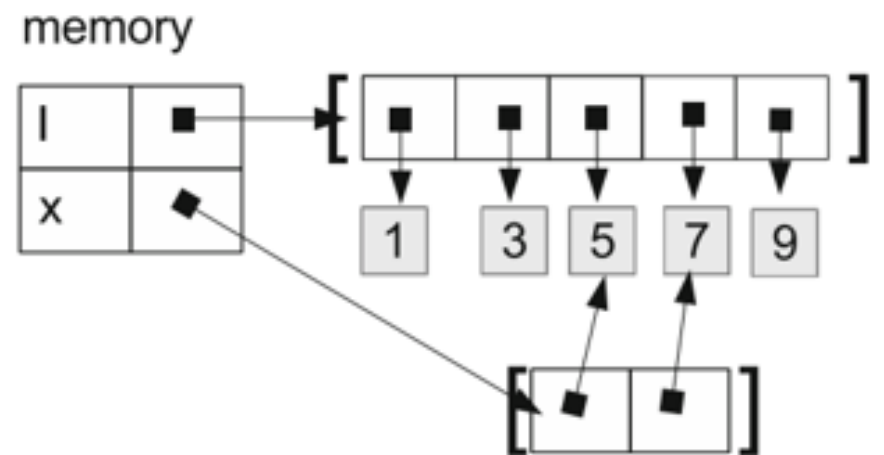
LIST SLICING

Cont'd

- Using this form [start : stop].
- The slicing generate a new list.

```
L1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
print( L1[ 3 :    ] )  
print( L1[    : 5 ] )  
print( L1[ 2 : 5 ] )
```

```
l = [1, 3, 5, 7, 9]  
x = l[2:4]
```



Lecture

05

LIST SLICING

Cont'd

- Using this form [start : stop : step]
 - If Step is **positive**:
 - Step – 1: the number of elements to skip
 - Start < Stop
 - Stop – 1: the stop index
 - If Step is **negative**:
 - Step + 1: the number of elements to skip
 - Start > Stop
 - Stop + 1: the stop index

Lecture

05

LIST SLICING

Cont'd

- Using this form [start : stop : step]
 - If Step is **positive**:
 - Step – 1 Start < Stop Stop – 1
- ```
L1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(L1[: : 1]) # all list
print(L1[: : 2]) # escape an element
print(L1[1 : 5 : 2])
```

## Lecture

## 05

## LIST SLICING

Cont'd

- Using this form [ start : stop : step]
    - If Step is **negative**:
      - Step + 1      Start > Stop      Stop +1
- ```
L1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print( L1[   :   : -1 ] )  # all list but reversed
print( L1[   :   : -2 ] )
print( L1[ -1 : -5 : -2 ] )
```

Lecture

05

LIST OPERATORS

- The list type supports operations including `+`, `*`, `in`, and `not in`.

```
L1 = [1, 2, 3, 4]
```

```
L2 = L1 + [5, 6]
```

```
print( L2 )
```

```
L2 = L1 * 2
```

```
print( L2 )
```

```
print( 1 in L1 )
```

```
print( 1 not in L1 )
```

Lecture

05

LIST FUNCTIONS

- **len(list):** to get the list length.
`L1 = [1, 2, 3, 4, 5]`
`print( len( L1 ) )`
- **min(list):** to get the minimum value in a list.
`print( min( L1 ) )`
- **max(list):** to get the maximum value in a list.
`print( max( L1 ) )`

LIST FUNCTIONS

Cont'd

- **sum(list):** to get the summation of the list elements.
`print( sum( L1 ) )`
- **append(element):** to add the element to the end of a list.
`L1.append(6)`
- **extend(list):** to add the elements of list L2 to the end of list L1.
`L2 = [6, 7, 8]`
`L1.extend( L2 )`
- **copy():** to make a copy of a list.
`L2 = L1.copy()`

LIST FUNCTIONS

Cont'd

- **insert(index, element):** to insert element at specified index.
`L1 = [1, 2, 3, 4, 5]`
`L1.insert( 6, 2 )`
- **pop(index):** to remove and returns the element at index from list L.
`L1.pop(3)`
- **count(element):** to count the number of occurrence of element.
`X = L1.count(3)`
- **clear():** to remove all elements in a list.
`L1.clear()`

LIST FUNCTIONS

Cont'd

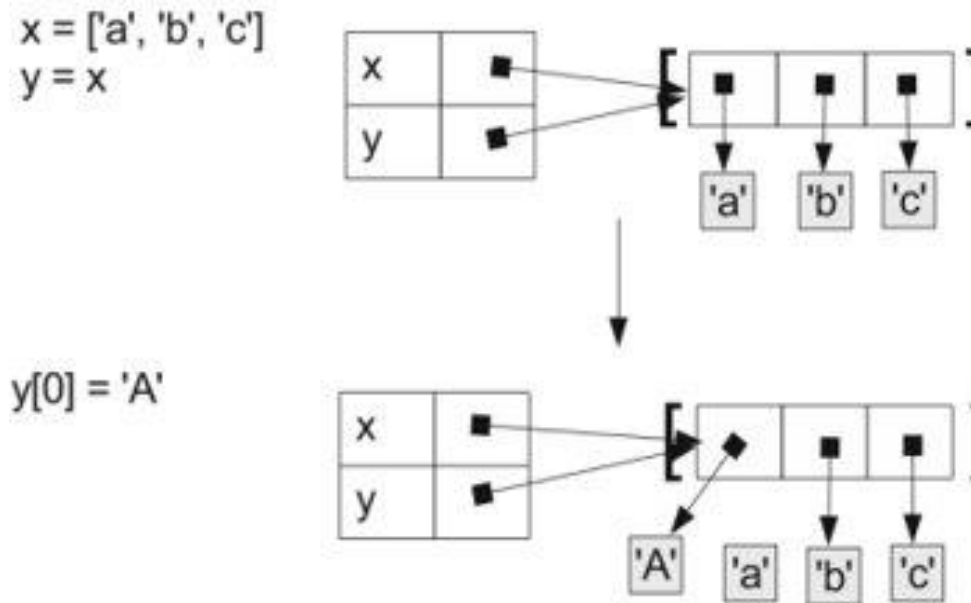
- **index(element):** to get the index of an element.
`L1 = [1, 2, 3, 4, 5]`
`L1.index( 2 )`
- **remove(element):** to remove the first occurrence of the element.
`L1.remove(3)`
- **reverse():** to reverse the list elements.
`L1.reverse()`
- **sort():** to sort the list elements in ascending order.
`L1.sort()` `L1.sort(reverse = True)`

Lecture

05

LIST EQUALITY

- Mutable property of lists. If two aliases refer to the same list object, any modification made by any alias, the other can refer to this object.



Lecture

05

LIST EQUALITY

Cont'd

- Prevent aliasing side effects:

```
L1 = [5, 6, 2, 7, 1, 9]
```

```
L2 = []
```

```
for x in L1:
```

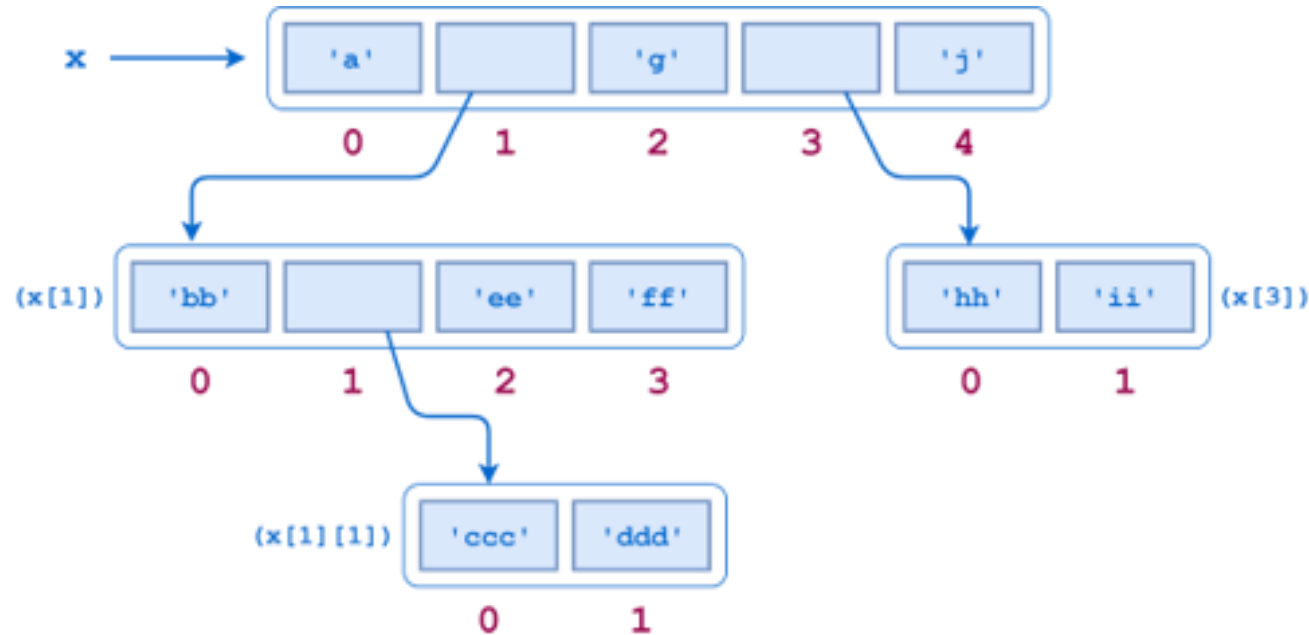
```
    L2.append(x)
```

Lecture

05

COMPLEX LIST

- An element in a list can be any sort of object, therefore a list can contain sublists.



- An element in a list can be any sort of object, therefore a list can contain sublists.

```
L1 = [ 1, 2, 3, 4, 5, [12, 13, 23] ]
```

```
print(L1[ 0 ])
```

```
print(L1[ 5 ][ 1 ])
```

Lecture

05

TUPLE DATA TYPE/STRUCTURE

- A **tuple** resembles a list but is **immutable**.
- Program execution is **faster** when manipulating a tuple than it is for the equivalent list.
- Sometimes you **don't want data to be modified**. If the values in the collection are meant to remain constant for the life of the program, using a tuple instead of a list guards against accidental modification.

Lecture

05

TUPLE DATA TYPE/STRUCTURE

Cont'd

- Tuple Immutable means, we can not add, delete, replace, reorder elements.

```
T0 = (2)
```

```
T1 = (2,)
```

```
print(type(T0), type(T1))
```

```
T2 = (11, 22, 33)
```

```
T3 = tuple([1, 2, 3, 4])
```

```
T4 = tuple("abc")
```

```
T2[0] = 44 #error 'tuple' object does not support item assignment
```

Lecture

05

TUPLE OPERATORS AND FUNCTIONS

- Most of the operators and functions used with lists can be used in a similar fashion with tuples, except the functions that make any change.

```
T1 = (11, 22, 33)
```

```
print(T1)
```

```
print(T1[0])
```

```
print(T1[0:2])
```

```
print(min(T1))
```

```
print(55 in T1)
```

```
T1.append(44) #AttributeError: 'tuple' object has no attribute 'append'
```

Lecture

05

PACKING AND UNPACKING TUPLE
OPERATIONS

- In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

```
T1 = ('Everyone', 'Smart', 'University') #pack
(s1, s2, s3) = T1 #unpack
print(s1) #Everyone
(s1, s2, s3, s4) = T3 # error too many values to unpack
(s1, *others) = T1
print(others) # will give ['Smart', 'University'] as an item
```

Lecture

05

DICTIONARY DATA TYPE/STRUCTURE

- A **dictionary** organizes elements between curly braces { }.
- Data structures organized by association are also called **tables** or **association lists**.
- In Python, a dictionary associates a set of **keys** with **data values**.
 - Keys must be immutable type (i.e., number, string, tuple)
 - Keys are unique.
 - Values can be of any type.

Lecture

05

DICTIONARY DATA TYPE/STRUCTURE

Cont'd

- A dictionary creation/declaration.

```
dict0 = {}
```

```
dict1 = dict() #empty dictionaries
```

```
dict2 = { 1 : 123 , "a" : 456 , 2 : True }
```

```
dict3 = dict( [ (1 , 123), ("a" , 456), (2 , True) ] )
```

Lecture

05

ACCESSING DICTIONARY ELEMENTS

- Using the index operator/subscript operator `[]` with element associated key.

```
friends = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }  
print(friends)  
print(friends[1]) #error: elements are not accessed by numerical index.  
print(friends['Mohamed'])  
print(friends['Abdulrahman']) #error: keyerror 'Abdulrahman'
```

Lecture

05

DICTIONARY OPERATORS

- Using **in**, **not in**, **==**, **!=** operators with dictionary.
friends1 = { 'Mohamed': 30 , 'Ahmed': 25 , 'Ali': 40 }
friends2 = { 'Mohamed': 30 , 'Ahmed': 25 , 'Ali': 40 }
print('Abdulrahman' in friends1)
print('Ali' not in friends1)
print(friends1 == friends2)
friends2 = { 'Mohamed': 35 , 'Ahmed': 25 , 'Ali': 40 }
print(friends1 == friends2)

Lecture

05

DICTIONARY OPERATORS

Cont'd

- Keys in dictionary are unique. If two or more elements have the same key, python will deal with the last element.

```
friends = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40, 'Ali': 44 }  
print(friends)
```

- Adding a new element using keys.

```
friends = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }  
friends['Abdulrahman'] = 33
```

- Updating the value for a specific key.

```
friends['Abdulrahman'] = 35
```

Lecture

05

DICTIONARY FUNCTIONS

- **get(key):** to the value in the given key.
`D1 = {1 : 1 , 2 : "A" , 3 : True}`
`D1.get( 1 )`
- **has_key(key):** to check if the key used in the dictionary or not.
`D1.has_key(3)`
- **items():** to return a list of tuples containing the keys and values.
`D1.items()`
- **values():** to return a list contains only the values.
`D1.values()`

Lecture

05

DICTIONARY FUNCTIONS

Cont'd

- **clear()**: to remove all elements in a dictionary.
D1 = {1 : 1 , 2 : "A" , 3 : True}
D1.clear()
- **pop(key)**: to remove and return the value in the given key.
D1.pop(3)
D1.pop(4) **#KeyError**
- **popitem()**: to remove and return the value of the last element.
D1.popitem()

DICTIONARY FUNCTIONS

Cont'd

- **keys()**: to return a list contains only the keys.
`D1.keys( )`
- **update({k : v})**: to add an element at the end of a dictionary.
`D1.update({4 : "A"})`
- **fromkeys(keys, value)**: to create a dictionary using a given keys and values.
`friends2 = [ 'Mohamed', 'Ahmed','Ali']`
`v = "X"`
`d = dict.fromkeys(friends2, v)`

Lecture

05

DICTIONARY FUNCTIONS

Cont'd

- **del element:** to delete the given element.

```
del D1[1]
```

- **len(dictionary):** to get the length of a dictionary.

```
D1 = {1 : 1 , 2 : "A" , 3 : True}
```

```
len( D1 )
```

- **dict(zip(L1, L2)):** to create a dictionary using two lists L1 and L2.

```
L1 = ["A", 1, "B"]
```

```
L2 = [1, 2, 3]
```

```
D1 = dict( zip( L1, L2 ) )
```

Lecture

05

LOOPING IN DICTIONARY

- Traversing the dictionary entries:

```
friends1 = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }
```

```
for elem in friends1:
```

```
    print("Key = ", elem, " Value = ", friends1[elem])
```

```
for key in friends1.keys():
```

```
    print(key)
```

```
for val in friends1.values():
```

```
    print(val)
```

Lecture

05

DICTIONARY COMPERHENSION
OPERATION

- Dictionary comprehension is an elegant and concise way to create a new dictionary from an iterable.
- Dictionary comprehension consists of an expression pair (**key: value**) followed by a **for statement** inside curly braces {}.

```
squares = { x : x * x  for  x  in  range(6) }  
print(squares)
```

Lecture

05

QUESTIONS FOR ONLINE DISCUSSION

- Write code to get the largest number from a list.
- Write code that takes two lists and returns True if they have at least one common member.
- Write code to check if each number is prime in a given list of numbers. Return True if all numbers are prime otherwise False.
- Write code to create a dictionary of your courses and display them.
- Write code to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x). Where n is given by the user.



Thank You

