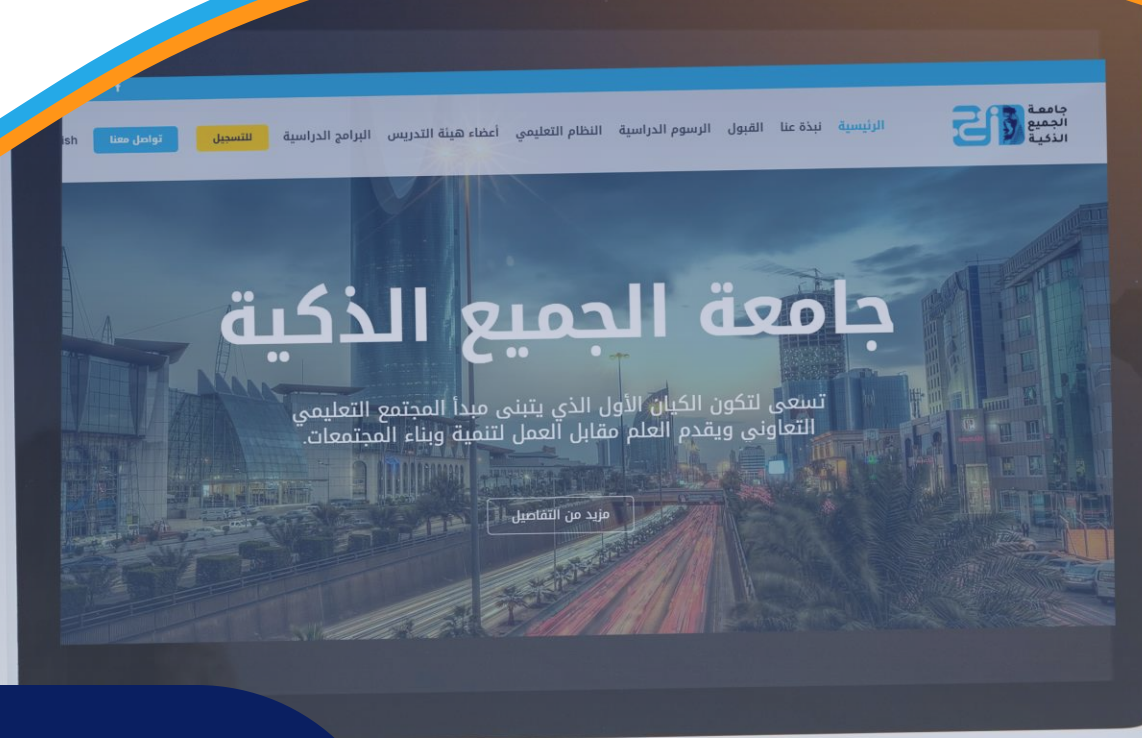




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



Lists and Dictionaries Data Types
Prepared by Dr. Hamada I. AbdulWakel
Lecture 5

CYB550
Programming Foundations In
Python

- Investigate new python data types.
- Explain how list and dictionary work.
- Use methods in these new data types.
- Decide whether a list or a dictionary is an appropriate data structure for a given application.



- A **list** allows the programmer to manipulate a sequence of data values of any types.
- A ~~**dictionary**~~ organizes data values by association with other data values rather than by sequential position.

- **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.
- Each item in a list has a unique index that specifies its position (from 0 to length – 1).

- Creating and printing list:
 - Using range() method.
 - Using []
 - Using list() method.
 - Using print() method.

- Code:
 - `L1 = range(6)`
 - `L2 = [1, 2, 3, 4]`
 - `L3 = ["this is an item", 12]`
 - `L4 = list()`
 - `L5 = list( [10, 20, 30, 40] )`
 - `L6 = list("python")`
 - `print(L1)`
 - `print(L6)`



- Accessing, replacing, and slicing elements in list:
 - List is mutable.
 - Using the index operator [].
 - The subscript operator is used to reference the target which is an element in the list.

- Code:
 - `Numbers = range(5)`
 - `Numbers[5]`
 - `Numbers[-1]`
 - `Numbers[ : 3]`
 - `Numbers[2 : ]`
 - `Numbers[2] = 15`

- Checking for list elements:
 - Using in operator to determine an element's presence or absence.
 - Use method index to locate an element's position in a list.
 - >, <, ==

- Code:
 - `L1 = [10, 20, 30, 40]`
`print(10 in L1)`
`print(50 in L1)`
`if 10 in L1:`
 `print(L1.index(10))`
`print(20 not in L1)`
`L2 = [1, 2, 3, 4]`
`print(L1 == L2)`



- List methods:

Method	Description
len(L)	Get the length of list L
min(L)	Get the minimum element in list L
max(L)	Get the maximum element in list L
sum(L)	Get the summation of elements in list L

- Code:
 - `L1 = [10, 20, 30, 40]`
`print(len(L1))`
`print(min(L1))`
`print(max(L1))`
`print(sum(L1))`

- List methods:

Method	Description
L.append(X)	Adds X element to the end of list L.
L1.extend(L2)	Adds the elements of list L2 to the end of list L1.
L.insert(index, X)	Inserts X element at index if index is less than the length of list L. Otherwise, inserts X at the end of L.
L.pop(index)	Removes and returns the element at index from list L. If index to specify, it removes and returns the element at the end of L.
L.count(X)	Counts the number of occurrence of element X in list L.

- Code:
 - `L1 = [10, 20, 30, 40]`
 - `L1.insert(1, 50)`
 - `L1.append(10)`
 - `L1.extend([1, 2, 3, 4])`
 - `L1.pop()`
 - `L1.pop(1)`
 - `L1.count(20)`

- List methods:

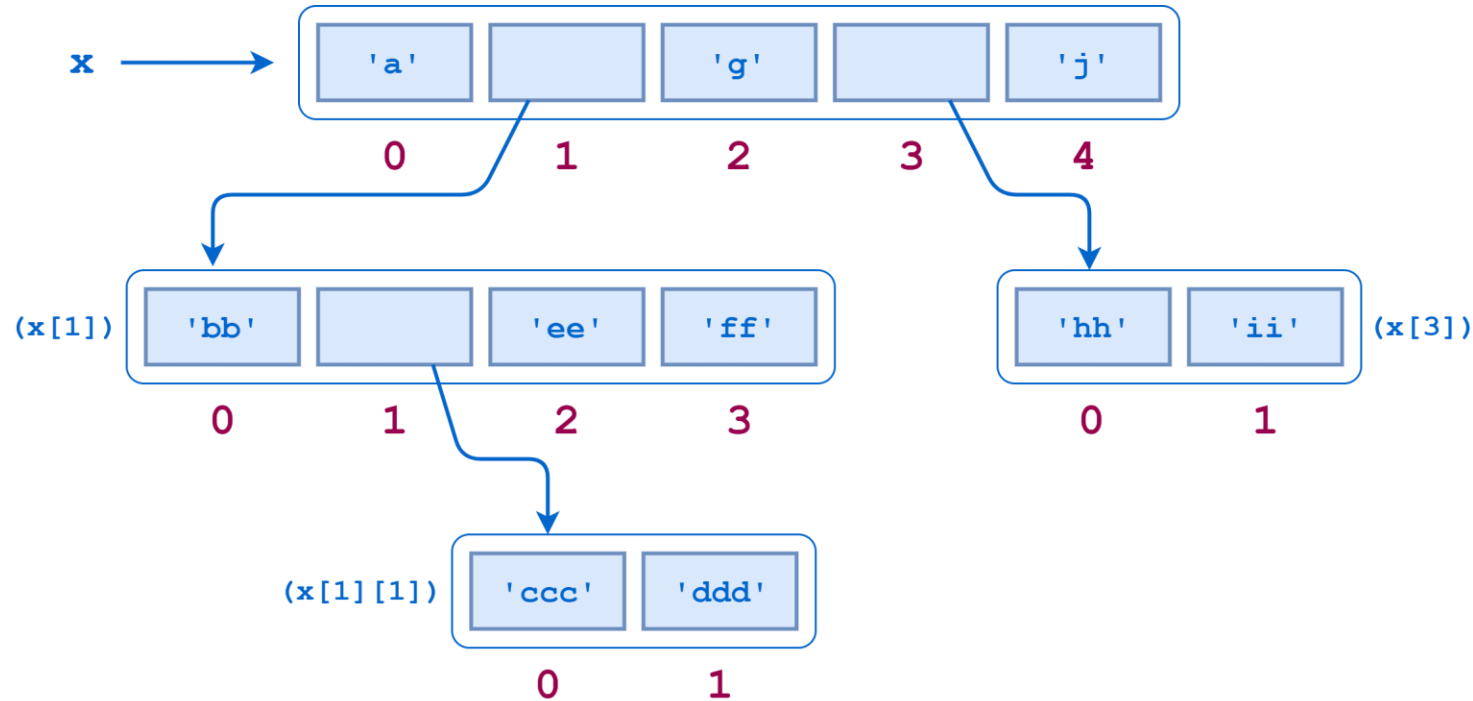
Method	Description
L.index(X)	Returns the index of the first occurrence of element X in the list L.
L.remove(X)	Removes the first occurrence of the element X from the list L.
L.reverse()	Reverses the list L.
L.sort()	Sorts the elements in the list L in ascending order.

- Code:
 - `L1 = [5, 6, 2, 7, 1, 9]`
`print(L1.index(2))`
`L1.reverse()`
`print(L1)`
`L1.remove(2)`
`print(L1)`
`L1.sort()`
`print(L1)`

- Mutable property of lists. If two aliases refer to the same list object, any modification make by any alias, the other can refer this object.
 - `L1 = [5, 6, 2, 7, 1, 9]`
`L2 = L1`
`L1[1] = 10`
`print(L2)`

- Prevent aliasing side effects:
 - `L1 = [5, 6, 2, 7, 1, 9]`
`L2 = []`
for x in L1:
 `L2.append(x)`

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



- A **tuple** resembles a list but is **immutable**.
- Tuples are defined by enclosing the elements in parentheses () instead of square brackets [].
- 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.

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

- Most of the operators and functions used with lists can be used in a similar fashion with tuples.
 - `T1 = (11, 22, 33)`
`print(T1)`
`print(T1[0])`
`print(T1[0:2])`
`print(min(T1))`
`print(55 in T1)`

- Most of the operators and functions used with lists can be used in a similar fashion with tuples.
 - `T1 = (2)`
`T2 = (2,)`
`print(type(T1), type(T2))`
`T3 = ('foo', 'bar', 'baz', 'qux')`
`(s1, s2, s3, s4) = T3`
`print(s1) #foo`
`(s1, s2, s3) = T3` **# error too many values to unpack**

- Code:
 - `T1 = 1, 2, 3`
`print(T1)`
`x1, x2, x3 = T1`
`print(x1)`
`x1, x2, x3 = 4, 5, 6`
`print(x1)`



- A **dictionary** organizes information by **association**, not position.
 - When you use a dictionary to look up the definition of “mammal,” you don’t start at page 1; instead, you turn directly to the words beginning with “M”.
- Data structures organized by association are also called **tables** or **association lists**.
- In Python, a dictionary associates a set of keys with data values.

- A dictionary declaration.

- Code

```
dict0 = {}
```

```
dict1 = { <key>: <value>, <key>: <value>, . . . . , <key>:<value> }
```

```
dict2 = dict([( <key>, <value> ), ( <key>, <value> ), . . . , ( <key>, <value> )])
```

```
dict3 = dict(keystring1='value', keystring2='value', . . . , keystringn='value')
```





- Accessing dictionary entries:
 - 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'

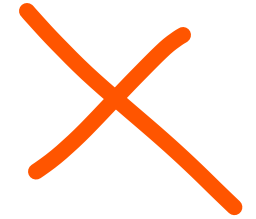


- Adding/removing entries to/from a dictionary:
 - `friends = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }`
`friends['Abdulrahman'] = 33`
`print(friends)`
`del friends['Mohamed']`
`print(friends)`
- Updating the value for a specific key, not adding a new entry, keys are unique
`friends['Abdulrahman'] = 35`



- Getting keys/values of a dictionary:
 - friends = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }
 - print(len(friends))
 - print(friends.keys())
 - print(friends.values())

- 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)





Method	Description
d.popitem()	Returns randomly select item from dictionary and also remove the selected item
d.clear()	Delete everything from dictionary
del d	Delete the dictionary d itself
d.get(key)	Returns value of key, if key is not found it returns None, instead on throwing KeyError exception
d.pop(key)	Removes the item from the dictionary, if key is not found KeyError will be thrown
d.has_key(key)	Returns True if the given key exists or False otherwise.
d.items()	Returns a list of tuples containing the keys and values for each entry.
dict(zip(list1, list2))	Aggregate two lists to a dictionary

- Code:
 - friends1 = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }
 - friends1.popitem()
 - friends1.get("Ali")
 - friends1.pop("Ali")
 - friends1.has_key("Ali")
 - friends1.items()
 - friends1.clear()
 - del friends1

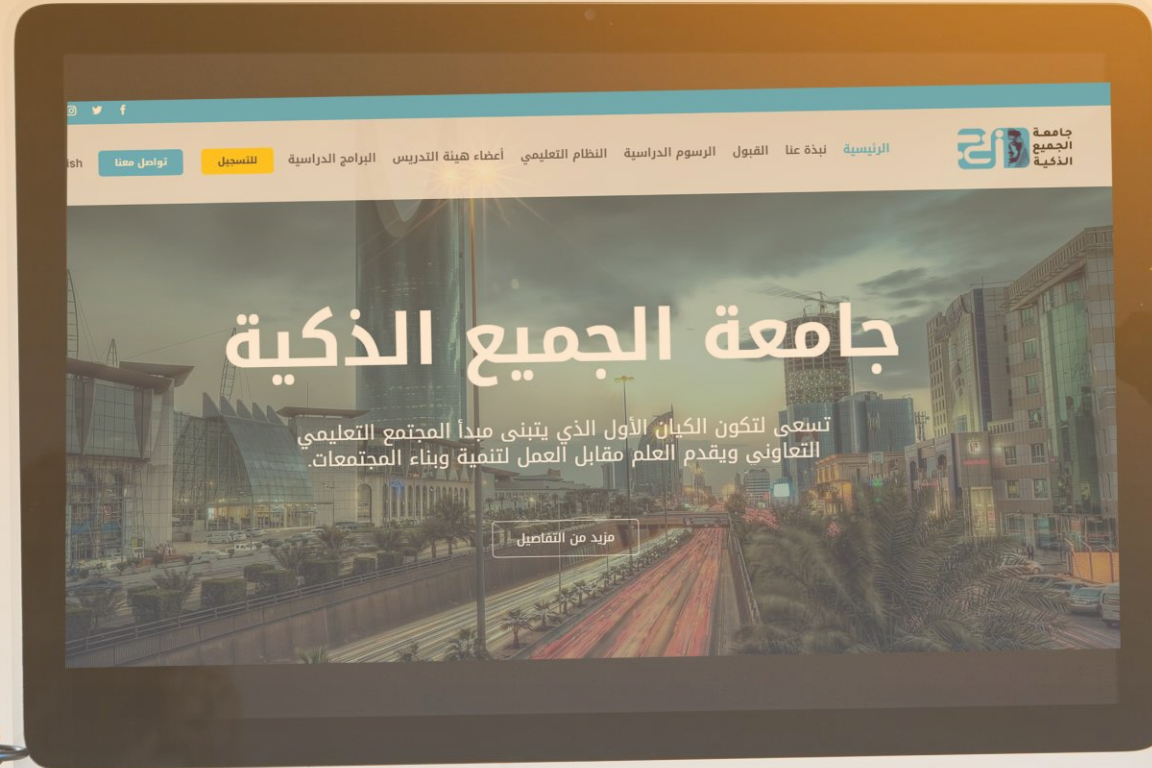


- Traversing the dictionary entries:
 - friends1 = { 'Mohamed': 30, 'Ahmed': 25, 'Ali': 40 }
 - for key in friends1:
 print(key, ":", friends1[key])

 - for val in friends1.values():
 print(val)



- 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)
```



جامعة الجميع الذكية  
EVERYONE'S SMART UNIVERSITY

THANK YOU