

Collections



Session Objectives



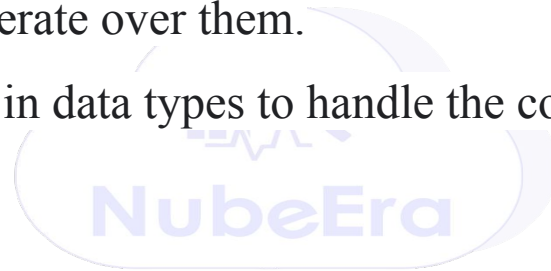
- Concept
- Collection
 - List
 - Tuples
 - Dictionary
 - Set
- Summary





● Concept

- The collection Module in Python provides **different types of containers**. A Container is an object that is used to store different objects and provide a way to access the contained objects and iterate over them.
- Python provides 4 built in data types to handle the collections(container)
 - List
 - Tuple
 - Dictionary
 - Set



CORPORATE



RECORDED



INTERACTIVE



● List

- Lists are used to store multiple items in a single variable.
- **Lists** are just like dynamic sized arrays,
- A single list may **contain Data Types** like Integers, Strings, as well as Objects.
- Lists are mutable, and hence, they can be altered even after their creation.
- List items are **ordered, changeable, and allow duplicate** values.
- List items are **indexed**, the first item has index **[0]**, the second item has index **[1]** etc.
- Example:
 - `thislist = ["apple", "banana", "cherry"]` `#['apple', 'banana', 'cherry']`
 - `thislist = ["abc", 1, 12.3, true, "xyz"]` `#['apple', 1 , 12.3, true , 'xyz']`





1. Methods of list

a. constructor

i. `mylist= list(("apple", "banana", "cherry"))`

b. len()

i. `print(len(mylist))` #how many items in list-3

c. type()

i. `print(type(mylist))` #<class 'list'>

d. insert()

i. `mylist.insert(1,"mango")` #['apple', 'mango', 'banana', 'cherry']

e. append()

i. `mylist.append("mango")` #['apple', 'banana', 'cherry', 'mango']

f. remove()

i. `mylist.remove("apple")` #['banana', 'cherry']





g. pop()

i. `mylist.pop(1)`

```
print(mylist)    #[‘apple’, ‘cherry’]
```

h. extend()

i. `mylist= list(("apple", "banana", "cherry"))`

```
yourlist= list(("kiwi", "mango"))
```

```
mylist.extend(yourlist)    #newlist=mylist+yourlist
```

```
print(mylist)
```

Output: `[‘apple’, ‘banana’, ‘cherry’, ‘kiwi’, ‘mango’]`





● Accessing items from list

- `mylist= list(("apple", "banana", "mango", "cherry"))`
 - `print(mylist)` `#['apple', 'banana', 'mango', 'cherry']`
 - `print(mylist[0])` `#['apple']`
 - `print(mylist[-1])` `#['cherry']`
 - `print(mylist[0:2])` `#['apple', 'banana']`
 - `print(mylist[1:])` `#['banana', 'mango', 'cherry']`
 - `print(mylist[-3:-1])` `#['banana', 'mango']`





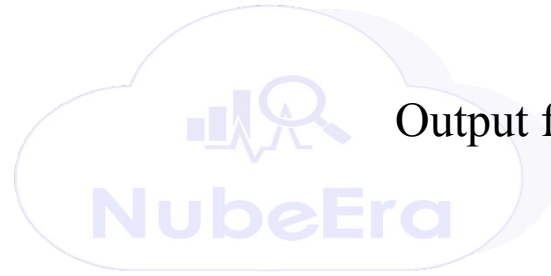
● Loop through list

- `mylist= list(("apple", "banana", "mango", "cherry"))`

- `for x in mylist:`
 `print(x)`

- `i=0`
 `while i< len(mylist):`
 `print(x)`

- `print(x) for x in mylist:`



Output for all: apple
 banana
 mango
 cherry



CORPORATE



RECORDED



INTERACTIVE



● tuple

- **Tuple** is a collection of Python objects much like a list.
- Tuples are used to store multiple items in a single variable and **allow duplicate** values.
- The sequence of values stored in a tuple can be of any type, and they are **indexed** by integers.
- A tuple is a collection which is **ordered and unchangeable**.
- Example:
 - `thistuple = ("apple", "banana", "cherry")` `#('apple', 'banana', 'cherry')`
 - `mtuple = ("abc", 1, 12.3, true, "xyz")` `#('apple', 1 , 12.3, true , 'xyz')`



CORPORATE



RECORDED



INTERACTIVE



1. Methods of tuple

a. constructor

i. `mytuple= tuple(("apple", "banana", "cherry"))`

b. `len()`

i. `print(len(mytuple))` #how many items in tuple-3

c. `type()`

i. `print(type(mytuple))` #<class 'tuple'>

d. `insert()/append()/remove()`

i. `mylist=list(mytuple)`

`mylist.insert(1,"mango")`

`mytuple=tuple(mylist)`

`print(mytuple)` #('apple', 'mango', 'banana', 'cherry')



CORPORATE



RECORDED



INTERACTIVE



● Accessing items from tuple

- mytuple= tuple(("apple", "banana", "mango", "cherry"))
 - print(mytuple) #('apple', 'banana', 'mango', 'cherry')
 - print(mytuple[0]) #('apple')
 - print(mytuple[-1]) #('cherry')
 - print(mytuple[0:2]) #('apple', 'banana')
 - print(mytuple[1:]) #('banana', 'mango', 'cherry')
 - print(mytuple[-3:-1]) #('banana', 'mango')





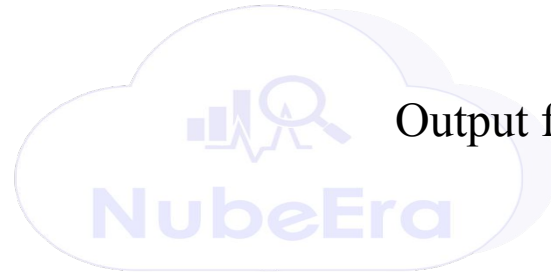
● Loop through tuple

- mytuple= tuple(("apple", "banana", "mango", "cherry"))

- for x in mytuple:
 print(x)

- i=0
 while i< len(mytuple):
 print(x)

- print(x) for x in mytuple:



Output for all: apple
 banana
 mango
 cherry



CORPORATE



RECORDED



INTERACTIVE



● Sets

- Sets are used to store multiple items in a single variable.
- A set is a collection which is both *unordered* and *unindexed*.
- Set do **not allow duplicate** values in set.
- Example:
 - `thisset = {"apple", "banana", "cherry"}` `#('banana', 'cherry', 'apple')`
 - `myset = {"abc", 1, 12.3, true, "xyz"}` `#('apple', 12.3, 1, 'xyz', true)`
- It display the output in **random order** and duplicate value ignored.





1. Methods of set

a. constructor

i. `myset= set(("apple", "banana", "cherry"))`

b. len()

i. `print(len(myset))` `#how many items in set-3`

c. type()

i. `print(type(myset))` `#<class 'set'>`

d. add()

i. `myset.add("mango")` `#{'apple', 'banana', 'cherry', 'mango'}`

e. pop()

i. `mylist.pop()` `#{'apple', 'banana' }randomly delete any item`

f. remove()

i. `myset.remove("apple")` `#{'banana', 'cherry'}`





g. union()

```
i. myset= set(("apple", "banana", "cherry"))  
yourset= set(("kiwi", "mango", "apple"))  
newset=myset.union(yourlist)           #newlist=mylist+yourlist  
print(newset)
```

Output: {'apple', 'banana', 'cherry', 'kiwi', 'mango'}

h. intersection()

```
i. newset=myset.intersection(yourlist)  
print(newset)           #{'apple'} item present in both sets
```





● Loop through set

- `myset= set(("apple", "banana", "mango", "cherry"))`
- `for x in myset:`
 `print(x)`

- `i=0`
 `while i< len(myset):`
 `print(x)`
- `print(x) for x in myset:`



Output for all: apple
banana
mango
cherry

(may different because it display randomly)



CORPORATE



RECORDED



INTERACTIVE



● Dictionary

- Dictionaries are used to store data values in **key:value** pairs.
- It is a collection which is **ordered, changeable and does not allow duplicates**.
- Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be *immutable*.
- Example:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

```
#{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```



CORPORATE



RECORDED



INTERACTIVE



1. Methods of Dictionary

a. constructor

i. `mydict = dict({"Brand": "Ford", "Year": 1999})`

b. len()

i. `print(len(mydict))` #how many items in dictionary-2

c. type()

i. `print(type(mydict))` #<class 'dict'>

d. Add new key

i. `mydict["colour"]="Black"` #add new key "colour" with value "Black"

e. update()

i. `mydict.update({"Year":1989})` #update key "Year" with value 1989

f. pop()

i. `mydict.pop("Year")` #Remove key "Year"





g. keys()

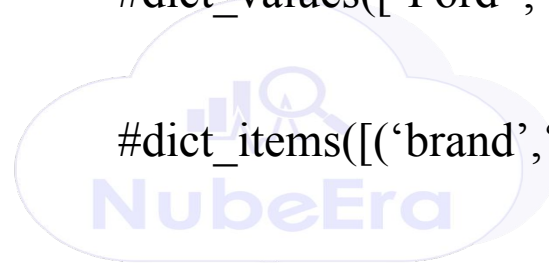
i. `print(mydict.keys())` `#dict_keys(['Brand', 'Year'])`

h. values()

i. `print(mydict.values())` `#dict_values(['Ford', 1999])`

i. items()

i. `print(mydict.items())` `#dict_items([('brand', 'Ford'), ('Year', 1999)])`



CORPORATE



RECORDED



INTERACTIVE



● Loop through Dictionary

- `mydict = dict({"Brand": "Ford", "Year": 1999})`
- `for x in mydict:`
 `print(x)`

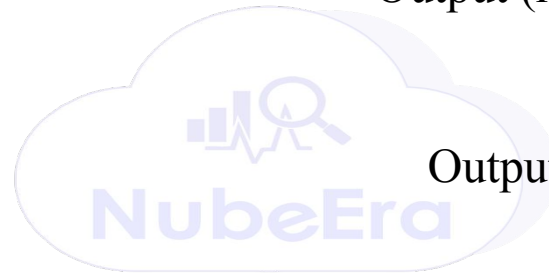
Output (keys): Brand
Year

- `for x in mydict:`
 `print(mydict[x])`

Output (values): Ford
1999

- `for x,y in mydict.items():`
 `print(x,y)`

Output (items): Brand Ford
Year 1999



CORPORATE



RECORDED



INTERACTIVE

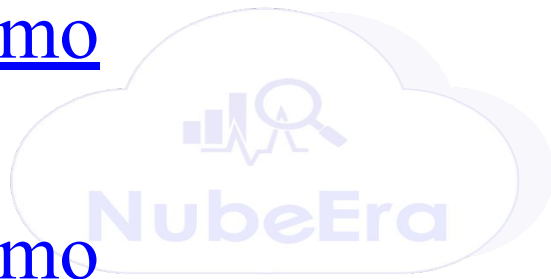
Summary



- Concept
- Collection
 - List
 - Tuples
 - Dictionary
 - Set

[demo](#)

[demo](#)



CORPORATE



RECORDED

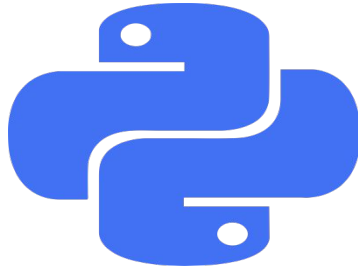


INTERACTIVE

???

The Important thing is not to
stop

Questioning



RECORDED



CORPORATE



INTERACTIVE



CORPORATE



RECORDED



INTERACTIVE