Marwadi
University

01CE1705-Programming with Python

Unit-3 List, Tuple and Dictionary

Prof. Chetan Chudasama
Computer Engineering Department

List

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary.

Create Python List:-

- In Python, a list is created by placing elements inside square brackets [], separated by commas.

```
my_list=[10,20,30]
```

- A list can have any number of items and they may be of different types (integer, float, string, etc.).

```
my_list=[10,"Hello",3.4,True]
```

- A list can also have another list as an item. This is called a nested list.

```
my_list = ["mouse", [8, 4, 6], ['a']]
my_list=[]#empty list
```

Access List elements:-

- There are various ways in which we can access the elements of a list.

List index:-

- We can use the index operator [] to access an item in a list.
- In Python, index start at 0. So, a list having 5 elements will have an index from 0 to 4.
- Trying to access indexes other than these will raise an IndexError.
- The index must be an integer. We can't use float or other types, this will result in TypeError.

•Nested lists are accessed using nested indexing.

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
print(my_list[0])#first element
print(my_list[2])#third element
```

```
list1=["rajkot",[2,0,1,5]]
print(list1[0][1])
print(list1[1][3])
print(my_list[4.0])#Error! Only integer can be used for indexing
```

Negative indexing:-

- The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
print(my_list[-1])#last element
print(my_list[-5])#fifth last elements
```

List slicing

- We can access a range of elements in a list by using the slicing operator :

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
print(my_list[2:5])
print(my_list[1:])
print(my_list[:4])
```

Note:-When We slice list, the start index is inclusive but end index is exclusive.

Add/Change elements:-

•Lists are mutable, meaning their elements can be changed.

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
my_list[0]=18#change the 1st element
my_list[2:4]=[33,55,7]#change 2nd,3rd and 4th elements
```

List Methods

append():-

•The append() methods adds an element to end of the list.

syntax:-list.append(element)

- The method takes single argument. This argument can be integer,string,list etc.

Example:adding element to a list

```
country_name=["India","Srilanka","England"]
country_name.append("Australia")
print(country_name)
['India','Srilanka','England','Australia’]
```

Example:adding list to a list

```
college_name=["Marwadi","VVP","Darshan","Atmiya"]
course_name=["CSE","Mechanical","Civil"]
college_name.append(course_name)
```

```
print(college_name)
['marwadi', 'VVP', 'darshan', 'atmiya', ['CSE', 'Mechanical', 'Civil’]]
```

- In the above code, a single item(course_name) is added to the college_name list.

extend():-

- The extend() method adds all the elements of an iterable(list,tuple,string) to the end of list.

Syntax:-list.extend(iterable)

Example:-

```
college_name=["marwadi","VVP","darshan","atmiya"]
course_name=["CSE","Mechanical","Civil"]#also use string and tuple instead of list
college_name.extend(course_name)
print(college_name)
['marwadi', 'VVP', 'darshan', 'atmiya', 'CSE', 'Mechanical', 'Civil']
```

- The extend() method modifies the original list.

insert():-

- The insert() method insert an element to the list at specified index.

syntax:-list.insert(index,element)

index - the index where the element needs to be inserted

element - this is the element to be inserted in the list

Example:

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
my_list.insert(3,"pant")
print(my_list)
['virat', 'rohit', 'rahul', 'pant', 'hardik', 'dhoni']
```

•The insert() method updates the current list.

Example: Insert tuple to the list

```
list1=[{1,2},[5,6,7]]
number_tuple=(3,4)
list1.insert(1,number_tuple)
print(list1)
[{1, 2}, (3, 4), [5, 6, 7]]
```

pop():-

- It removes the element at given index from the list and returns removed element.

Syntax:-list.pop(index)

Example:

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
print("Remove Element",my_list.pop(2))
print("Updated List",my_list)
```

Output:

```
Remove Element rahul
Updated List ['virat', 'rohit', 'hardik', 'dhoni’]
```

- The pop() method takes a single argument (index).
- The argument passed to the method is optional. If not passed, the default index -1 is passed as an argument (index of the last element).
- If the index passed to the method is not in range, it throws IndexError: pop index out of range exception.

remove():-

- The remove() method removes the first matching element from the list.

Syntax:list.remove(element)

Example:

```
my_list=["virat","rohit","rahul","hardik","dhoni"]
my_list.remove("rahul")
print("Updated List",my_list)
```

Output:

```
Updated List ['virat', 'rohit', 'hardik', 'dhoni’]
```

- It takes a single element as an argument and removes it from the list.
- If the element doesn't exist, it throws ValueError:list.remove(x): x not in list.
- It doesn't return any value (returns None).
- If a list contains duplicate elements, the remove() method only removes the first matching element.

del keyword:-

- It is used to delete lists, element within list,variables,dictionaries,tuple etc.

Example:delete variable,list

```
a=10
indianPlayer=["virat","rohit","rahul","hardik","dhoni","bhuvi","bumrah"]
del a
del indianPlayer[2]#remove rahul from the list
del indianPlayer[2:4]#remove rahul and hardik from the list
del indianPlayer#remove whole list
```

- We can't delete items of tuples and strings in Python. It's because tuples and strings are immutables.We can delete an entire tuple or string.

clear():-

- It removes all elements from the list.

Syntax:list.clear()

Example:

```
indianPlayer=["virat","rohit","rahul","hardik"]
indianPlayer.clear()
print(indianPlayer)
```

Output:

```
[]
```

- It doesn't take any parameters.
- It only empties the given list. It doesn't return any value.

- If we are using Python 2 or Python 3.2 and below, we cannot use the clear() method. we can use the del operator instead.

sort():-

- It can be used to sort a List in ascending, descending or user-defined order.

Syntax:-list.sort(reverse=True/False,key=myFunc)

reverse:-Optional. Reverse=True will sort the list in descending order. Default is reverse=False

key:-optional. A function to specify sorting criteria

Example:

```
numbers=[1,3,4,2]
numbers.sort()#number.sort(reverse=True) to sort list in descending order.
print(numbers)
```

Example:

```
indianPlayer=["Sachin","virat","rohit","rahul","hardik"]
indianPlayer.sort()
print(indianPlayer)
```

output:

```
['Sachin', 'hardik', 'rahul', 'rohit', 'virat’]
```

- The sort() method sorts the list in order from A-Z, to a-z in the alphabet.

Example:

```
list1=["cc","b","aaa","eeee","ddddddd"]
list1.sort(key=len)
print(list1)
```

Example:

```
list1=[[2,9],[1,10],[3,7]]
def sortBySecond(element):
    return element[1]
list1.sort(key=sortBySecond)
print(list1)
```

copy():-

•The copy() method returns a copy of the specified list.

Syntax:new_list=list.copy()

- The copy() method doesn't take any parameters.
- The copy() method returns a new list. It doesn't modify the original list.

Example:

```
spring_fruits = ['Apricot', 'Kiwi','Grapefruit','Cherry','Strawberry']
summer_fruits=spring_fruits.copy()
```

- Now we have a copy of our list, we can modify it to reflect our new fruit offering. We can do so using the append() and remove() methods.

When to use copy() method

Say that we operate a fruit stand and are looking to update our product list to reflect the new fruits in season for summer. To do so, we want to create a new list of products based on the old one. We want to save our old one for later use. This will allow us to keep track of what products we sold last season in an old list. We'll also be able to keep track of what we are selling this season in a new list. We can use the copy() method to duplicate a list and accomplish this task.

List copying using = operator

- We can also use the = operator to copy a list.

old_list = [1, 2, 3]

new_list = old_list

- Howerver, there is one problem with copying lists in this way. If you modify new_list, old_list is also modified.
- It is because the new list is referencing or pointing to the same old_list object.
- However, if you need the original list unchanged when the new list is modified, you can use the copy() method.

count():-

•It returns the number of times the specified element appears in the list.

Syntax:-list.count(element)

Example:

```
numbers=[18,3,7,45,55,18,3,12,18]
print(numbers.count(18))
print(numbers.count(77))
```

Output

```
3
0
```

Example:Count Tuple and List Elements Inside List

```
numbers=[["rahul","rohit"],"virat","hardik",(1983,2011),"pant",(1983,2011)]
print(numbers.count(['rahul','rohit']))
print(numbers.count((1983,2011)))
```

Output

1

2

index():-

- The index() method returns the index of the specified element in the list.

Syntax:list.index(element,start,end)

element: the element to be searched

start: start searching from this index

end: search element upto this index

Example:

```
indianPlayer=["virat","rohit","rahul","hardik","bumrah"]
print(indianPlayer.index("hardik"))
```

Output:

```
3
```

- If the element is not found, a ValueError exception is raised.
- The index() method only returns the first occurrence of the matching element.

reverse():-

•The reverse() method reverses the elements of the list.

Syntax:list.reverse()

Example:

```
numbers=[10,18,17,12,55,7,3]
numbers.reverse()
print(numbers)
```

Output:

```
[3, 7, 55, 12, 17, 18, 10]
```

- The reverse() method doesn't take any arguments.
- The reverse() method doesn't return any value. It updates the existing list.

Tuple

- Tuples are used to store multiple items in a single variable.
- A tuple is similar to a list. The difference between the two is that we can not change the elements of tuple once it is assigned whereas we can change the elements of a list.

Creating Tuple:

- A tuple is created by placing all the items (elements) inside parentheses (), separated by commas.
- A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

Example:-

```
a=()#empty tuple
a=(1,2,3)#tuple having integer
a=(1,"Hello",True,77.54)#tuple with mixed dataTypes
a=(["rohit","rahul"],"virat","hardik",["bhuvi","bumrah","shami"])#nested tuple
```

- The parentheses are optional.

•A tuple can also be created without using parentheses. This is known as tuple packing.
Example:

```
my_tuple=1,2,3,"rajkot"
a,b,c,d=my_tuple#tuple unpacking
print(a,b,c,d)
```

Note: What is tuple unpacking:-In Python, we are also allowed to extract the values back into variables. This is called “Unpacking”

- Creating a tuple with one element is a bit tricky.
- Having one element within parentheses is not enough. We will need a ending comma to indicate that it is a tuple.

Example:

```
a=("hello")
print(type(a))#<class 'str'>
```

```
a=("hello",)#creating a tuple having one element
print(type(a))#<class 'tuple'>
a="hello",#parenthesis is optional
print(type(a))#<class 'tuple'>
```

Access tuple elements:-

- There are various ways in which we can access the elements of a tuple.

1. Indexing

- We can use the index operator [] to access an item in a tuple, where the index starts from 0.
- So, a tuple having 6 elements will have index from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an IndexError.

- The index must be an integer, so we cannot use float or other types. This will result in TypeError.

•Nested tuples are accessed using nested indexing.
Example:

```
indianPlayer=("bhuvi","bumrah","chahal","hardik","pant",(1983,2007,2011))
print(indianPlayer[1]))
print(indianPlayer[8]))#tuple index of of range
print(indianPlayer[3.5]))#tuple index must be integers or slices, not float
print(indianPlayer[5][2]))#used to display nested tuple element
```

2.Negative Indexing:

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

Example:
```
indianPlayer=("bhuvi","bumrah","chahal","hardik")
print(indianPlayer[-2])
```

3.Slicing:

•We can access a range of elements in a tuple by using slicing operator :
Example:
```
indianPlayer=("bhuvi","bumrah","chahal","hardik","pant")
indianPlayer[1:4]
indianPlayer[:-3]#bhuvi,bumrah
indianPlayer[2:]
```

Changing a Tuple

- Tuples are immutable. This means that elements of a tuple cannot be changed once they have been assigned.
- But, if the element is itself a mutable data type like a list, its nested elements can be changed.

Example:

```
indianPlayer=("sachin","dhoni",[1983,2007,2011])
indianPlayer[1]="virat"#tuple does not support item assignment
indianPlayer[3][1]=2009
```

- We can use + operator to combine two tuples. This is called concatenation.
- We can also repeat the elements in a tuple for a given number of times using the * operator.

Example:

```
a=(1,2,3)
b=(4,5)
print(a+b)
print(b*3)
```

Deleting a Tuple

- We cannot change the elements in a tuple. It means that we cannot delete element from a tuple.
- We can delete a tuple by using del keyword.

Tuple Inbuilt Methods:-

- Methods that add items or remove items are not available with tuple.
- Only the following two methods are available.

count():-

- The count() method returns the number of times the specified element appears in the tuple.

Syntax:tuple.count(element)

Example:

```
subjectName=("java","python","os","dbms","network","dbms")
print(subjectName.count("php"))
print(subjectName.count("dbms"))
```

Example:count list and tuple inside Tuple

```
subjectName=("maths",["python","java"],("os","networking"))
print(subjectName.count(["python","java"]))
```

index():-

- The index() method returns the index of the specified element in the tuple.

Syntax:-tuple.index(element,start_index,end_index)

element: the element to scan

start_index:start scanning the element from start_index

end_index:stop scanning the element at end_index

- the index() scans the element in the tuple from start_index to end_index.
- It returns the index of given element in the tuple.
- It generate valueError exception if the element is not found in the tuple.

•The index() method only returns the first occurrence of the matching element.

Example:

```
subjectName=("web designing","java","python","dbms","os","java")
print(subjectName.index("java"))#index of first java element is returned
print(subjectName.index("english"))#thorws an error since 3 is absent in the tuple
print(subjectName.index("java",2))
```

Other Tuple Operation:-

1.Tuple Membership Test:

•We can check if an element exists in a tuple or not, using the keyword in.

Example:

```
subjectName=("web designing","java","python","dbms","os","java")
print("python" in subjectName)
print("computer network" in subjectName)
print("computer network" not in subjectName)
```

2.Iterating Through Tuple

- We can use a for loop to iterate through each item in a tuple.

Example:

```
subjectNames=("web designing","java","python","dbms","os","java")
for subject in subjectNames:
    print(subject,end=" ")
```

Advantages of Tuple over List

- Tuples are more memory efficient than lists.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.

Dictionary

- Dictionaries are used to store data in key:value pairs.
- It is an ordered collection of elements.

Creating Python Dictionary:-

- Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.
- The values can be of any data type and can repeat, keys must be of immutable type(number or string) and must be unique.

Example:

```
my_dict={}#empty dictionary
my_dict={"name":"vaibhav","college":"MU Rajkot","age":30}#dictionary with string keys
my_dict={1:"Ahmedabad",2:"Mehsana",3:"Rajkot"}#dictionary with integer keys
my_dict={"name":"virat",18:[10,20,30]}#dictionary with mixed keys
my_dict = dict({"name":"vaibhav","age":34})#using dict()
```

Accessing Elements from Dictionary

- While indexing is used with other data types to access values, a dictionary uses keys.
- Keys can be used either inside square brackets [] or with the get() method.
- If we use the square brackets [], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print(my_dict["name"])
print(my_dict.get("age"))
#Trying to access keys which does not exist throws error
print(my_dict["address"])#keyError
print(my_dict.get("addr ess"))#None
```

Changing and Adding Dictionary Elements

- Dictionaries are mutable. We can add new elements or change the value of existing elements using an assignment operator.
- If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
my_dict["name"]="harsh"#update value
my_dict["qualification"]="BE MTech"#add element
print(my_dict)
```

Removing Elements from dictionary:-

- We can remove a particular item in a dictionary by using the pop() method. This method removes an item with the provided key and returns the value.
- The popitem() method can be used to remove and return an arbitrary (key, value) item pair from the dictionary.
- All the items can be removed at once, using the clear() method.
- We can also use the del keyword to remove individual items or the entire dictionary itself.

Dictionary Methods

get():-

•It returns the value for the specified key if the key is in the dictionary.

Syntax:dict.get(key,value)

key:-key to be searched in the dictionary

value:-Value to be returned if the key is not found. The default value is None.

- get method returns:
  - the value for specified key if key is in the dictionary
  - None if the key not found and value is not specified
  - value if the key not found and value is specified

Example:

```
my_dict ={"name":"vaibhav","age":30}
print(my_dict.get("name"))
print(my_dict.get("salary"))#value not provided
print(my_dict.get("salary","Key not found"))#value provided
Output:
vaibhav
None
Key not found
```

fromkeys():-

•The fromkeys() method creates a dictionary from the given sequence of keys and values.

Syntax:-dict.fromkeys(keys,value)

Example:

```
alphabets=['a','b','c']
number=1
print(dict.fromkeys(alphabets,number))
```

```
{'a': 1, 'b': 1, 'c': 1}
```

- The same value is assigned to all the keys of the dictionary.

•If the value of the dictionary is not provided, None is assigned to the keys.

Example:

```
alphabets=['a','b','c']
print(dict.fromkeys(alphabets))
{'a': None, 'b': None, 'c': None}
```

clear():-

•It removes all elements from the dictionary.

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
my_dict.clear()
print(my_dict)
```

Output:

```
{}
```

- It doesn't take any parameters.
- It doesn't return any value.

popitem():-

•It removes and returns the last element (key, value) pair inserted into the dictionary.

Syntax:-dict.popitem()

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print(my_dict.popitem())
```

Output:

```
('city', 'rajkot')
```

- It doesn't take any parameters.
- The popitem() method generates a KeyError error if the dictionary is empty.

pop():-

- The pop() method removes the specified element from the dictionary.

•The value of the removed element is the return value of the pop() method.

Syntax:dict.pop(keyname,defaultValue)

keyname:The keyname of the element you want to remove
defaultvalue:A value to return if the specified key do not exist. If this parameter is not specified, and the no item with the specified key is found, an error is raised

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print(my_dict.pop("name"))
```

print(my_dict)

Output:

vaibhav

{'age': 30, 'city': 'rajkot’}

update():-

- Python update() method updates the dictionary with the key and value pairs.
- It inserts key/value if it is not present. It updates key/value if it is already present in the dictionary.

Syntax:-dict.update(key/value)

Example:

my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}

my_dict.update({'qualification':'BE'})

print(my_dict)

```
{'name': 'vaibhav', 'age': 30, 'city': 'rajkot', 'qualification': 'BE’}
```

- It doesn’t return any value but updates the Dictionary with new element.
- If update() is called without passing parameters, the dictionary remains unchanged.

Example: Update When Tuple is passed

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
my_dict.update([("EnrollNo",4521),("mobileNo",78745)])
print(my_dict)
```

- Here, we have passed a list of tuples [("EnrollNo",4521),("mobileNo",78745)] to the update() function.
- In this case, the first element of tuple is used as the key and the second element is used as the value.

keys():-

•It extracts the keys of the dictionary and returns the list of keys as a view object.

Syntax:dict.keys()

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print(my_dict.keys())
dict_keys(['name', 'age', 'city'])
```

- It doesn't take any parameters.

values():-

•It returns a view object that displays a list of all the values in the dictionary.

Syntax:dict.values()

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print(my_dict.values())
```

Output:

```
dict_values(['vaibhav', 30, 'rajkot’])
```

- It doesn't take any parameters.

items():-

- The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.
- The view object will reflect any changes done to the dictionary.

Syntax:dict.items()

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
x=my_dict.items()
my_dict['city']='pune'
print(x)
dict_items([('name', 'vaibhav'), ('age', 30), ('city', 'pune')])
```

Other Dictionary Operation

Dictionary Membership Test

- We can test if a key is in a dictionary or not using the keyword in. Notice that the membership test is only for the keys and not for the values.

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
print("name" in my_dict)
print("address" in my_dict)
```

Output:

```
True
False
```

Iterating Through a Dictionary

- We can iterate through each key in a dictionary using a for loop.

Example:

```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
for data in my_dict:
    print(data,end=" ")
```

Output:

```
name age city
```

Example:

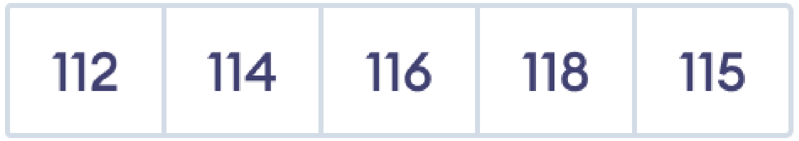
```
my_dict ={"name":"vaibhav","age":30,"city":"rajkot"}
for data in my_dict:
    print(data,":",my_dict[data])
```

Output:

```
name : vaibhav
age : 30
city : rajkot
```

Set

- A set is a collection of unique data. That is, elements of a set cannot be duplicate.
- For example, Suppose we want to store information about **student IDs**. Since **student IDs** cannot be duplicate, we can use a set.



Set of Student ID

- **Create a Set in Python**

```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)
```

Set

- # create a set of string type
```
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)
```
- # create a set of mixed data types
```
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```
- # create an empty set
```
empty_set = set()
```
- # check data type of empty_set
```
print('Data type of empty_set:', type(empty_set))
```

Set

- **Add Items to a Set in Python**

```
numbers = {21, 34, 54, 12}
print('Initial Set:',numbers)

# using add() method
numbers.add(32)
print('Updated Set:', numbers)
```

- **Remove an Element from a Set**

```
languages = {'Swift', 'Java', 'Python'}
print('Initial Set:',languages)
# remove 'Java' from a set
removedValue = languages.discard('Java')
print('Set after remove():', languages)
```