MODULE III PYTHON DATA STRUCTURE 9

List : Finding and Updating an Item – Nested Lists – Cloning Lists – Looping Through a List – Sorting a List – List Concatenation – List Slices – List Methods – Mutability.

Tuples: Creation, Accessing, Updating, Deleting Elements in a Tuple, Tuple Assignment, Tuple as Return Value, Nested Tuples, Basic Tuple Operations – Sets.

Dictionary: Creating, Accessing, Adding Items, Modifying, Deleting, Sorting, Looping, Nested Dictionaries Built-in Dictionary Function – Finding Key and Value in a Dictionary.

Lists-Introduction

- A list is a collection of values.
- It may contain different types of values.
- To define a list, you must put values separated with commas in square brackets.

```
>>> days=['Monday','Tuesday',3,4,5,6,7]
>>> print(days)
['Monday', 'Tuesday', 3, 4, 5, 6, 7]
```

```
>>> stud=["abdul","Basa",'Chandru',"Divya"]
>>> print(stud)
['abdul', 'Basa', 'Chandru', 'Divya']
```

Allow Duplicates

- lists are indexed, lists can have items with the same value

```
>>> stud=["abdul","Basa",'Chandru',"Divya",'Divya']
>>> stud
['abdul', 'Basa', 'Chandru', 'Divya', 'Divya']
```

List Length

- To determine how many items a list has

```
>>> stud=["abdul","Basa",'Chandru',"Divya",'Divya']
>>> print(len(stud))
5
```

List Items

```
>>> stud1=["abi","balu",'cibi',"divya"]
>>> rrn=[2001,2002,2003,2004]
>>> dept=['cse','cse','cse','cse']
>>> print(stud1,rrn,dept)
['abi', 'balu', 'cibi', 'divya'] [2001, 2002, 2003, 2004] ['cse', 'cse', 'cse', 'cse']

>>> CSE=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> print(CSE)
[2002, 'abdul', 'First Year', 'Dept of CSE', 19]
```

type()

- lists are defined as objects with the data type 'list'

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> type(cse)
<class 'list'>
```

The list() Constructor

- constructor when creating a new list

```
>>> stud=list(("hi"))
>>> print(stud)
['h', 'i']
```

```
>>> stud=list((2001,"abdul"))
>>> print(stud)
[2001, 'abdul']
```

Accessing list & Operations

- List items are indexed
- by referring to the index number

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> print(cse[1])
abdul
```

Negative Indexing

- It starts from the end

```
>>> print(cse[-1])
19
```

Slicing a List

- slice a string- with the slicing operator.
- *cse=[2002,'abdul',"First Year",'Dept of CSE',19]*

>>> print(cse[0:4])

[2002, 'abdul', 'First Year', 'Dept of CSE']

>>> print(cse[0:])
[2002, 'abdul', 'First Year', 'Dept of CSE', 19]

>>> print(cse[0:-1])
[2002, 'abdul', 'First Year', 'Dept of CSE']

>>> print(cse[:-1])
[2002, 'abdul', 'First Year', 'Dept of CSE']

>>> print(cse[3:-1])
['Dept of CSE']

>>> print(cse[-3:-1])
['First Year', 'Dept of CSE']

Reassigning Elements of a List

- A list is mutable. This means that you can reassign elements

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse[1]="Syed"
>>> print(cse)
[2002, 'Syed', 'First Year', 'Dept of CSE', 19]

>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse[0:2]=[2001,"Abdul"]
>>> print(cse)
[2001, 'Abdul', 'First Year', 'Dept of CSE', 19]
```

Add List Items

Insert Items

-inserts an item at the specified index

```
>>> cse.insert(0,"Crescent")
>>> print(cse)
['Crescent', 2001, 'Abdul', 'First Year', 'Dept of CSE', 19]
```

Append Items

- To add an item to the end of the list

```
>>> cse.append("Chennai")
>>> print(cse)
['Crescent', 2001, 'Abdul', 'First Year', 'Dept of CSE', 19, 'Chennai']
```

Extend List

- append elements from *another list* to the current list, use the extend() method

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> IoT=["vandalur"]
>>> cse.extend(IoT)
>>> print(cse)
[2002, 'abdul', 'First Year', 'Dept of CSE', 19, 'vandalur']
```

Add Any Iterable

The extend() method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> csetup=("chennai")
>>> cse.extend(csetup)
>>> print(cse)
[2002, 'abdul', 'First Year', 'Dept of CSE', 19, 'c', 'h', 'e', 'n', 'n', 'a', 'i']
```

Remove List Items

The remove() method removes the specified item.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse.remove(19)
>>> print(cse)
[2002, 'abdul', 'First Year', 'Dept of CSE']
```

Remove Specified Index

The pop() method removes the specified index.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse.pop(0)
2002
>>> print(cse)
['abdul', 'First Year', 'Dept of CSE', 19]
```

the pop() method removes the last item.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse.pop()
19
>>> print(cse)
[2002, 'abdul', 'First Year', 'Dept of CSE']
```

The del keyword also removes the specified index

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> del cse[0]
>>> print(cse)
['abdul', 'First Year', 'Dept of CSE', 19]
```

The del keyword can also delete the list completely.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> del cse
>>> print(cse)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name ‘cse' is not defined
```

Clear the List

The clear() method empties the list.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> cse.clear()
>>> print(cse)
[ ]
```

Loop Lists

for loop:

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> for x in cse:
...     print(cse)
...
[2002, 'abdul', 'First Year', 'Dept of CSE', 19]
[2002, 'abdul', 'First Year', 'Dept of CSE ', 19]
[2002, 'abdul', 'First Year', 'Dept of CSE ', 19]
[2002, 'abdul', 'First Year', 'Dept of CSE ', 19]
[2002, 'abdul', 'First Year', 'Dept of CSE ', 19]
```

Loop Through the Index Numbers

- loop through the list items by referring to their index number.
- Use the range() and len() functions to create a suitable iterable.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> for i in range(len(cse)):
...     print(cse[i])
...
2002
abdul
First Year
Dept of CSE
19
```

While Loop

- len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> i=0
>>> while i < len(cse):
...     print(cse[i])
...     i=i+1
...
2002
abdul
First Year
Dept of CSE
19
```

Looping Using List Comprehension

- List Comprehension offers the shortest syntax for looping through lists

```
>>> cse=[2002,'abdul',"First Year",'Dept of CSE',19]
>>> [print(x) for x in cse]
2002
abdul
First Year
Dept of CSE
19
[None, None, None, None, None]
```

Working with lists

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
>>> cse=['abdul',"First Year",'Dept of CSE']
>>> newcse=[]
>>> for x in cse:
...     if "a" in x:
...             newcse.append(x)
...     print(newcse)
...
['abdul']
['abdul', 'First Year']
['abdul', 'First Year']
```

Condition

- The *condition* is like a filter that only accepts the items that valuate to True

```
>>> stud=[2001,"abdul",“CSE"]
>>> newlist=[x for x in stud if x!="abdul"]
>>> print(newlist)
[2001, ‘CSE']
```

Iterable

- The *iterable* can be any iterable object, like a list, tuple, set etc.
  - range() function to create an iterable

```
>>> newlist=[x for x in range(10)]
>>> print(newlist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Accept only numbers lower than 5

```
>>> newlist=[x for x in range(10) if x < 5]
>>> print(newlist)
[0, 1, 2, 3, 4]
```

Expression

- The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

```
>>> stud=["abdul",“CSE"]
>>> newlist=[x.upper() for x in stud]
>>> print(newlist)
['ABDUL', ‘CSE']
```

- Set all values in the new list

```
>>> newlist=["fathima" for x in stud]
>>> print(newlist)
['fathima', 'fathima']
```

- Replace the values in new list

```
>>> stud=["abdul",“CSE"]
>>> newlist=[x if x!="abdul" else "kavin" for x in stud]
>>> print(newlist)
['kavin', ‘CSE']
```

Sort Lists

Sort List Alphanumerically

- List objects have a sort() method that will sort the list alphanumerically, ascending, by default

```
>>> stud=["abdul","banu","cibi","david","kavin","janu"]
>>> stud.sort()
>>> print(stud)
['abdul', 'banu', 'cibi', 'david', 'janu', 'kavin']
```

```
>>> rrn=[2001,2003,2005,2002,2004]
>>> rrn.sort()
>>> print(rrn)
[2001, 2002, 2003, 2004, 2005]
```

Sort Descending

- To sort descending, use the keyword argument reverse = True

```
>>> stud=["abdul","banu","cibi","david","kavin","janu"]
>>> stud.sort(reverse=True)
>>> print(stud)
['kavin', 'janu', 'david', 'cibi', 'banu', 'abdul']


>>> rrn=[2001,2003,2005,2002,2004]
>>> rrn.sort(reverse=True)
>>> print(rrn)
[2005, 2004, 2003, 2002, 2001]
```

Copy a List

- to make a copy, one way is to use the built-in List method copy()

```
>>> stud=["abdul","banu","cibi","david","kavin","janu"]
>>> mystud=stud.copy()
>>> print(mystud)
['abdul', 'banu', 'cibi', 'david', 'kavin', 'janu']
```

- Make a copy of a list with the list() method

```
>>> stud=["abdul","banu","cibi","david","kavin","janu"]
>>> mystud=list(stud)
>>> print(mystud)
['abdul', 'banu', 'cibi', 'david', 'kavin', 'janu']
```

Join Lists

Join Two Lists

- using the + operator

```
>>> IoT=["abdul","rahman"]
>>> CSE=["fathima","divya"]
>>> dept=IoT+CSE
>>> print(dept)
['abdul', 'rahman', 'fathima', 'divya']
```

- to join two lists is by appending all the items from list2 into list1

```
>>> IoT=["abdul","rahman"]
>>> CSE=["fathima","divya"]
>>> for x in CSE:
...     IoT.append(x)
...     print(IoT)
...
['abdul', 'rahman', 'fathima']
['abdul', 'rahman', 'fathima', 'divya']
```

- the extend() method, which purpose is to add elements from one list to another list

```
>>> IoT=["abdul","rahman"]
>>> CSE=["fathima","divya"]
>>> IoT.extend(CSE)
>>> print(IoT)
['abdul', 'rahman', 'fathima', 'divya']
```

Iterating on the List

```
>>> num=[1,2,3,4,5]
>>> for n in num:
...     print(n)
...
1
2
3
4
5
```

Multidimensional Lists

A list may have more than one dimension.

```
num=[[1,2,3],[4,5,6]]
>>> print(num)
[[1, 2, 3], [4, 5, 6]]
```

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

- The append() method appends an element to the end of the list.

```
>>>IoT=["abdul"]
>>>cse=["rahman"]
IoT.append(cse)
>>>print(IoT)
['abdul', ['rahman']]
```

The clear() method removes all the elements from a list

```
>>> stud.clear()
>>> print(stud)
[]
```

- The copy() method returns a copy of the specified list

```
>>> CSE=[‘Fathima’, ‘divya’]
>>> x=CSE.copy()
>>> print(x)
['fathima', 'divya']
```

- The count() method returns the number of elements with the specified value.

```
>>> CSE=[ ‘Fathima’, ‘divya’]
>>> x=CSE.count("fathima")
>>> print(x)
1
```

- The extend() method adds the specified list elements (or any iterable) to the end of the current list.

```
>>> IoT=[‘abdul’, ‘rahman’]
>>> CSE=[‘Fathima’, ‘divya’]
>>> IoT.extend(CSE)
>>> print(IoT)
['abdul', 'rahman', 'fathima', 'divya']
```

- The index() method returns the position at the first occurrence of the specified value.

```
>>> cse=[‘abdul’, ‘rahman’]
>>> x=cse.index(‘rahman’)
>>> print(x)
1
```

- The insert() method inserts the specified value at the specified position

```
>>> CSE=[‘abdul’, ‘rahman’]
>>> CSE.insert(1,2001)
>>> print(CSE)
['abdul', 2001, 'rahman']
```

- The pop() method removes the element at the specified position

```
>>>CSE=[‘abdul’,2001, ‘rahman’, 2002]
>>> x=CSE.pop(3)
>>> print(x)
2002
>>> print(CSE)
['abdul', 2001, 'rahman']
```

- The remove() method removes the first occurrence of the element with the specified value

```
>>>CSE=[‘abdul’,2001, ‘rahman’, 2002]
>>>CSE.remove(2001)
>>> print(CSE)
['abdul', 'rahman’,2002]
```

- The reverse() method reverses the sorting order of the elements

```
>>>CSE=[‘abdul’,2001, ‘rahman’, 2002]
>>> CSE.reverse()
>>> print(CSE)
[2002, 'rahman', 2001, 'abdul']
```

The sort() method sorts the list ascending by default.

```
>>> CSE=[‘abdul’, ‘rahman’]
>>> CSE.sort()
>>> print(CSE)
['abdul', 'rahman']
```

- Sort the list descending

```
>>> CSE=[‘abdul’, ‘rahman’]
>>> CSE.sort(reverse=True)
>>> print(CSE)
['rahman', 'abdul']
```

Tuple

- A tuple is like a list.
- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and **unchangeable.**
- Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc …

```
>>> name=('Ahmed','Basa','Cibi')
>>> print(name)
('Ahmed', 'Basa', 'Cibi')
```

Allow Duplicates

```
>>> dept=('cse','aids','cse','iot','cs')
>>> dept
('cse', 'aids', 'cse', 'iot', 'cs')
```

Access Tuple Items

- access tuple items by referring to the index number

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept[1])
aids
```

Negative Indexing

- Negative indexing means start from the end.

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept[-1])
cs
```

Range of Indexes

- It specify a range of indexes by specifying where to start and where to end the range

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept[1:3])
('aids', 'cse')
```

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept[:3])
('cse', 'aids', 'cse')
```

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept[1:])
('aids', 'cse', 'iot', 'cs')
```

Update Tuples

Change Tuple Values

- Once a tuple is created, it cannot change its values.
- Tuples are **unchangeable**, or **immutable** as it also is called.
- But there is a workaround. we can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
>>> dept=('cse','aids','cse','iot','cs')
>>> x=(list(dept))
>>> x[0]="ece"
>>> dept=tuple(x)
>>> print(dept)
('ece', 'aids', 'cse', 'iot', 'cs')
```

Add Items

- Once a tuple is created, we cannot add items to it.
- We can convert it into a list, add your item(s), and convert it back into a tuple.

```
>>> dept=('cse','aids','cse','iot','cs')
>>> x=list(dept)
>>> x.append("it")
>>> dept=tuple(x)
>>> print(dept)
('cse', 'aids', 'cse', 'iot', 'cs', 'it')
```

Remove Items

```
>>> dept=('cse','aids','cse','iot','cs')
>>> x=list(dept)
>>> x.remove('cse')
>>> dept=tuple(x)
>>> print(dept)
('aids', 'cse', 'iot', 'cs')
```

- The del keyword can delete the tuple completely:

```
>>> dept=('cse','aids','cse','iot','cs')
>>> del dept
```

Unpack Tuples

Packing a Tuple

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple

```
>>> dept=('cse','aids','cse','iot','cs')
>>> print(dept)
('cse', 'aids', 'cse', 'iot', 'cs')
```

Unpacking a Tuple

- Its allowed to extract the values back into variables. This is called "unpacking"

```
>>> dept=('cse','aids',’mech','iot','cs')
>>> (ece,eee,aero,auto,bio)=dept
>>> print(ece)
cse
>>> print(eee)
aids
>>> print(aero)
mech
>>> print(auto)
iot
>>> print(bio)
cs
>>> print(dept)
('cse', 'aids', ‘mech', 'iot', 'cs')
```

Loop Tuples

Loop Through a Tuple

- loop through the tuple items by using a for loop.

```
>>> dept=('cse','aids','cse','iot','cs')
>>> for x in dept:
...     print(dept)
...
('cse', 'aids', 'cse', 'iot', 'cs')
('cse', 'aids', 'cse', 'iot', 'cs')
('cse', 'aids', 'cse', 'iot', 'cs')
('cse', 'aids', 'cse', 'iot', 'cs')
('cse', 'aids', 'cse', 'iot', 'cs')
```

Loop Through the Index Numbers

```
>>> dept=('cse','aids',’ece','iot','cs')
>>> for i in range(len(dept)):
...     print(dept[i])
...
cse
aids
ece
iot
cs
```

Join Tuples

Join Two Tuples

- To join two or more tuples we can use the + operator:

```
>>> dept=('cse','aids',’mech','iot','cs')
>>> dept1=('eee','ece','auto','aero','bio')
>>> crescent=dept+dept1
>>> print(crescent)
('cse', 'aids', ‘mech', 'iot', 'cs', 'eee', 'ece', 'auto', 'aero', 'bio')
```

Multiply Tuples

- To multiply the content of a tuple a given number of times, we can use the * operator:

```
>>> dept=('cse','aids',’mech','iot','cs')
>>> crescent=dept*2
>>> print(crescent)
('cse', 'aids', ‘mech', 'iot', 'cs', 'cse', 'aids', ‘mech', 'iot', 'cs')
```

Tuple Methods

- Python has two built-in methods

1. count()
2. index()

count()

```
>>> dept=('cse','aids','cse','iot','cs')
>>> x=dept.count("cse")
>>> print(x)
2
```

index()

```
>>> dept=('cse','aids','ece','iot','cs')
>>> x=dept.index('cse')
>>> print(x)
0
```

Python Dictionaries

- A dictionary holds key-value pairs.
- Declare it in curly braces, with pairs separated by commas.
- Separate keys and values by a colon(:)
- A dictionary is a changeable and does not allow duplicates.

```
>>> student={'Abdul’:’CSE', 'year':1}
>>> student
{'Abdul’: ‘CSE', 'year': 1}
>>> type(student)
<class 'dict'>
```

Access Dictionary Items

Accessing Items

- by referring to its key name, inside square brackets

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> x=book['year']
>>> print(x)
1
```

get()

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> x=book.get("Python")
>>> print(x)
Programming
```

Get Keys

- The keys() method will return a list of all the keys in the dictionary.

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> x=book.keys()
>>> print(x)
dict_keys(['Python', 'dept', 'year'])
```

Change Dictionary Items

Change Values

- To change the value of a specific item by referring to its key name

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book["year"]=2
>>> print(book)
{'Python': 'Programming', 'dept': ‘CSE', 'year': 2}
```

Add Dictionary Items

Adding Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book["university"]="crescent"
>>> print(book)
{'Python': 'Programming', 'dept': ‘CSE', 'year': 1, 'university': 'crescent'}
```

Update Dictionary

- The update() method will update the dictionary with the items from a given argument.
- If the item does not exist, the item will be added.

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book.update({"university":"crescent"})
>>> print(book)
{'Python': 'Programming', 'dept': ‘CSE', 'year': 1, 'university': 'crescent'}
```

```
>>> book={"Python":"Programming","dept":"AIDS","year":1}
>>> book.update({"dept":'cse'})
>>> print(book)
{'Python': 'Programming', 'dept': 'cse', 'year': 1}
```

Remove Dictionary Items

Removing Items

- The pop() method removes the item with the specified key name

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book.pop("Python")
'Programming'
>>> print(book)
{'dept': ‘CSE', 'year': 1}
```

- The popitem() method removes the last inserted item

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book.popitem()
('year', 1)
>>> print(book)
{'Python': 'Programming', 'dept': ‘CSE'}
```

- The del keyword removes the item with the specified key name

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> del book['year']
>>> print(book)
{'Python': 'Programming', 'dept': ‘CSE'}
```

- The del keyword can also delete the dictionary completely

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> del book
>>> print(book)
```

- The clear() method empties the dictionary

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> book.clear()
>>> print(book)
{}
```

Loop Dictionaries

Loop Through a Dictionary

- loop through a dictionary by using a for loop.

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> for x in book:
...     print(book)
...
{'Python': 'Programming', 'dept': ' CSE ', 'year': 1}
{'Python': 'Programming', 'dept': ' CSE ', 'year': 1}
{'Python': 'Programming', 'dept': ' CSE ', 'year': 1}
```

- Print all *values* in the dictionary, one by one

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> for x in book:
...     print(book[x])
...
Programming
CSE
1
```

- the values() method to return values of a dictionary

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> for x in book.values():
...     print(x)
Programming
CSE
1
```

- the keys() method to return the keys of a dictionary

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> for x in book.keys():
...     print(x)
Python
dept
year
```

- Loop through both *keys* and *values*, by using the items() method

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> for x,y in book.items():
...     print(x,y)
...
Python Programming
dept CSE
year 1
```

Copy Dictionaries

Copy a Dictionary

- the built-in Dictionary method copy()

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> x=book.copy()
>>> print(x)
{'Python': 'Programming', 'dept': ‘CSE', 'year': 1}
```

- Make a copy of a dictionary with the dict() function

```
>>> book={"Python":"Programming","dept":“CSE","year":1}
>>> x=dict(book)
>>> print(x)
{'Python': 'Programming', 'dept': ‘CSE', 'year': 1}
```

Nested Dictionaries

- A dictionary can contain dictionaries, this is called nested dictionaries

```
>>> blibrary={"book":{"Python":"Programming","dept":“CSE","year":1},
                    "book1":{"c":"Program","dept":"it"}}
>>> print(blibrary)
{'book': {'Python': 'Programming', 'dept': ‘CSE', 'year': 1}, 'book1': {'c': 'Program', 'dept': 'it'}}
```

- Create three dictionaries, then create one dictionary that will contain the other three dictionaries

```
>>> book1={"name":"python","year":1}
>>> book2={"name":"CP","year":1}
>>> blibrary={"book1":book1,"book2":book2}
>>> print(blibrary)
{'book1': {'name': 'python', 'year': 1}, 'book2': {'name': 'CP', 'year': 1}}
```

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Python Sets

Python Sets

- A set can have a list of values. Define it using curly braces.

```
>>> num={1,2,3}
>>> num
{1, 2, 3}
```

It returns only one instance of any value present more than once.

```
>>> num={1,2,3,3,4}
>>> num
{1, 2, 3, 4}
```

Unordered
Duplicates Not Allowed

Python Sets

- It is mutable. You can change its elements or add more.
- Use the add() and remove() methods to do so.

```
>>> num.remove(4)
>>> num
{1, 2, 3}

>>> num.add(4)
>>> num
{1, 2, 3, 4}
```

Length of a Set

- To determine how many items a set has, use the len() function.

```
dept={“cse”, “aero”, “mech”, “ece”, “eee”, “bio”}
 print(len(dept))
 6
```

type()

```
 type(dept)
<class 'set’>
```

set() Constructor

```
 dept=set(("cse","aero","ece","eee","mech","bio"))
 print(dept)
{'mech', 'aero', 'cse', 'bio', 'eee', 'ece'}
```

Update

```
dept={"cse","IoT","AI&DS"}
dept.update(["cs"])
print(dept)
{'IoT', 'cs', 'AI&DS', 'cse'}
```

Copy()

```
dept={"cse","Iot","CS","AI&DS"}
dep=dept.copy()
print(dep)
{"cse","Iot","CS","AI&DS"}
```

Discard

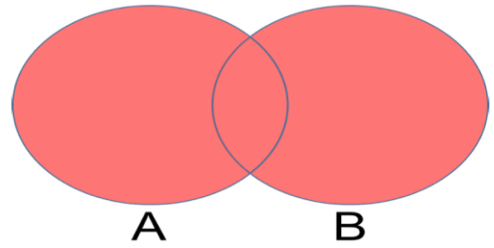
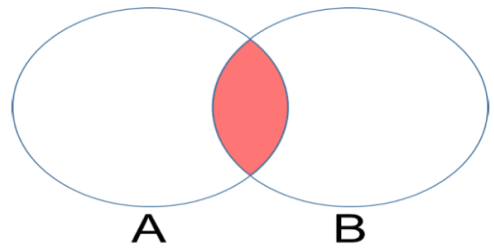
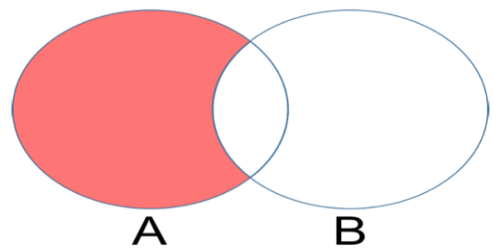
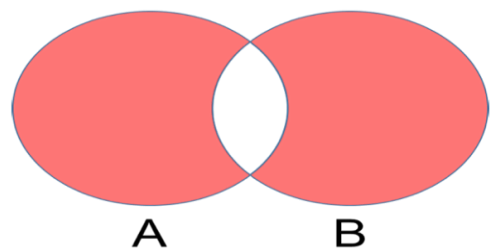
```
dept={"cse","IoT","AI&DS“, “cs”}
dept.discard("cs")
print(dept)
{'IoT', 'AI&DS', 'cse'}
```

Clear()

```
dept.clear()
```

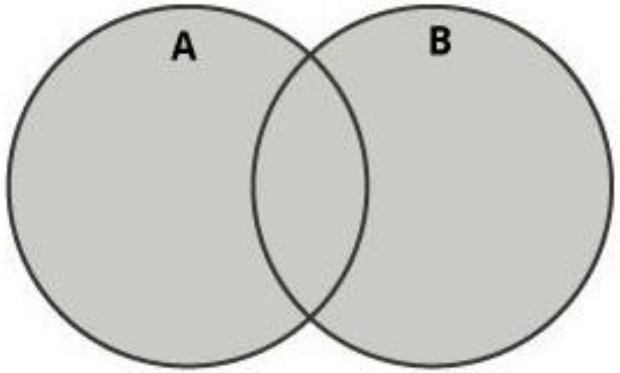
PoP()

```
dept={"cse","IoT","AI&DS“}
dept.update(["cs"])
dept.pop()
print(dept)
{'IoT', 'AI&DS', 'cse'}
```

Set Operation	Venn Diagram	Interpretation
Union		$A \cup B$, is the set of all values that are a member of *A*, or *B*, or both.
Intersection		$A \cap B$, is the set of all values that are members of both *A* and *B*.
Difference		$A \setminus B$, is the set of all values of *A* that are not members of *B*
Symmetric Difference		$A \triangle B$, is the set of all values which are in one of the sets, but not both.

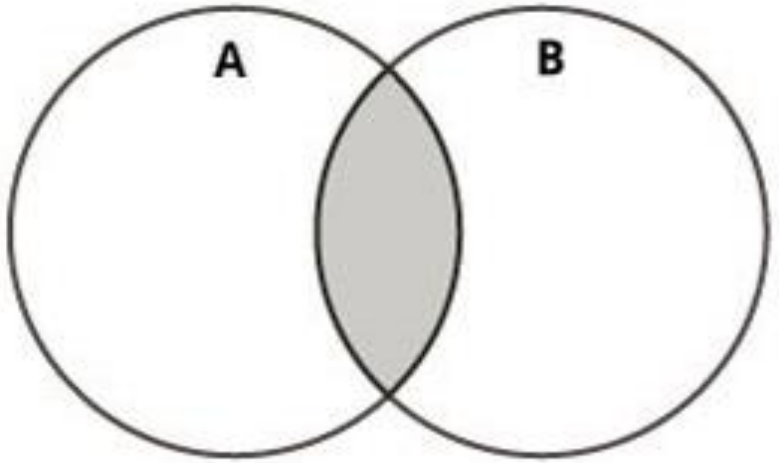
union()

```
dept={"cse"}
year={2}
set=dept.union(year)
print(set)
{2, 'cse'}
```



intersection()

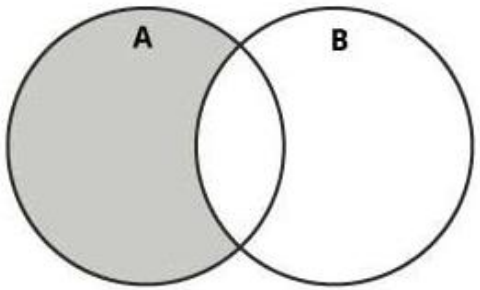
```
crescent={"cse","aero","mech"}
srm={"cse","aero","mech","bio"}
univ=crescent.intersection(srm)
print(univ)
{'mech', 'aero', 'cse'}
```



Difference()

```
dept={"cse","Iot","CS","AI&DS"}
dept1={"mech","areo","cse","bio"}
crescent=dept-dept1
print(crescent)
```

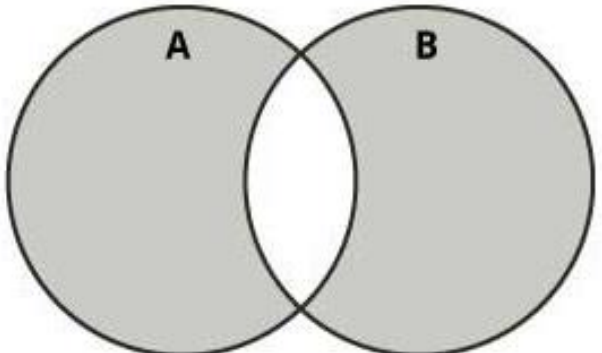
{'CS', 'AI&DS', 'Iot'}



Symmetric Difference()

```
dept={"cse","Iot","CS","AI&DS"}
dept1={"mech","aero","cse","bio"}
print(dept^dept1)
```

{‘aero', 'AI&DS', 'bio', 'Iot', 'CS', 'mech'}



```
dept={"cse","Iot","CS","AI&DS"}
dept1={"mech","areo","cse","bio"}
dept<dept1
False
```

```
dept={"cse","Iot","CS","AI&DS"}
dept1={"mech","areo","cse","bio"}
dept<=dept1
False
```

```
dept={"cse","Iot","CS","AI&DS"}

dept1={"mech","areo","cse","bio"}

dept>dept1

False
```

```
dept={"cse","Iot","CS","AI&DS"}

dept1={"mech","areo","cse","bio"}

dept>=dept1

False
```

```
dept={"cse","Iot","CS","AI&DS"}

dept1={"mech","areo","cse","bio"}

dept==dept1

False
```

```
dept={"cse","Iot","CS","AI&DS"}

dept1={"mech","areo","cse","bio"}

dept!=dept1

True
```

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns `True` if two sets have a null intersection
issubset()	Returns `True` if another set contains this set
issuperset()	Returns `True` if this set contains another set

pop()	Removes and returns an arbitrary set element. Raises `KeyError` if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a `KeyError`
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others


Thank
you!