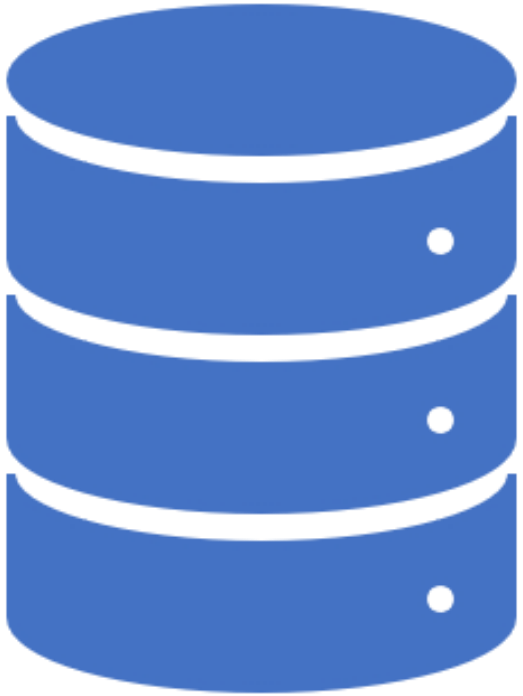


Data Structure with Python

Mr. V. M .Vasava

GPG,IT Dept.

Agenda

Introduction Array

Basic Operation of Array

Array Implementation with python

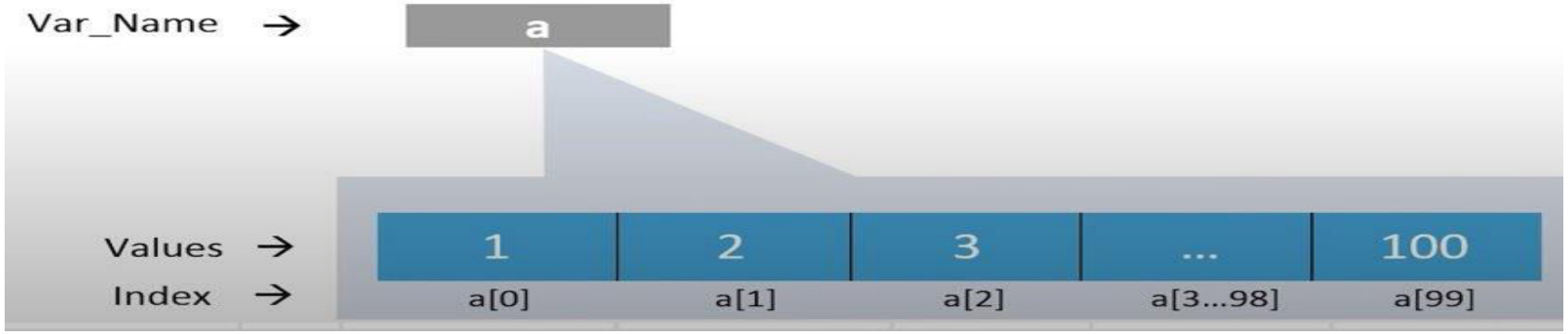
Example

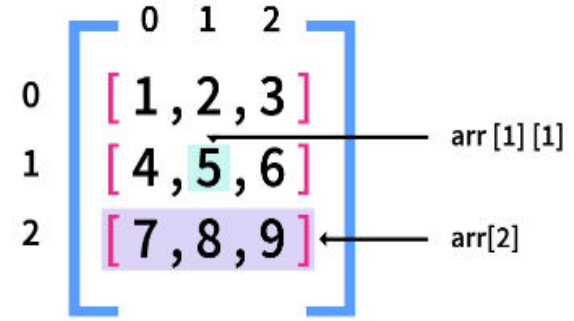
Array

Array is a container that holds multiple values of same type.

It is used to store collections of data

Arrays are handled by a Python object-type module **array**.
Basic structure of array:

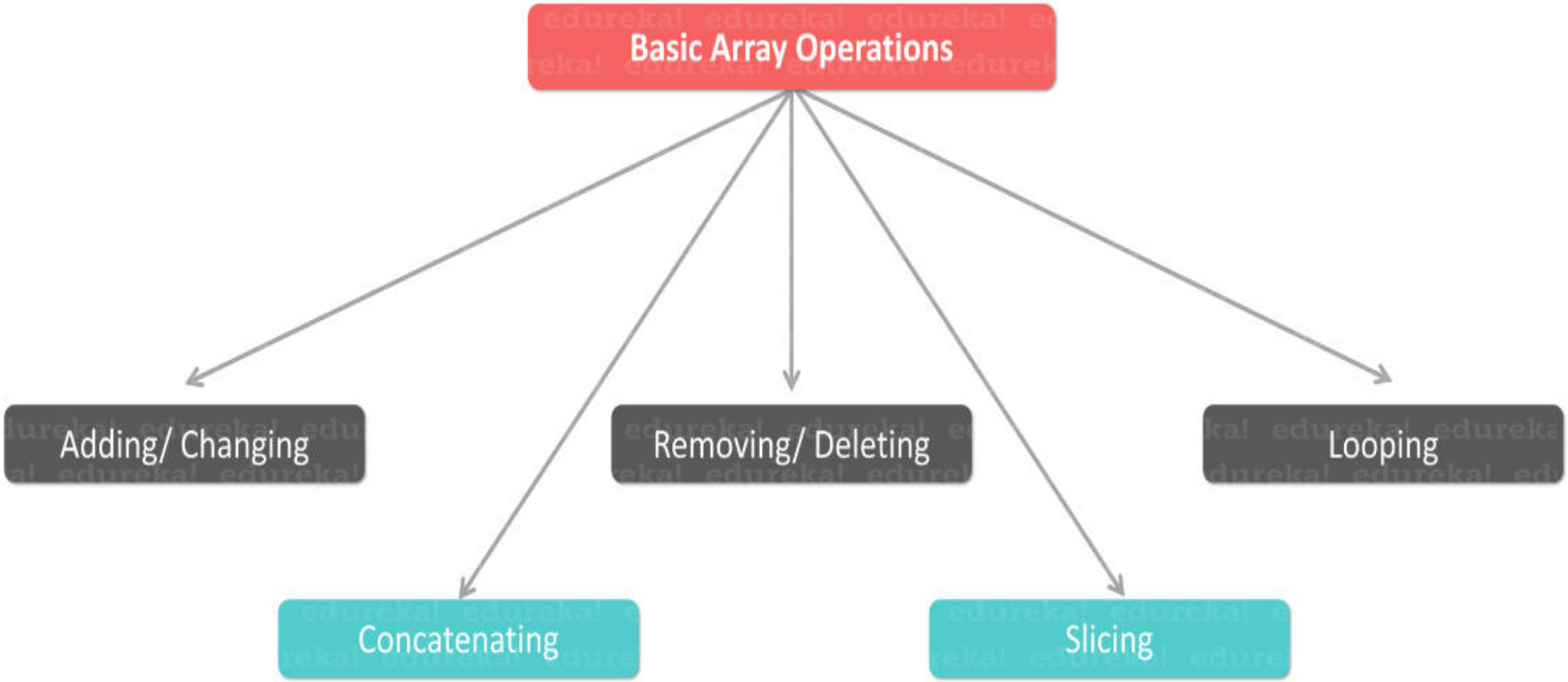




2D Array

We can create a 2D array in python as below:
array_name=[n_rows][n_columns]

Basic operations of Array



Implementation array with python

Implementation array

- Array module
- Numpy array module

Operation of Array

- 1.Traverse – print all the array elements one by one.
- 2.Insertion – Adds an element at the given index.
- 3. Deletion – Deletes an element at the given index.
- 4. Search – Searches an element using the given index or by value.
- Update – Updates an element at the given index.

Creation of array

- Array in python can be created after importing the array module.

Syntax 1D Array :

arrayName=array(typecode, [initializers])

Typecode are the codes that are used to define the type of value the array will hold.

Creating an 1D & 2D array with python

```
In [3]: import array

In [4]: a=array.array('i',[1,2,3,4,5])

In [5]: a
Out[5]: array('i', [1, 2, 3, 4, 5])

In [6]: from array import *

In [7]: arr=array('i',[10,20,30,40,50])

In [8]: arr
Out[8]: array('i', [10, 20, 30, 40, 50])
```

2D Array

```
In [80]: arr=[[1,2,3],[4,5,6],[7,8,9]]
    ...:
    ...: print("Element at [1][0]=",arr[1][0])
Element at [1][0]= 4

In [81]:
```

Type Code	Python Type	Min.size in Bytes
‘c’	character	1
‘i’	int	2
‘l’	long	2
‘f’	Float	4
‘d’	Double	8

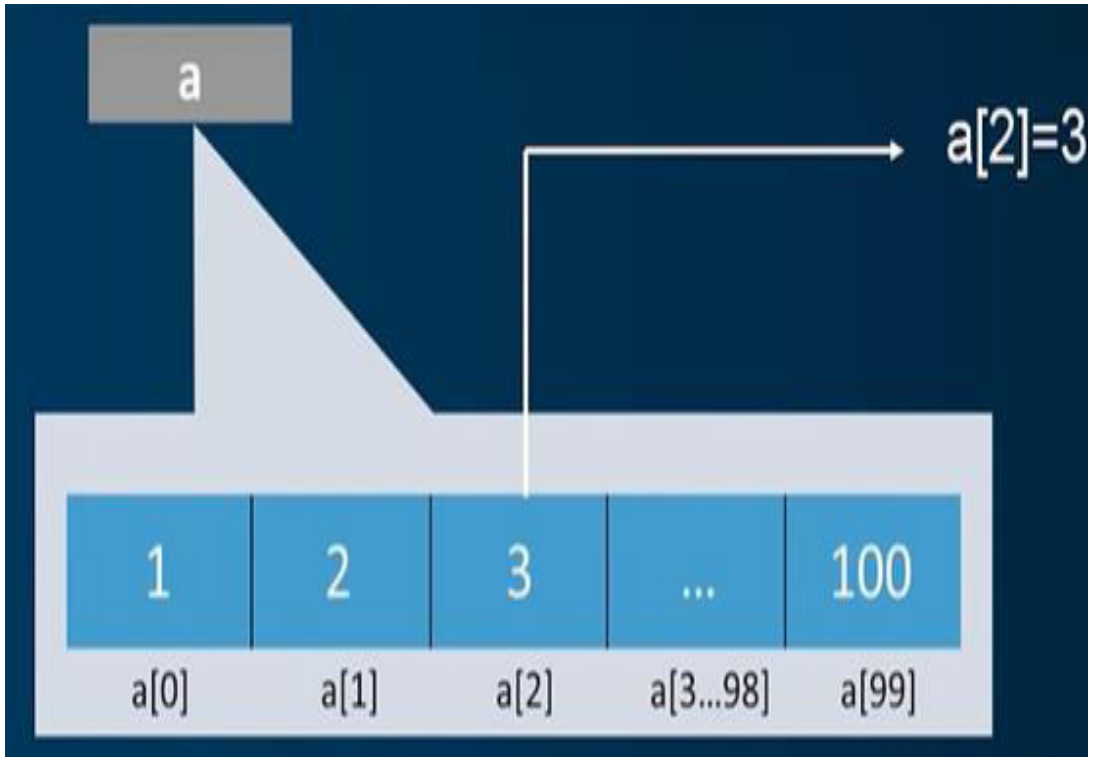
Example

- **Example:** Get all array's supported type codes and type code used to define an array.

```
In [16]: import array

In [17]: array.typecodes
Out[17]: 'bBuhHiIlLqQfd'

In [18]: a = array.array('i',[8,9,3,4])

In [19]: b = array.array('d', [2.3,3.5,6.2])

In [20]: a.typecode
Out[20]: 'i'

In [21]: b.typecode
Out[21]: 'd'
```

Accessing Array element 1D

- Accessing elements using index values using [] Symbols.
- Indexing starts from 0 not 1. Negative index values can be used as well.




```
In [4]: a=array('i',[10,20,30,40,50])

In [5]: a
Out[5]: array('i', [10, 20, 30, 40, 50])

In [6]: a[0]
Out[6]: 10

In [7]: a[-4]
Out[7]: 20
```

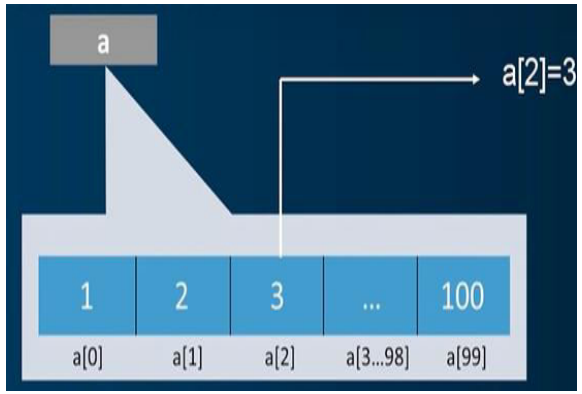
Accessing Array element using for loop

- Without Index

```
In [8]:
   ...: from array import *
   ...: a=array('i',[1,2,3,4,5])     #create 1 D array using array module
   ...: #without index method using for loop access ele.
   ...: for i in a:  #Visit element one by one till not end list
   ...:     print("Array ele:",i)
Array ele: 1
Array ele: 2
Array ele: 3
Array ele: 4
Array ele: 5

In [9]:
```

1	2	3	4	5

- With Index




```
In [9]: from array import *
   ...: a=array('i',[1,2,3,4,5])    #create 1 D array using array module
   ...: n=len(a)    #find length of array
   ...: for i in range(n):  #Visit element index by index till not end list index
   ...:     print("Array ele:",a[i])   #Display value
Array ele: 1
Array ele: 2
Array ele: 3
Array ele: 4
Array ele: 5

In [10]:
```

A[0]	A[1]	A[2]	A[3]	A[4]
1	2	3	4	5

Creating & Accessing Array element 2D

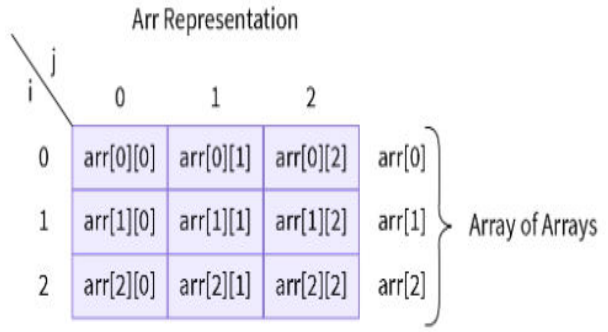
- **2D Array:** `Two dimensional array is an array within an array. It is an array of arrays.`

1.Using Lists of Lists:

It can create a 2D array by using nested lists, where each inner list represents a row of the 2D array.

- To create a two-dimensional array(list) with rows and columns.
- Var_name=[[r1,r2,r3,..,rn],[c1,c2,c3,.....,cn]]

```
In [16]: b=[[1,2,3],[4,5,6]]  #create 2D Array

In [17]: b
Out[17]: [[1, 2, 3], [4, 5, 6]]

In [18]: print(type(b))
<class 'list'>
```



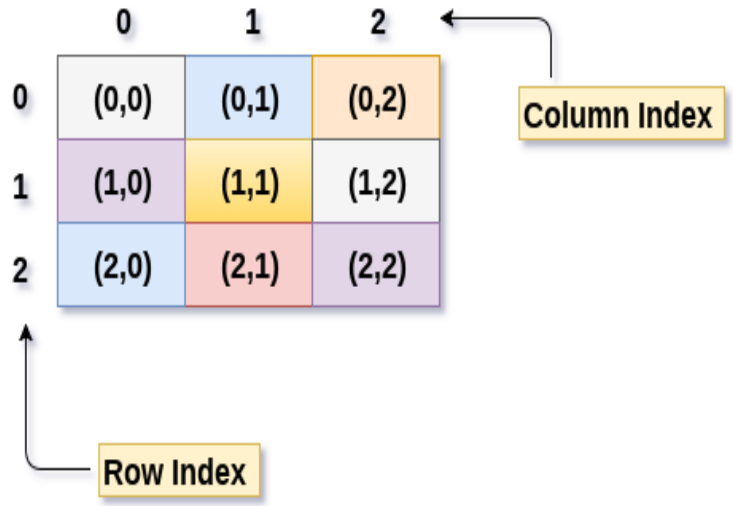
Accessing Array element 2D

- Access element based on index position.
- Syntax: We can get row value using [] operator

array[row index]

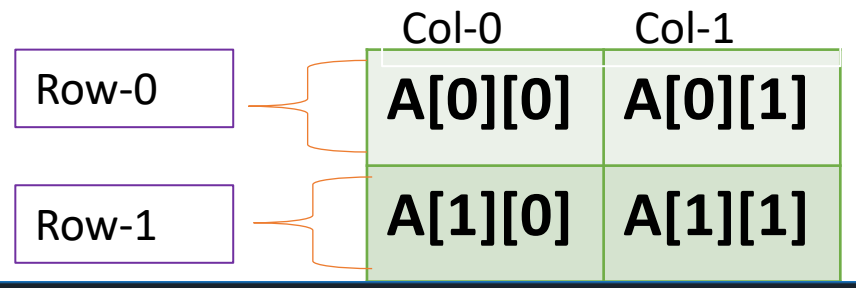
We can get column value using [][]

Array[row index][column index]




```
In [1]: b=[[1,2,3],[4,5,6]]  #create 2D Array (2x3) using lists of lists
   ...: print(b[0]) #get the first row
   ...: print(b[0][2]) #get the first row third element
   ...: for r in b:   # iterate row by row
   ...:     for c in r: # columns is used to iterate the values present in each row.
   ...:         print(c,end = " ")
   ...:     print()
   ...: print(type(b))
[1, 2, 3]
3
1 2 3
4 5 6
<class 'list'>

In [2]:
```

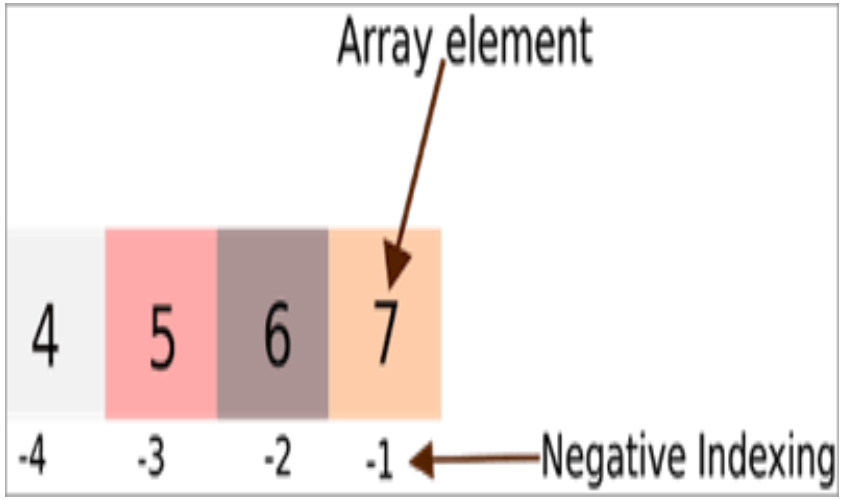
Example




```
In [13]: A=[]    #Two Dimesional array with 2 *2
    ...: for i in range(2):
    ...:     a=[]
    ...:     for j in range(2):
    ...:         a.append(int(input("Enter ele.")))
    ...:     A.append(a)
    ...: arr=np.array(A)  # To Convert Matrix A to arr
    ...: #display of 2D Matrix
    ...: for k in range(2):
    ...:     for m in range(2):
    ...:         print(A[k][m],end=' ')
    ...:     print()
    ...: print(arr)    #2D Array type
Enter ele.1
Enter ele.2
Enter ele.3
Enter ele.4
1 2
3 4
[[1 2]
 [3 4]]

In [14]: print(type(A))
<class 'list'>
```

1) Traversing an Array

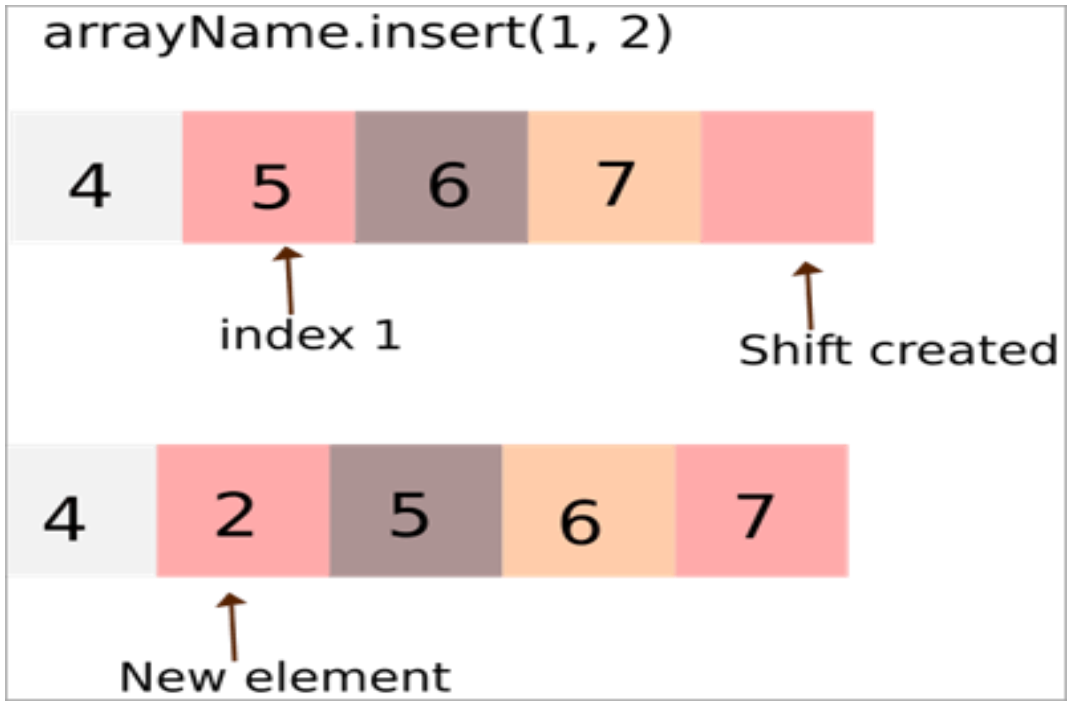
- we can access elements of an array by **indexing**, **slicing** and **looping**.
- **Indexing Array**
- An array element can be accessed by indexing, similar to a list i.e. by using the location where that element is stored in the array.
- The index is enclosed within square brackets **[ ]**, the first element is at index **0**, next at index **1** and so on.




```
In [23]: a
Out[23]: array('i', [8, 9, 3, 4])

In [24]: a[0]
Out[24]: 8

In [25]: a[-1]
Out[25]: 4
```

2) Inserting into an Array

- Insert operation is to insert one or more data elements into an array
- The same goes for a List – an array uses its method **insert(i, x)** to add one to many elements in an array at a particular index.




```
In [26]: a
Out[26]: array('i', [8, 9, 3, 4])

In [27]: a.insert(8, -1)

In [28]: a
Out[28]: array('i', [8, 9, 3, 4, -1])

In [29]: a.insert(1,60)

In [30]: a
Out[30]: array('i', [8, 60, 9, 3, 4, -1])
```

3.Deletion Operation

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- Here, we remove a data element at the middle of the array using the python in-built **remove() and pop().** method.

```
In [31]: a
Out[31]: array('i', [8, 60, 9, 3, 4, -1])

In [32]: a.remove(4)

In [33]: a
Out[33]: array('i', [8, 60, 9, 3, -1])

In [34]: a.pop()
Out[34]: -1

In [35]: a
Out[35]: array('i', [8, 60, 9, 3])
```

4.Searching an Array

- Array allows us to search it's elements. It provides a method called **index(x)**.
- This method takes in an element, **x**, and returns the index of the first occurrence of the element.

```
In [37]: b
Out[37]: array('d', [2.3, 3.5, 6.2])

In [38]: b.index(3.5)
Out[38]: 1

In [39]: try:
    ...:         print(b.index(3.3))
    ...:         print(b.index(1))
    ...:
    ...: except ValueError as e:
    ...:         print(e)
array.index(x): x not in array

In [40]:
```

Modify or Updating an Array Element in an Index

- We can update an array's element by using indexing. Indexing allows us to modify a single element and unlike **insert()**,
- it raises an **IndexError** exception if the index is out of range.

```
In [41]: from array import array

In [42]: a = array('i', [4,5,6,7])

In [43]: a[1] = 9

In [44]: a
Out[44]: array('i', [4, 9, 6, 7])

In [45]: len(a)
Out[45]: 4

In [46]:
```

Array vs List

S.No.	List	Array
1	List is used to collect items that usually consist of elements of multiple data types.	An array is also a vital component that collects several items of the same data type.
2	List cannot manage arithmetic operations.	Array can manage arithmetic operations.
3	It consists of elements that belong to the different data types.	It consists of elements that belong to the same data type.
4	When it comes to flexibility, the list is perfect as it allows easy modification of data.	When it comes to flexibility, the array is not suitable as it does not allow easy modification of data.
5	It consumes a larger memory.	It consumes less memory than a list.
6	In a list, the complete list can be accessed without any specific looping.	In an array, a loop is mandatory to access the components of the array.
7	It favors a shorter sequence of data.	It favors a longer sequence of data.

Numpy array

Introduction to Numpy

- Numpy is a python package and it stands for **numerical python**
- Fundamental package for numerical computation in python
- Supports **N-Dimensional array** objects that can be used for processing multidimensional data
- Supports different data types

Creation of an array

- A numpy array is grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
- The number of dimensions is the rank of array.
- The shape of an array is a tuple of integers giving the size of the array along each dimesions.
- **Syntax:**

numpy.array(*object*, *dtype=None*)

Example

```
In [17]: import numpy as np

In [18]: lst=[1,2,3,4,5]

In [19]: ar=np.array(lst)

In [20]: ar=np.array(lst,dtype='int')

In [21]: ar
Out[21]: array([1, 2, 3, 4, 5])

In [22]: print(type(ar))
<class 'numpy.ndarray'>

In [23]: lst=[1,2,3,4,5.0]

In [24]: ar=np.array(lst)

In [25]: print(ar)
[1. 2. 3. 4. 5.]
```

NumPy array attributes

- In NumPy, attributes are properties of NumPy arrays that provide information about the array's shape, size, data type, dimension, and so on.

Attributes	Descriptions
ndim	returns number of dimension of the array
size	returns number of elements in the array
shape	returns the size of the array in each dimension.
itemsize	returns the size (in bytes) of each elements in the array

Example

```
In [43]: ar
Out[43]: array([1, 2, 3, 4, 5])

In [44]: print(ar.ndim)
1

In [45]: print(ar.size)
5

In [46]: print(ar.shape)
(5,)

In [47]: print(ar.itemsize)
4
```

Example

```
In [53]: arr=np.arange(1,7)   #create array with arnage() method

In [54]: arr
Out[54]: array([1, 2, 3, 4, 5, 6])

In [55]: arr.reshape(2,3) #change array dimensions 1D to 2D
Out[55]:
array([[1, 2, 3],
       [4, 5, 6]])

In [56]: arr.reshape(2,3,1) #change array dimensions 2D to 3D
Out[56]:
array([[[1],
        [2],
        [3]],

       [[4],
        [5],
        [6]]])

In [57]:
```

NumPy Arithmetic Operations

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
print(x)
print(y)
```

```
[[1. 2.]
 [3. 4.]]
[[5. 6.]
 [7. 8.]]
```

Addition

```
In [20]:
```

```
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

Subtract

```
In [21]:
```

```
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

Example

In [22]:

```
print(x * y)
print(np.multiply(x, y))
print(x.dot(y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[19. 22.]
 [43. 50.]]
```

In [23]:

```
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19. 22.]
 [43. 50.]]
[[19. 22.]
 [43. 50.]]
```

Example

In [24]:

```
print(x / y)
print(np.divide(x, y))
```

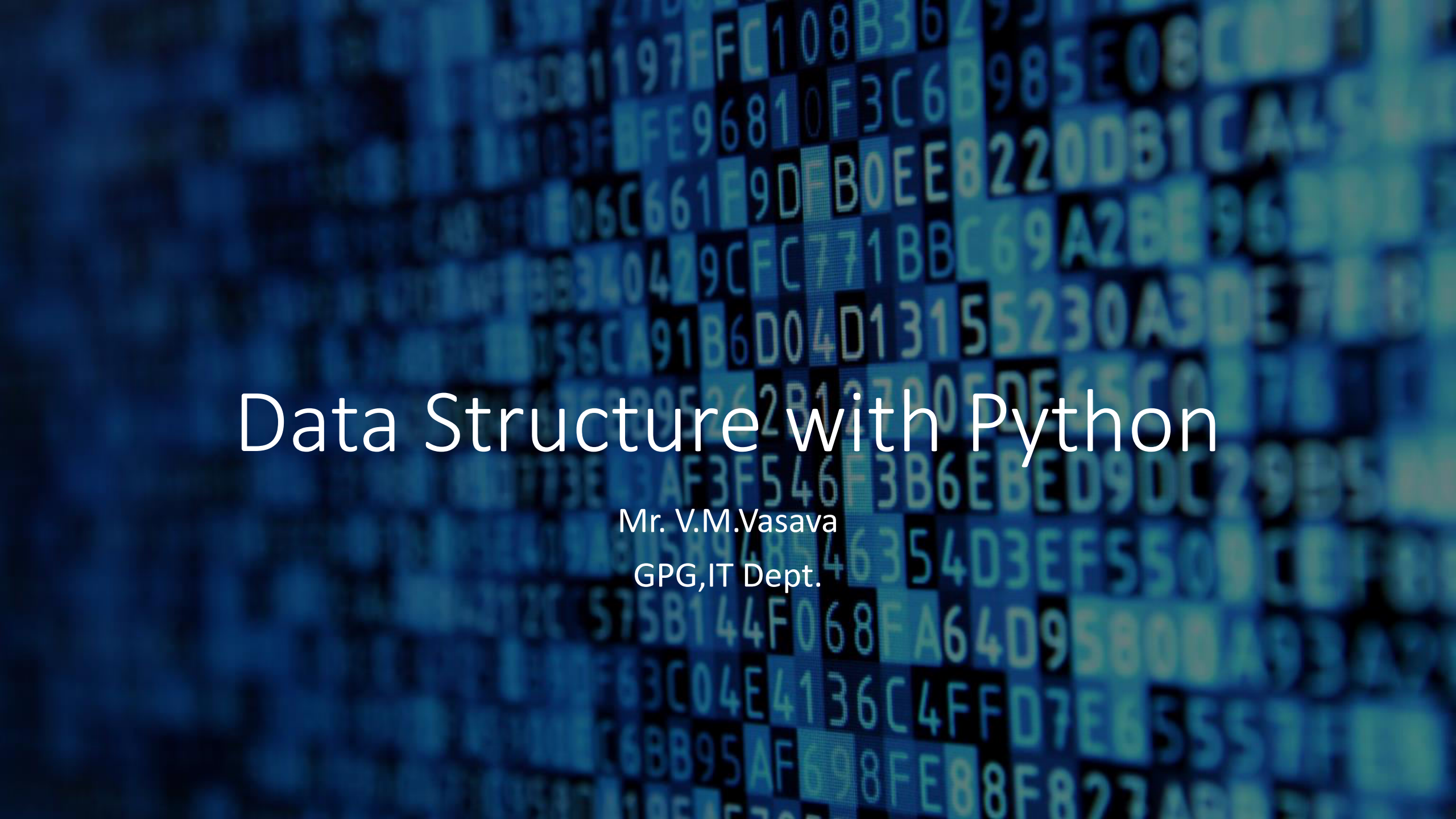
```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

In [26]:

```
print(np.sum(x))          # Compute sum of all elements
print(np.sum(x, axis=0))  # Compute sum of each column
print(np.sum(x, axis=1))  # Compute sum of each row
```

```
10.0
[4. 6.]
[3. 7.]
```

Any Questions



Data Structure with Python

Mr. V.M.Vasava

GPG,IT Dept.

Agenda

Introduction

Basic Terminology of DS

Classification of DS

Data Structures

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Data Structure is representation of the logical relationship existing between individual elements of data.

Data Structure mainly specific four things:

- Organization of data
- Accessing Methods
- Degree of associativity
- Processing alternatives for information

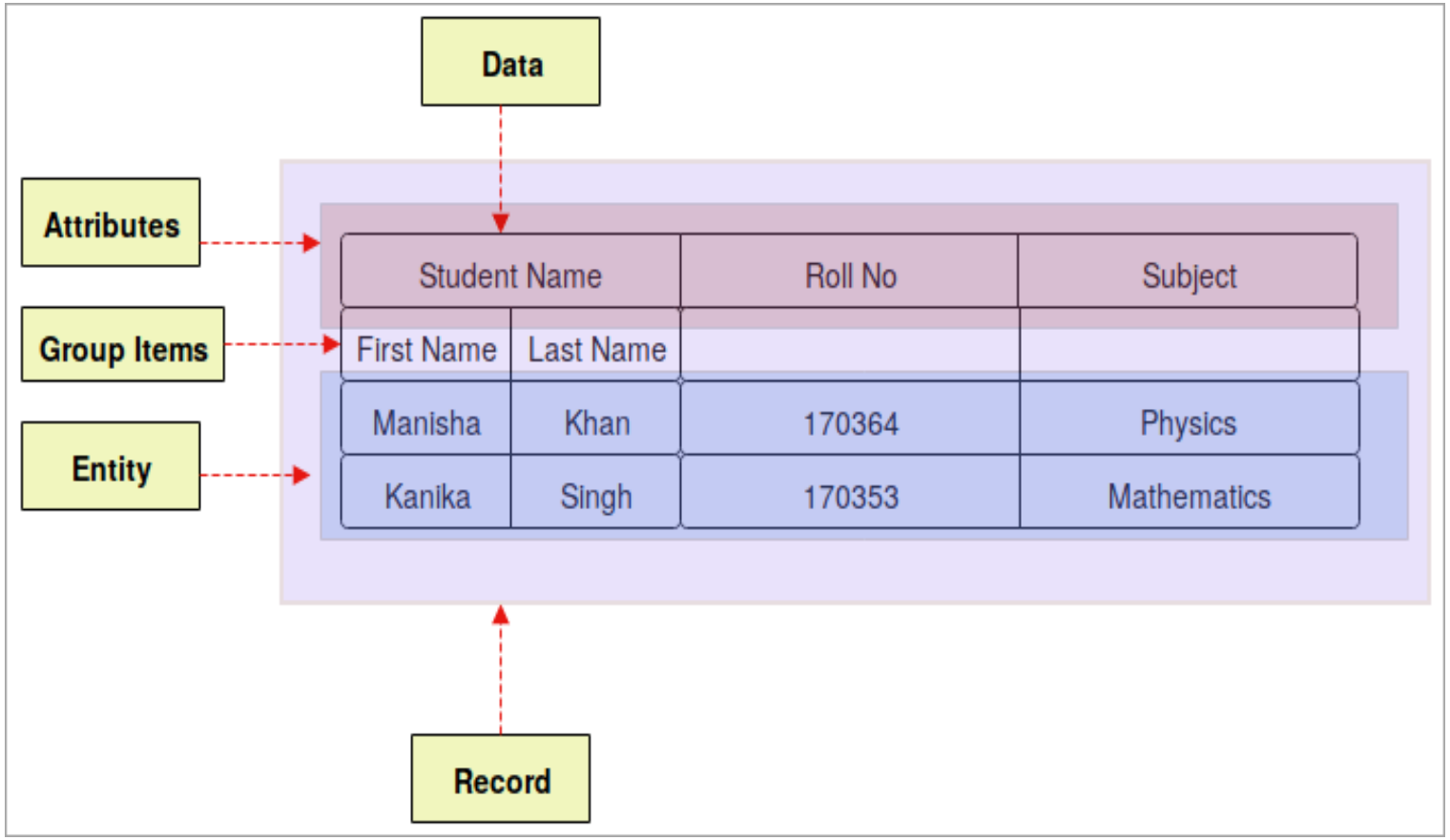
Data Structure

- In computer science, a **data structure** is a data organization, management, and storage format that enables efficient access and modification.
- In simple words,
- Data Structure is a way in which data is stored on computer.

Example

Organizing fruits in these baskets are **similar** to organizing integers, string data and floating point numbers.

Basic Concepts of Data



Data

Define : **Data**

Data are simple values or set of values. For example 101 Nivedita A

Data can be defined as a representation of facts, concepts, or instructions in a formalized manner, which should be suitable for communication, interpretation, or processing by human or electronic machine.

Data is represented with the help of characters such as alphabets (A-Z, a-z), digits (0-9) or special characters (+,-,/,*,<,>,= etc.).

Types of Data:

Atomic Data

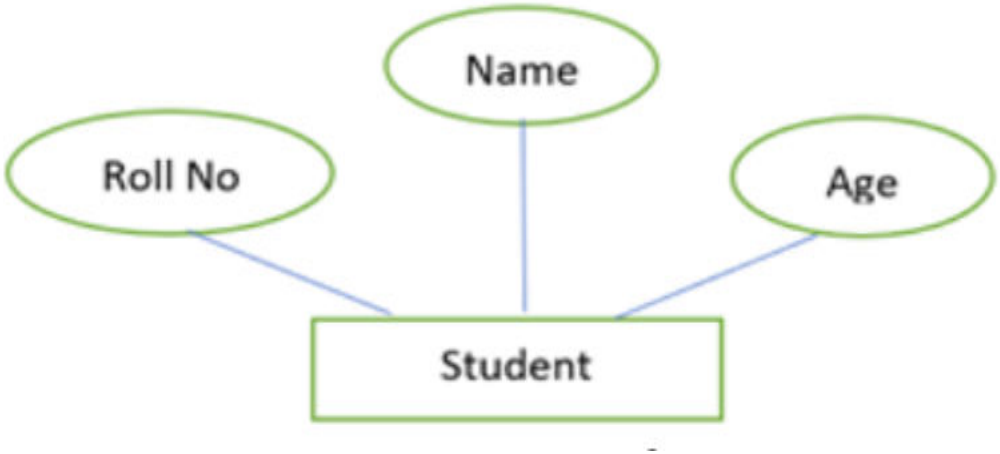
Composite Data

Continued..

- Data Item - single unit of values. For example : Nivedita

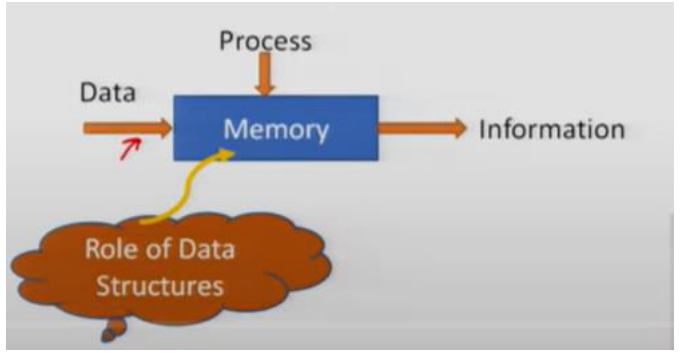
- Entity- it is something that has attributes or properties.

  For example:



Basic Terminology of DS

Information: information means meaningful or processed data.



For example:

RollNo: 101

Name: Nivedita

Field: Field is a single elementary unit of information representing the attribute of an entity.

For example:

Roll No	Name

Field

Continued..

Record: Record can be defined as the collection of field values of given entity.

- For example name, rollno, address, course and marks can be grouped together to form the record for the student.

Name	RollNo
Nidhi	101
Ankita	102

→ Record

File: A File is a collection of various records of one type of entity.

Group Items: The data items which are further subdivided into parts are known as group items.

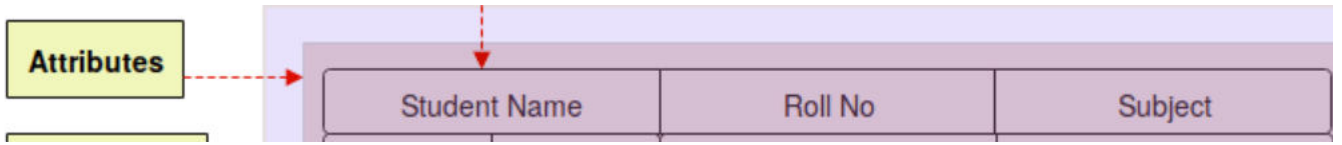
for example, name of a student can have first name and the last name.

Attribute

Name	Roll no	Subject
Kanika	9742912	Physics
Manisha	8536438	Mathematics

The attribute is the column that stores the information related to the particular name of the column.

For example, "Name = Kanika" here the attribute is "Name" and "Kanika" is an entity.



Area of uses DS

Data Structures are widely used in almost every aspect of **Computer Science**

1. Operating System

2. Compiler Design

3. Artifical intelligence

4. Graphics

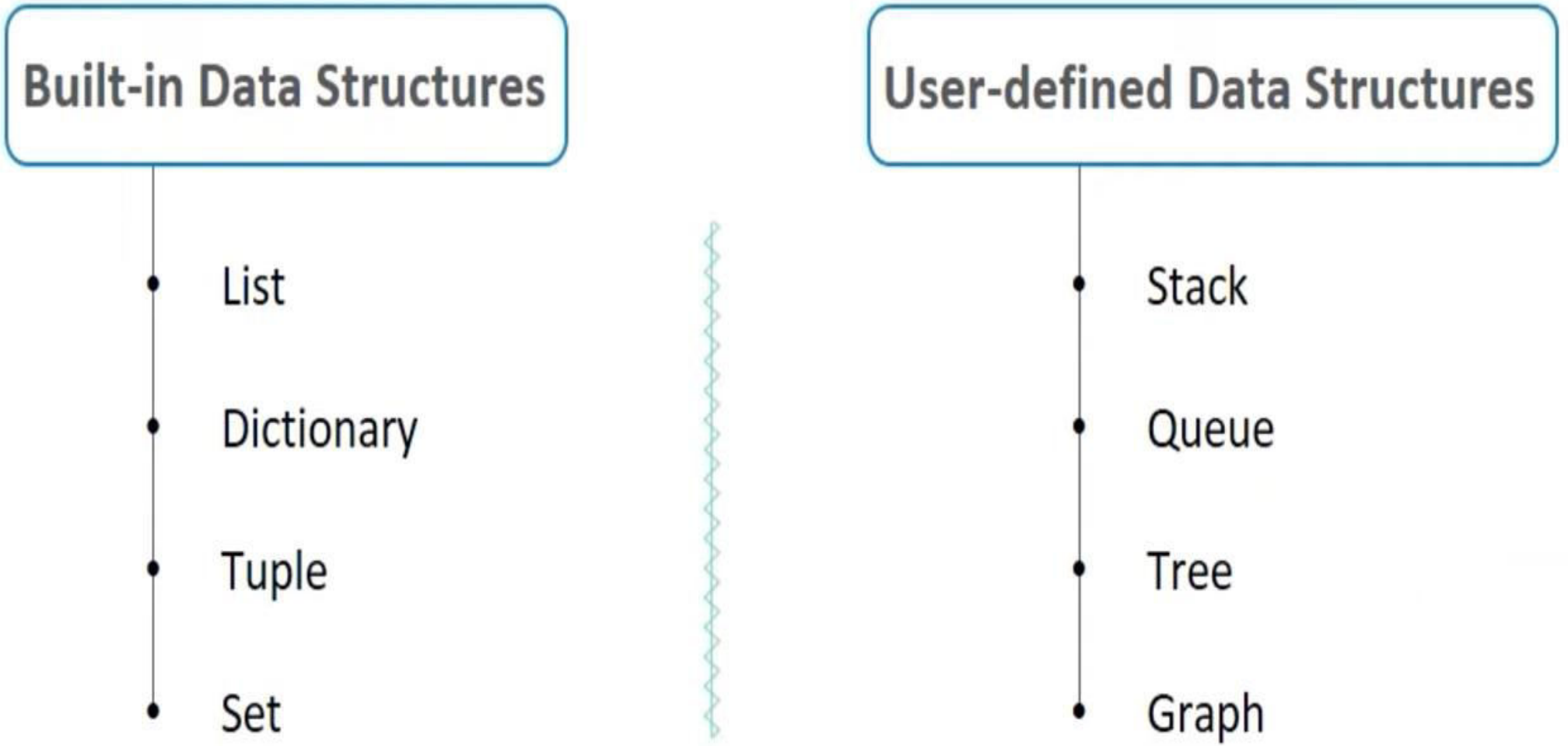
5. Database management

6. Numerical analysis and many more.

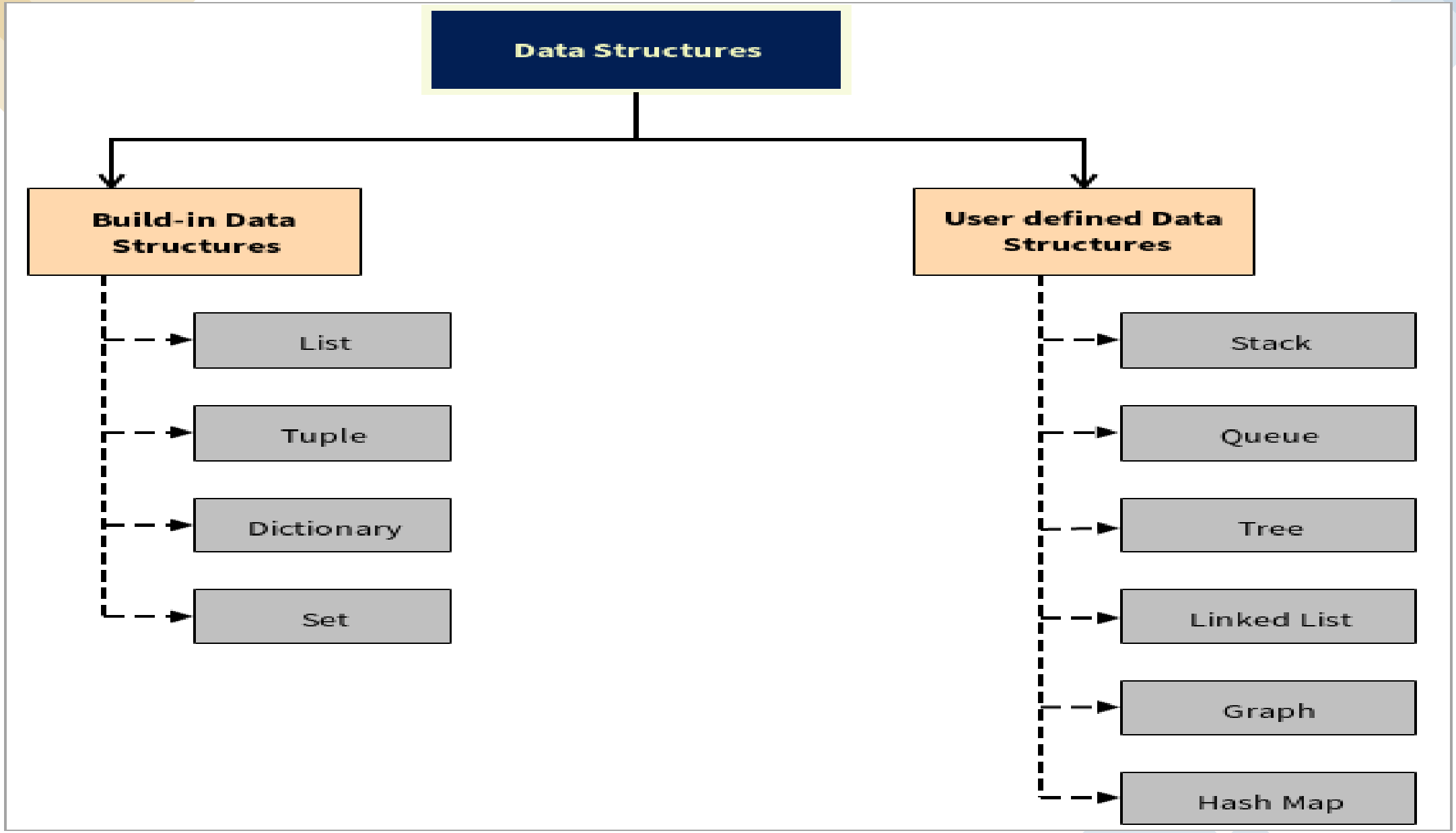
Types of DS

- There are main 2 Types:

Classification of DS



Classification of DS



List in python

Lists

- Lists are used to store data of different data types in a sequential manner.
- Every element of the list has an address which we can call the index of an element.
- It starts from 0 and ends at the last element. **positive index**.
  - For notation, it is like (0, n-1).
- It supports negative indexing as well which begins from -1 (access elements from the last to first.)
- For example :factors = [1,2,5,10]

  names = ["Anand",“Ahmedabad",“surat"]
- Type need not be uniform

  mixed = [3, True, “Black"]

Tuple in python

Tuple is an ordered collection of elements enclosed within ()

Tuple

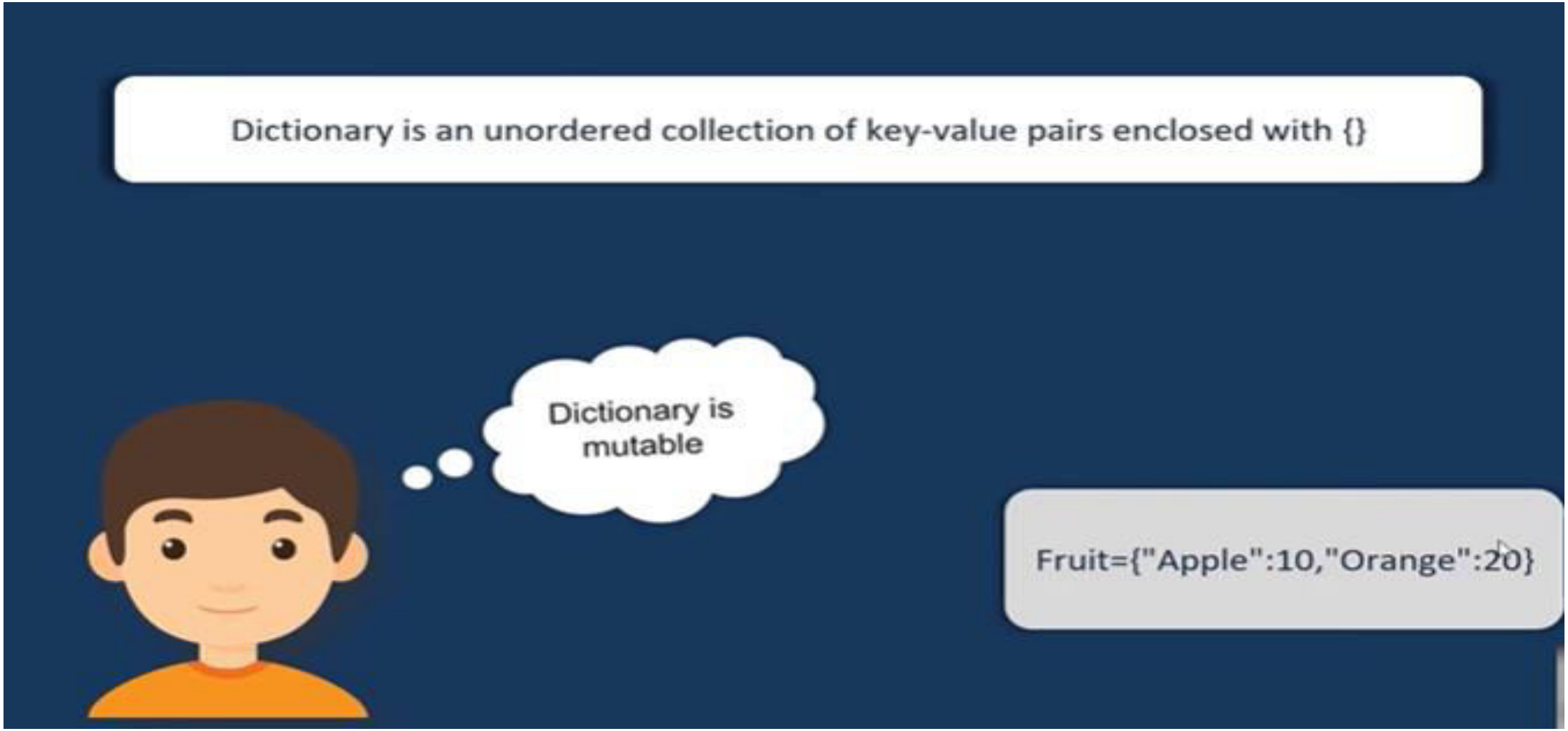
A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Python Tuple is a collection of objects separated by commas. In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition but a tuple is immutable, unlike lists which are mutable.

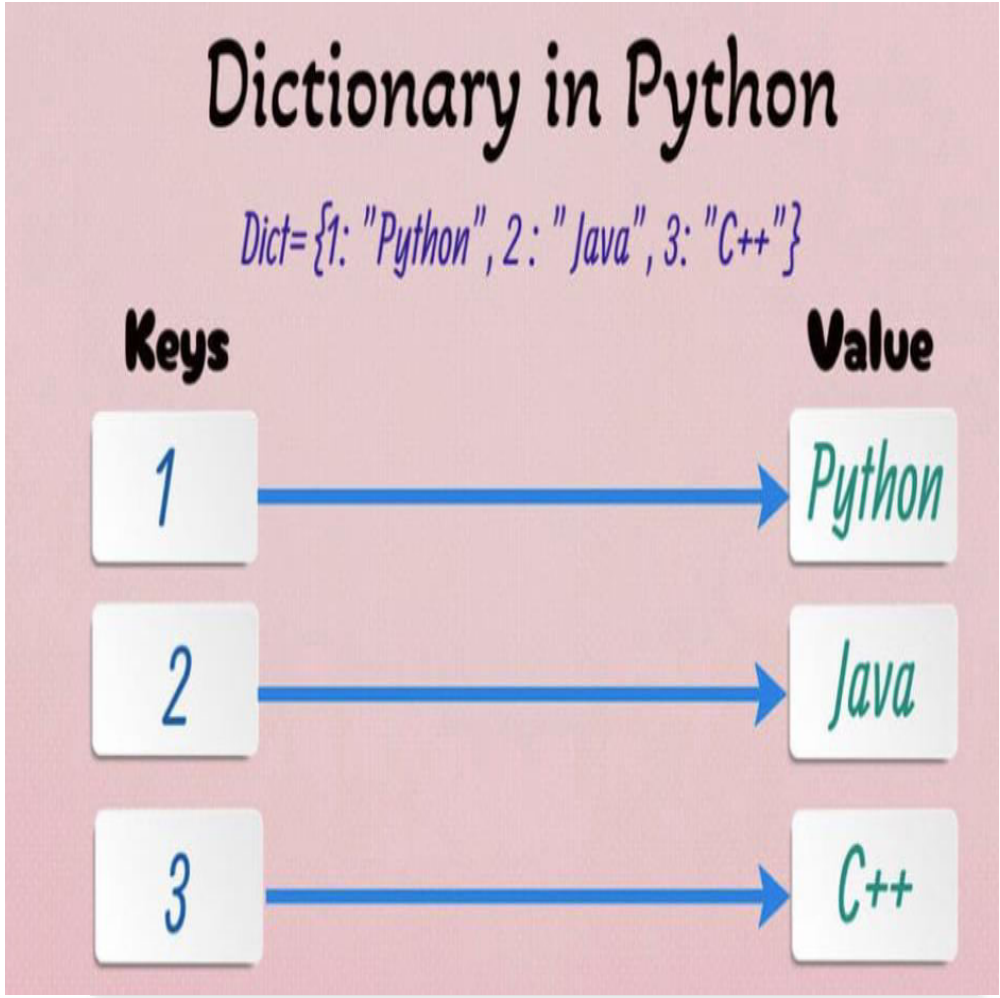
Var=(“welcome”, “IT”, ”GGP”)

Dictionary in python

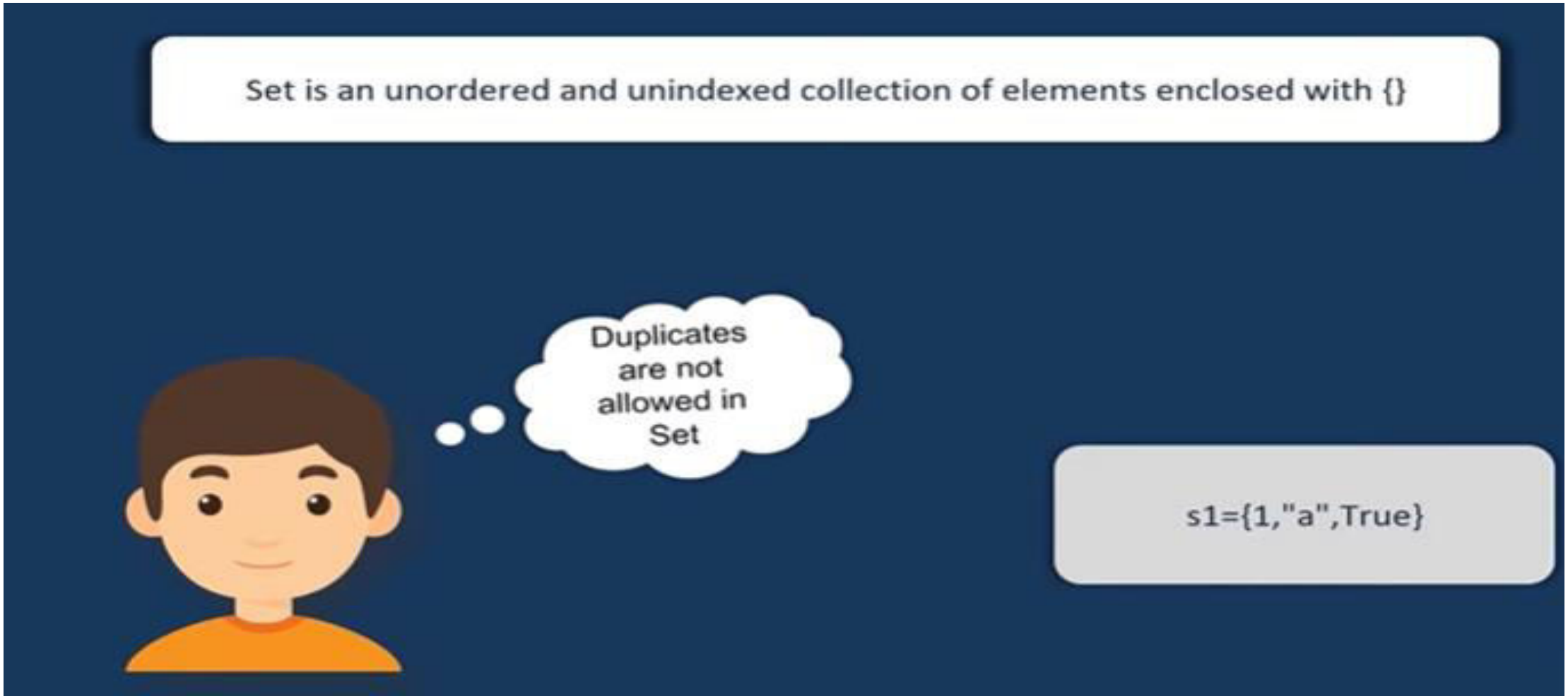


Dictionary

- Python dictionary is an unordered collection of items.
- Each item of a dictionary has a **key/value pair.**
- Dictionaries are optimized to retrieve values when the key is known.
- Dictionaries are case-sensitive in nature.

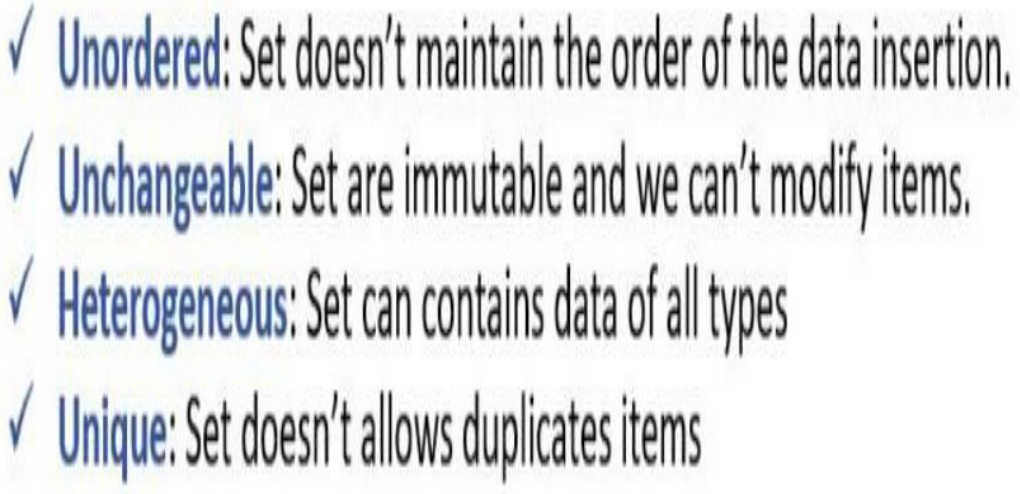


Set in python



Set

- A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
- Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.



Python Data Structure Operations

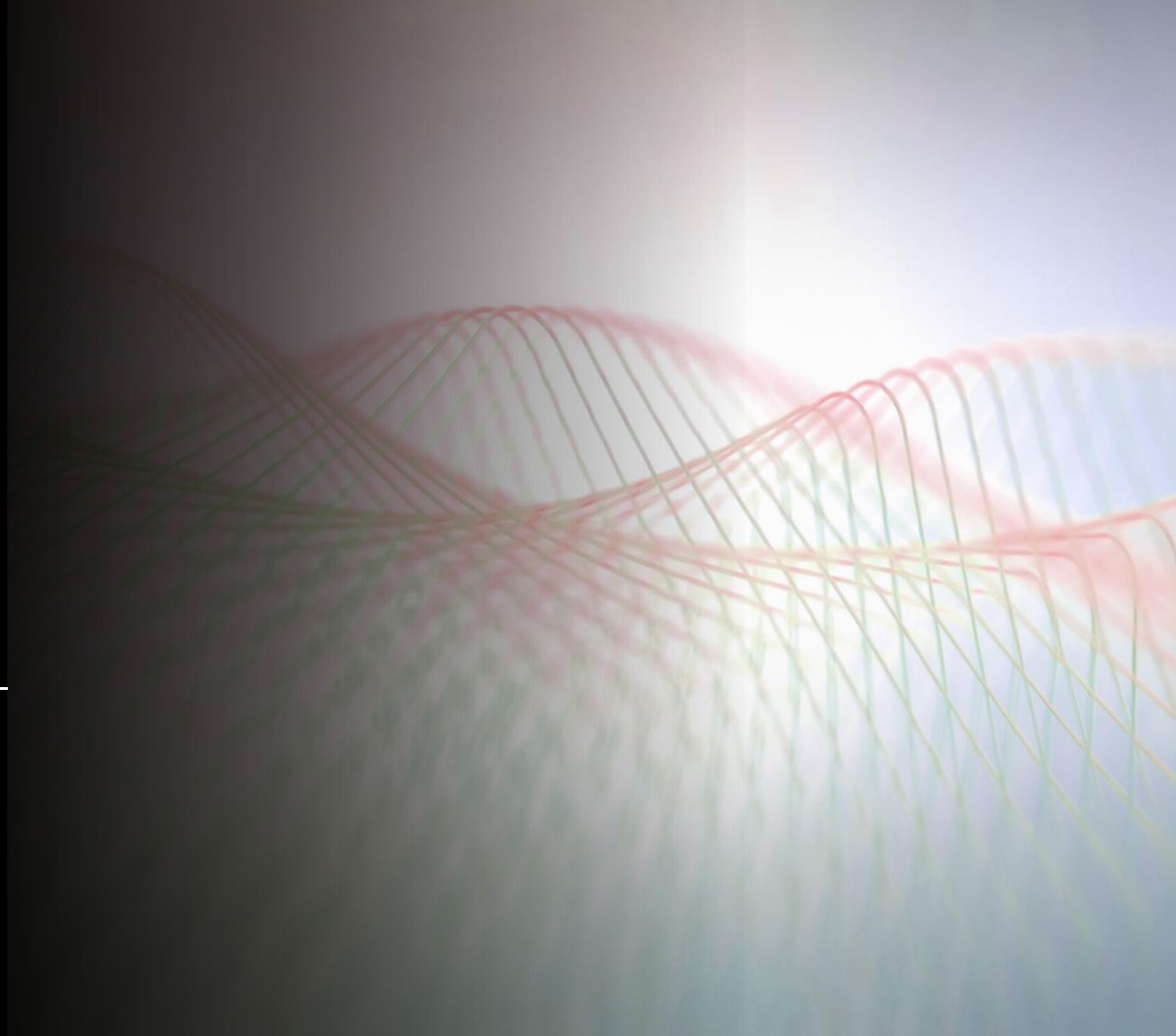
- **Traversing:** Traversing a Data Structure means to visit the element stored in it.
- **Searching:** Searching means to find a particular element in the given data-structure.
- **Insertion:** Insertion means to add an element in the given data structure at any time and anywhere.
- **Deletion:** Deletion means to delete an element in the given data structure.
- **Sorting:** The process of arranging the data elements in a data structure in a specific order (ascending or descending) by specific key values is called sorting.

Thank You..

Data Structure with Python

Mr. V.M.Vasava

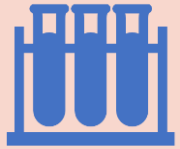
GPG,IT Dept.



Agenda

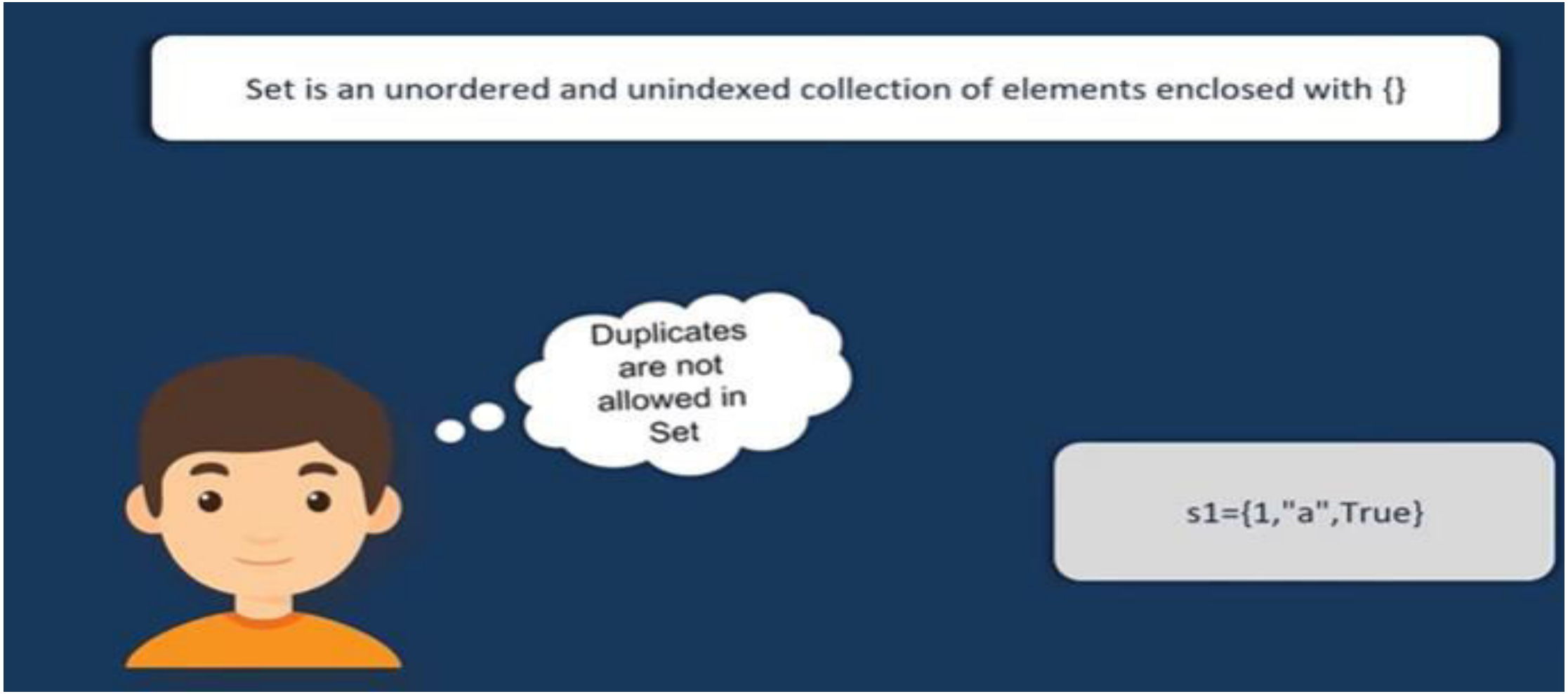
Set

Basic Operation of Set



Methods of Set

Set in python



Set

$S = \{20,\ 'Jessa',\ 35.75\}$

- ✓ **Unordered**: Set doesn't maintain the order of the data insertion.
- ✓ **Unchangeable**: Set are immutable and we can't modify items.
- ✓ **Heterogeneous**: Set can contains data of all types
- ✓ **Unique**: Set doesn't allows duplicates items

- A set is an **unordered** collection of items. Every set element is unique and must be **immutable**.
- Set are represented by { }.
- The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table.
- For example:

```
In [6]: Days = {"Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"}

In [7]: Days
Out[7]: {'Friday', 'Monday', 'Saturday', 'Sunday',
'Thursday', 'Tuesday', 'Wednesday'}

In [8]:
```

Example

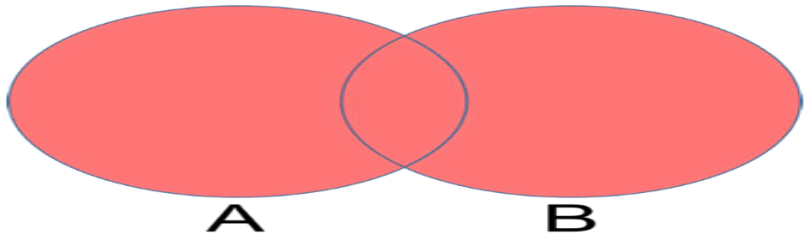
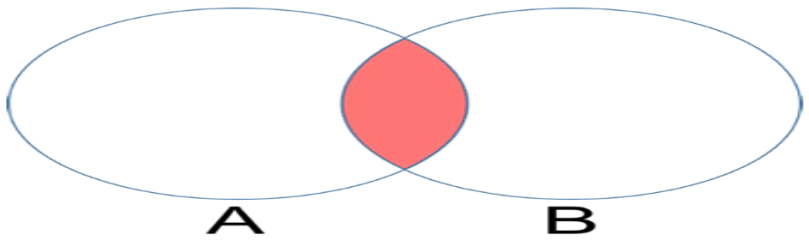
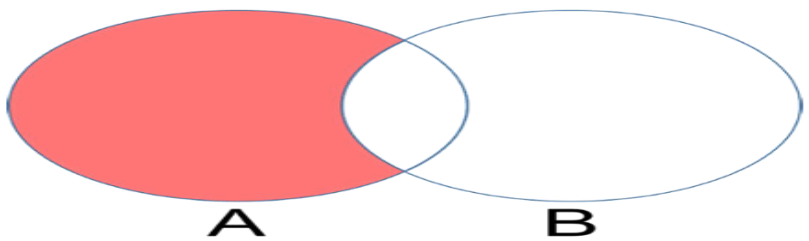
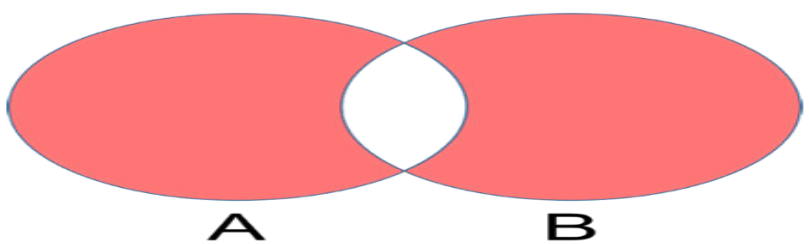
- In python set create two ways:

1. curly braces {}

2. set() function

```
In [45]: s={} #create empty set

In [46]: st=set() #create empty set with set() function

In [47]: s={1,2,3,"Ashwini"} #create set with values

In [48]: s
Out[48]: {1, 2, 3, 'Ashwini'}

In [49]: print(set([1,2,1,2,"Vilas"])) #create set with set() fun and duplicate
{1, 2, 'Vilas'}

In [50]: type(st)
Out[50]: set
```

Set Operation

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.

Operations

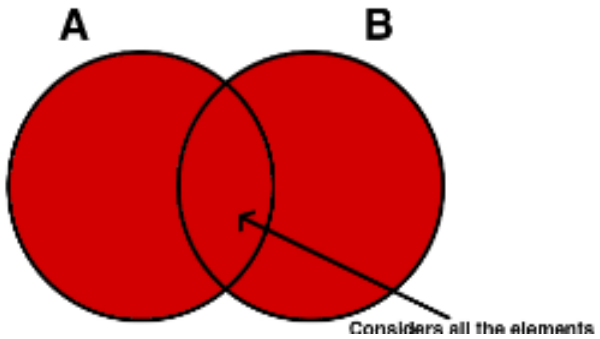
1. Set Union

Union of A and B is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the union() method.

Python Set union() - Returns a new set by combining all the distinct elements of the specified sets

Syntax - A.union(set2, set3, ..., setn)



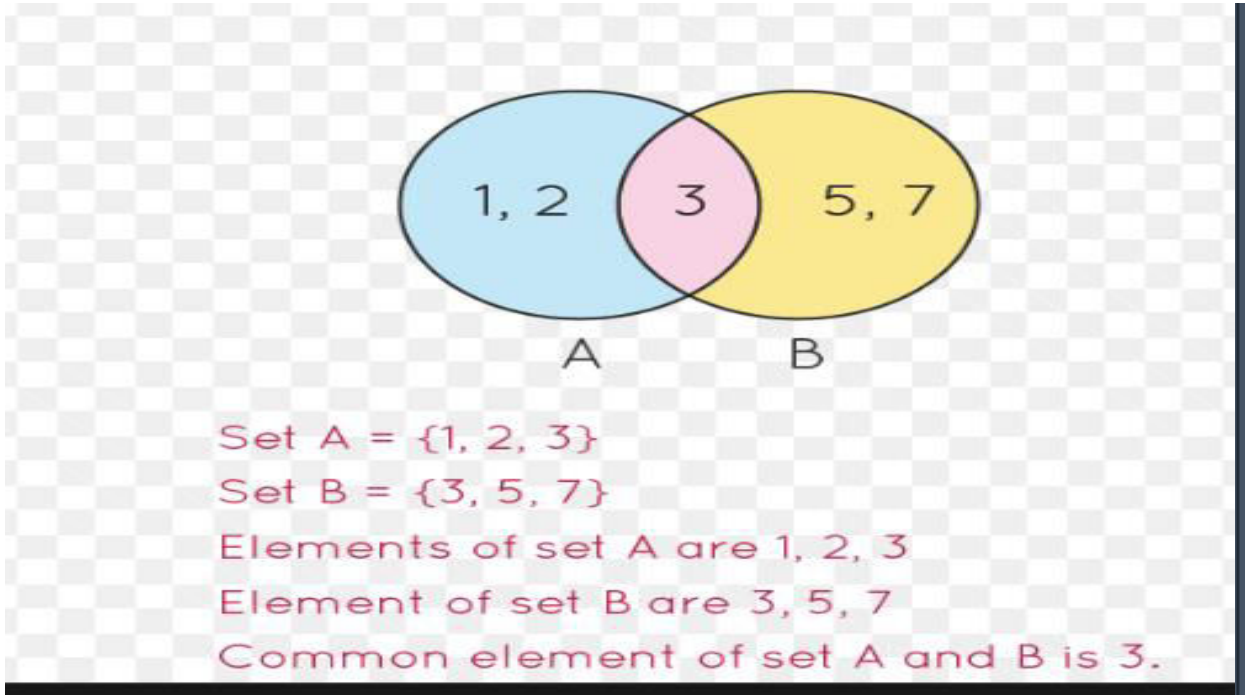
top

```
In [18]: A={1,2,3}

In [19]: B={3,5,7}

In [20]: A|B
Out[20]: {1, 2, 3, 5, 7}

In [21]: A.union(B)
Out[21]: {1, 2, 3, 5, 7}

In [22]:
```

Operations of set

2.Intersection

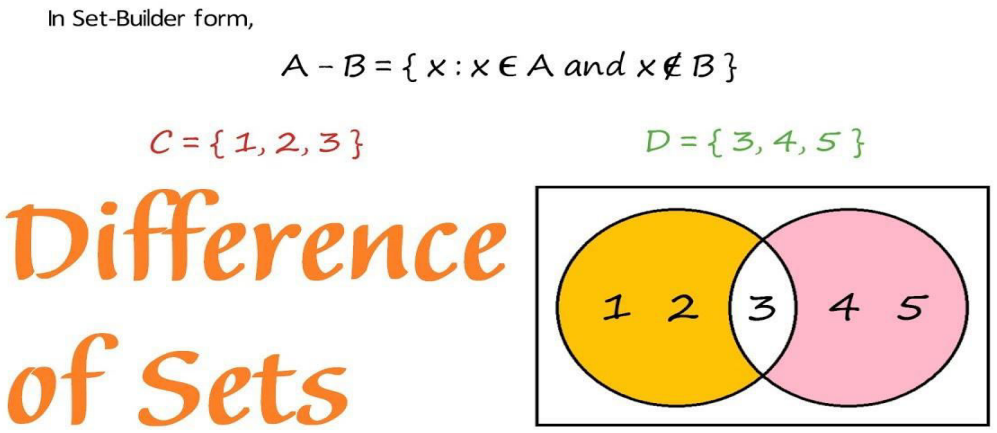
- Intersection of A and B is a set of elements that are common in both the sets.
- Intersection is performed using & operator. Same can be accomplished using the **intersection()** method.




```
In [22]: A
Out[22]: {1, 2, 3}

In [23]: B
Out[23]: {3, 5, 7}

In [24]: A & B
Out[24]: {3}

In [25]: A.intersection(B)
Out[25]: {3}
```

Operations of set

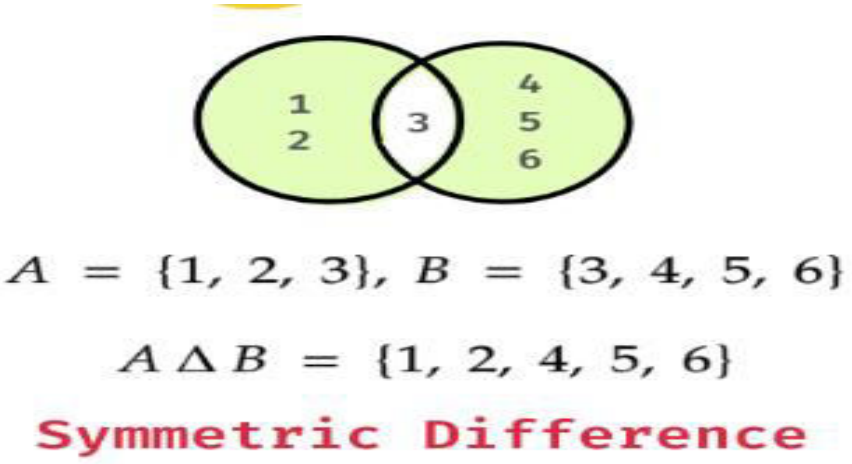
- **3.Set Difference**
- Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A.
- **Difference is performed using - operator. Same can be**




```
In [27]: A
Out[27]: {1, 2, 3}

In [28]: B
Out[28]: {3, 4, 5}

In [29]: A-B
Out[29]: {1, 2}

In [30]: B-A
Out[30]: {4, 5}

In [31]: A.difference(B)
Out[31]: {1, 2}

In [32]: B.difference(A)
Out[32]: {4, 5}
```

Set Symmetric Difference

- Symmetric Difference of A and B is a set of elements in A and B but **not** in both.
- Symmetric difference is performed using ^ operator.
- Same can be accomplished using the method symmetric_difference().




```
In [34]: A
Out[34]: {1, 2, 3}

In [35]: B
Out[35]: {3, 4, 5}

In [36]: A^B
Out[36]: {1, 2, 4, 5}

In [37]: A.symmetric_difference(B)
Out[37]: {1, 2, 4, 5}

In [38]: B.symmetric_difference(A)
Out[38]: {1, 2, 4, 5}

In [39]:
```

Set Methods

Methods	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
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
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
remove()	Removes an element from the set. If the element is not a member, raises a KeyError

Example

```
In [39]: vowels = {'a', 'e', 'i', 'u'}

In [40]: vowels.add('o')

In [41]: vowels
Out[41]: {'a', 'e', 'i', 'o', 'u'}

In [42]: vowels.remove('o')

In [43]: vowels
Out[43]: {'a', 'e', 'i', 'u'}

In [45]: vowels.clear()

In [46]: vowels
Out[46]: set()
```

Example

```
In [1]: my_set = {1, 3, 4, 5, 6}

In [2]: my_set.discard(4)

In [3]: my_set
Out[3]: {1, 3, 5, 6}

In [4]: my_set.remove(6)

In [5]: my_set
Out[5]: {1, 3, 5}

In [6]: my_set.discard(2)

In [7]: my_set.remove(2)
Traceback (most recent call last):

  File "C:
\Users\Admin\AppData\Local\Temp\ipykernel_43564\360328
8120.py", line 1, in <cell line: 1>
    my_set.remove(2)
```

Any Questions ???

Data Structure with Python

Mr. V. M. Vasava

GPG,IT Dept.

Agenda

Algorithm

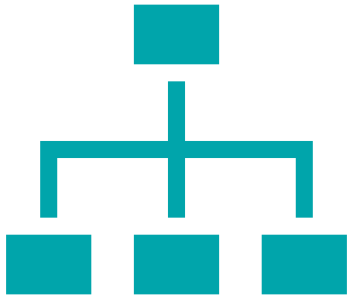
Characteristics of Algorithm

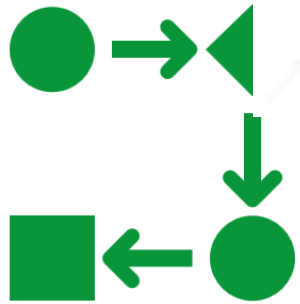
Asymptotic Notation

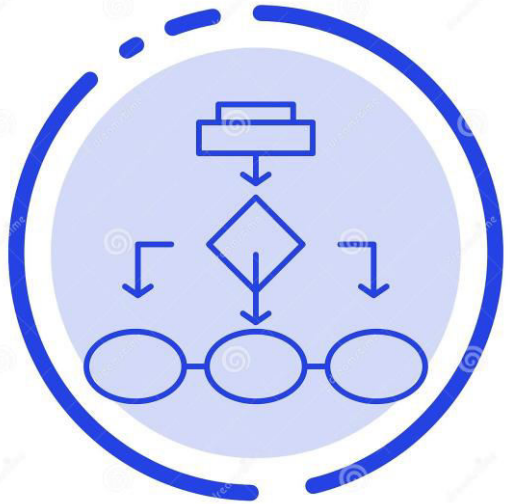
Algorithm

- Data Structure and algorithm are the foundation of computer programming.
- Algorithmic thinking ,problem solving & data structures are vital role for software engineers.
- The terms "Algorithm" comes from :
- Derived from **Muhammad al –Khwarizmi**, a Person mathematician and Astronomer.

Algorithm







Algorithm is finite set of logic or instructions, written in order for accomplish the certain predefined task.

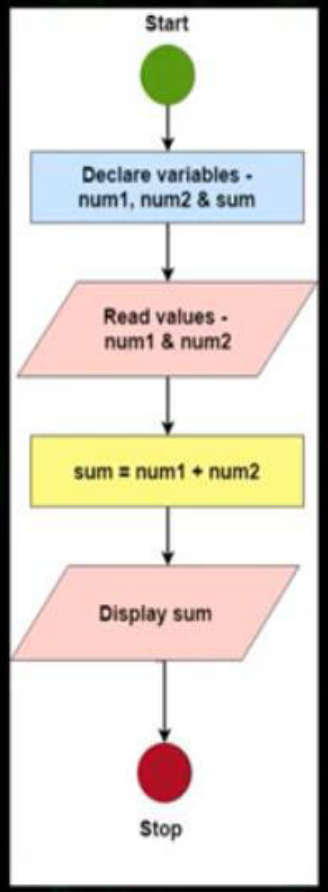
An algorithm is a step by step procedure to solve a problem.

How to Write an Algorithm ?

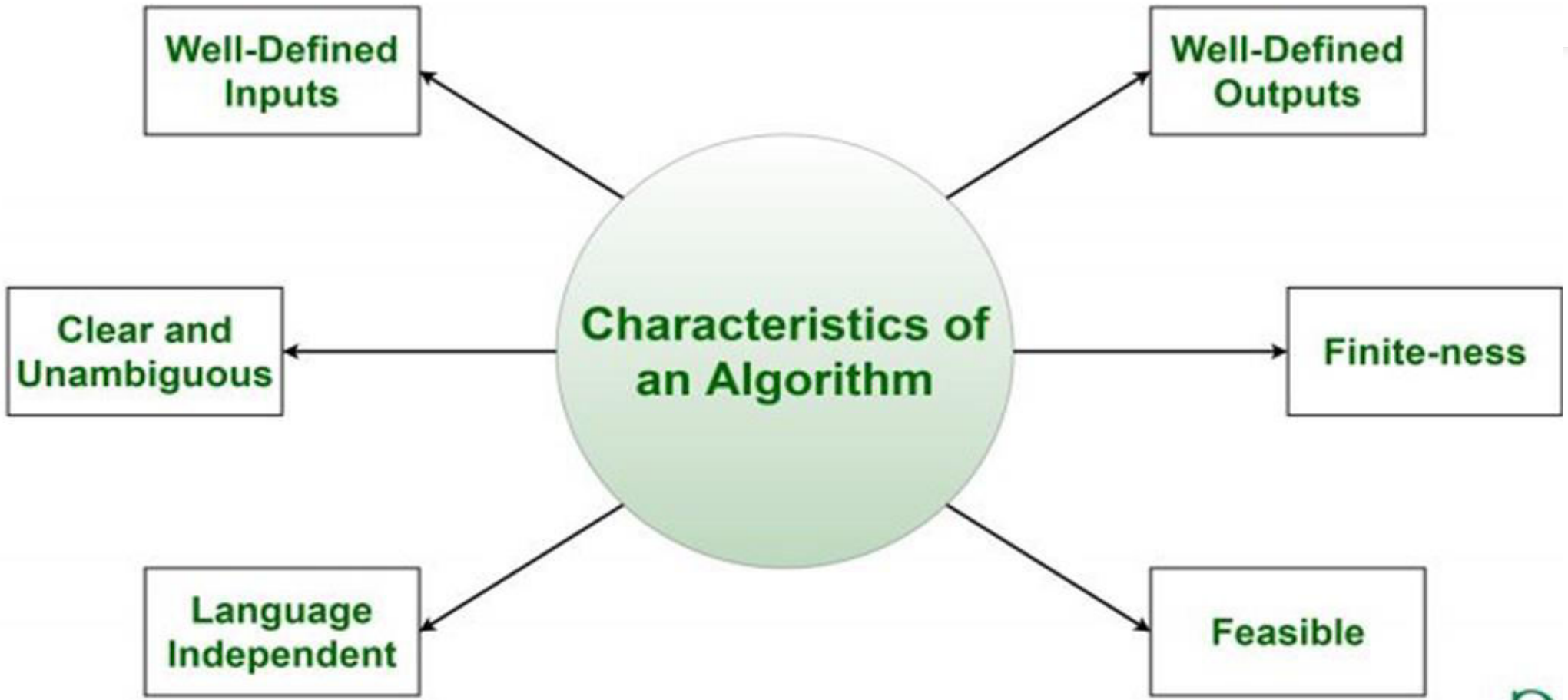
Example of an Algorithm in Programming –

Write an algorithm to add two numbers entered by user. –

1. Step 1: Start
2. Step 2: Declare variables num1, num2 and sum.
3. Step 3: Read values num1 and num2.
4. Step 4: Add num1 and num2 and assign the result to sum.(sum←num1+num2)
5. Step 5: Display sum
6. Step 6: Stop



Characteristics of an Algorithm



Characteristics of an Algorithm

Input- There should be 0 or more inputs supplied externally to the algorithm.

Output- There should be atleast 1 output obtained.

Definiteness- Every step of the algorithm should be clear and well defined.

Finiteness- The algorithm should have finite number of steps.

Correctness- Every step of the algorithm must generate a correct output.

Analysis of Algorithm

Performance of algorithm is a process of making evaluative judgement about algorithm.

The performance of algorithm is measured on the basis of following properties:

1. Time complexity

2.Space Complexity

Time & Space Complexity

Time Complexity

It is a way of representing the amount of time needed by a program to run to the completion.

Space Complexity

It is the amount of memory space required by an algorithm, during a course of its execution.

Algorithm Measuring Techniques

1.Measuring time to execute

2.Counting operations involved

3.Abstract notion order of growth

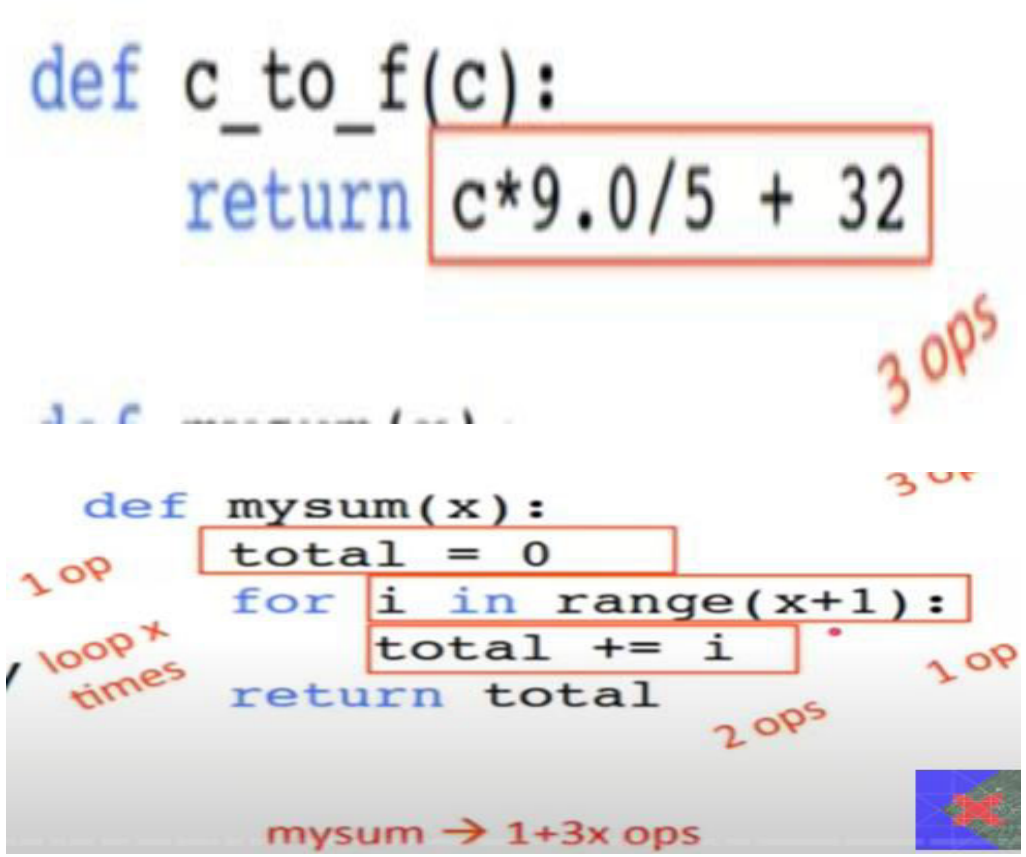
For Example:

```
import time

start = time.time()
i = 1
while i<201:
    print(i)
    i+=1

print(time.time() - start)
```

Different time for different algorithm	✓
Time varies if implementation changes	✗
Different machines different time	✗
Does not work for extremely small input	✗
Time varies for different inputs, but can't establish a relationship	✗

Counting operations

- **Constant Time**
- Mathematical operations
- Comparisons
- Assignments
- Accessing object to memory
- Counting the no. of operation executed sized on input.



Orders of growth

Goals:

- want to evaluate program's efficiency when **input is very big**
- want to express the **growth of program's run time** as input size grows
- want to put an **upper bound** on growth – as tight as possible
- do not need to be precise: **"order of" not "exact"** growth
- we will look at **largest factors** in run time (which section of the program will take the longest to run?)

Example

```
def fact_iter(n):
    """assumes n an int >= 0"""
    answer = 1
    while n > 1:
        answer *= n
        n -= 1
    return answer
```

answer = answer * n

temp = n-1
n = temp

- computes factorial
- number of steps: 1 + 5n + 1
- worst case asymptotic complexity: O(n)

Asymptotic Notation

Asymptotic means approaching a value or curve arbitrarily closely…

Asymptotic analysis : It is a technique of representing limiting behavior. It can be used to analyze the performance of an algorithm for some large data set.

Asymptotic Notations are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases.

Asymptotic Notation

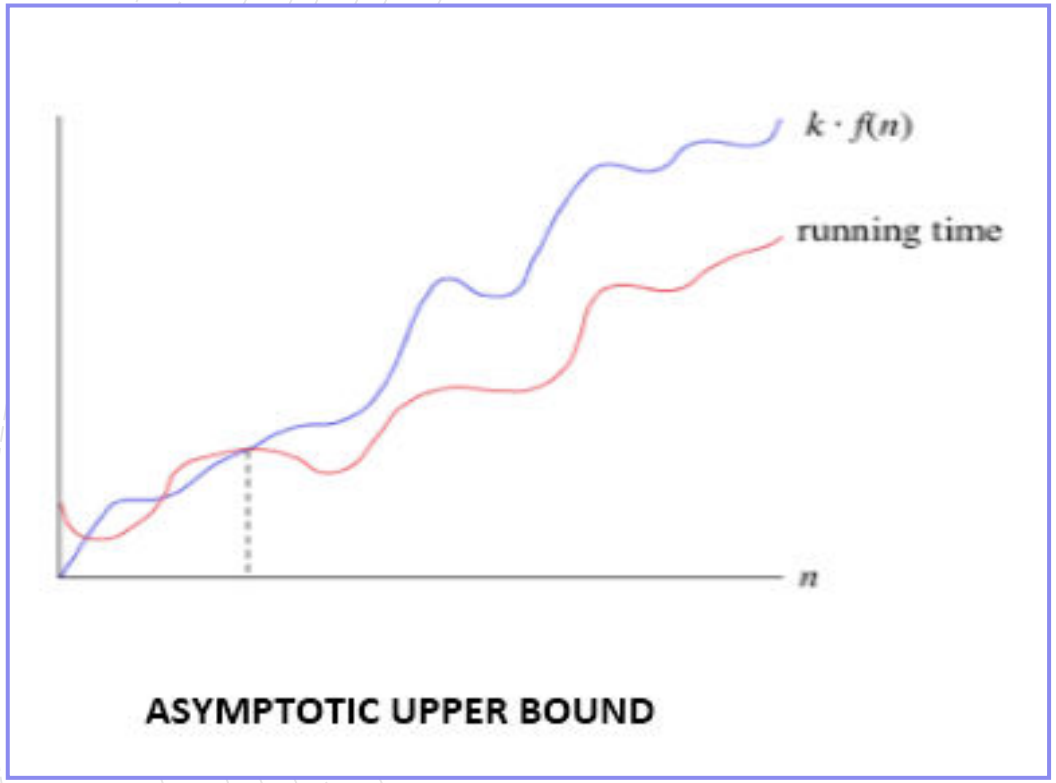
Three notations are used to calculate the running time complexity of an algorithm:

1. Big-oh (O) notation -Big O notation specifically describes worst case scenario.

2. Omega (Ω) Notation -Omega(Ω) notation specifically describes best case scenario.

3. Theta (θ) -This notation represents the average complexity of an algorithm.

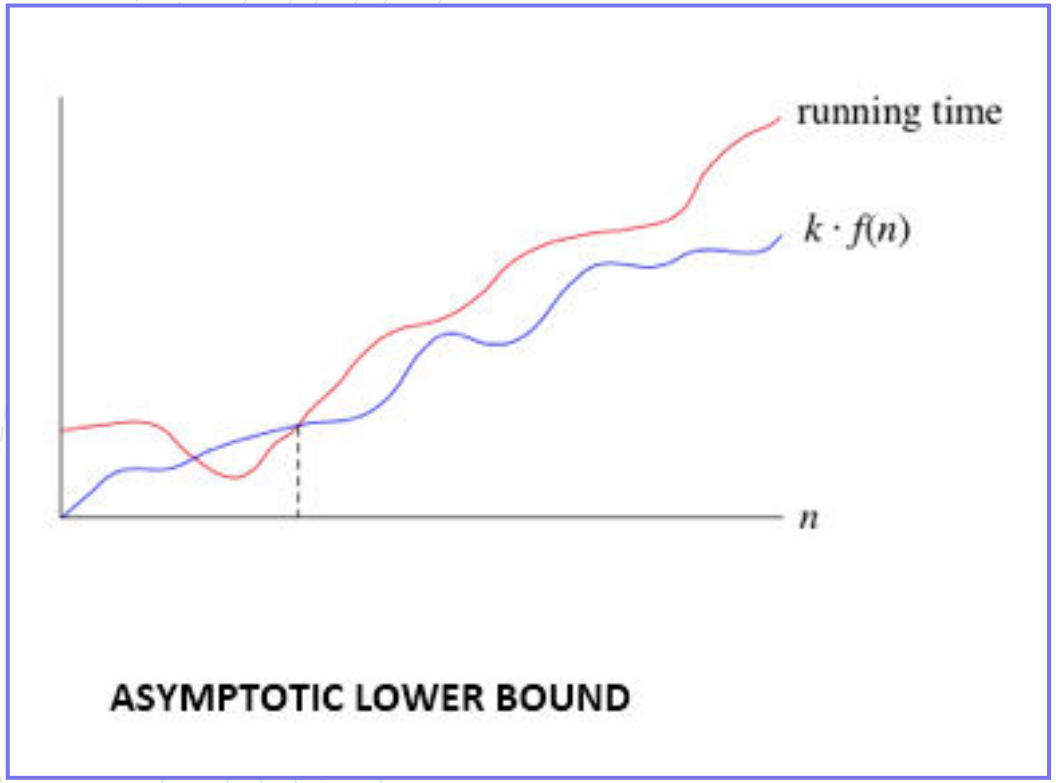
Big O Notation



Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity.

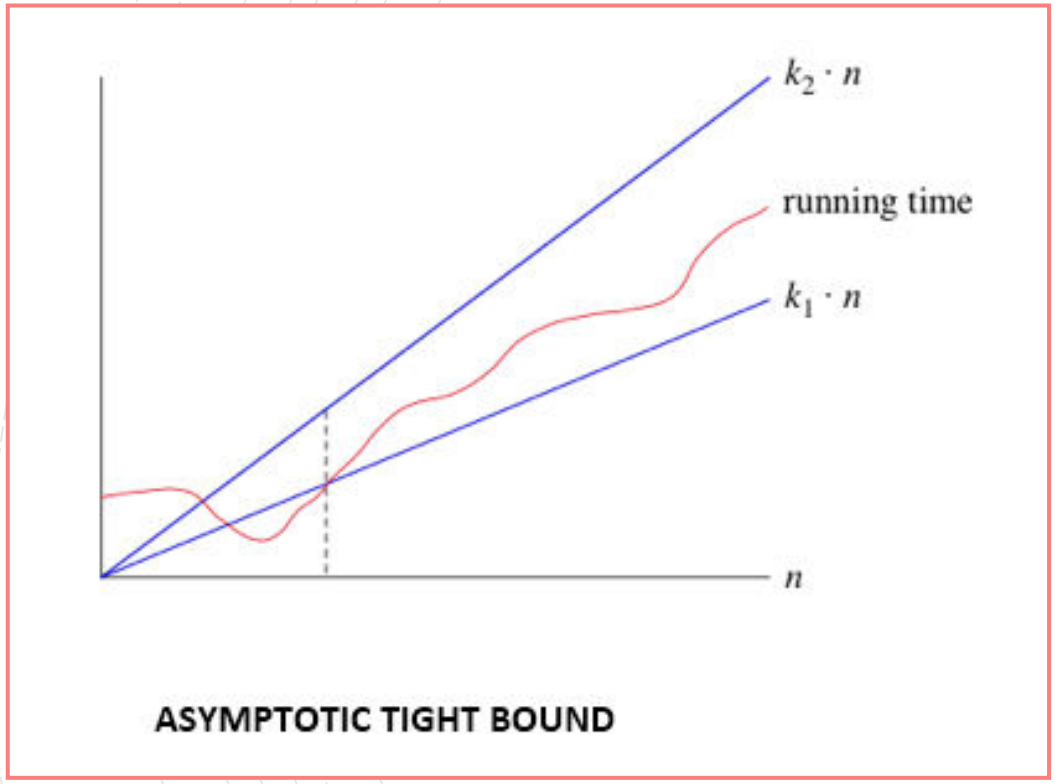
- Big O notation specifically describes worst case scenario.
- It represents the upper bound running time complexity of an algorithm.
- It is the measure of the longest amount of time..
- The function **f (n) = O (g (n))**
- **Such that f(n) <= C g(n)**
- e.g. f(n) =2n+3

Omega Notation(Ω)



- The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time.
- It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.
- The function f (n) = Ω (g (n)) .

Theta (θ)



ASYMPTOTIC TIGHT BOUND

- This notation describes both upper bound and lower bound of an algorithm.so it defines exact asymptotic behavior.
- The function f (n) = θ (g (n))

Complexity of algorithm

The Complexity of an **algorithm** is a measure of the amount of time and/or space required by an **algorithm** for an input of a given size (n).

The complexity function f(n) for certain cases are:

1. Best Case : The minimum possible value of f(n) is called the best case.

2. Average Case : The expected value of f(n).

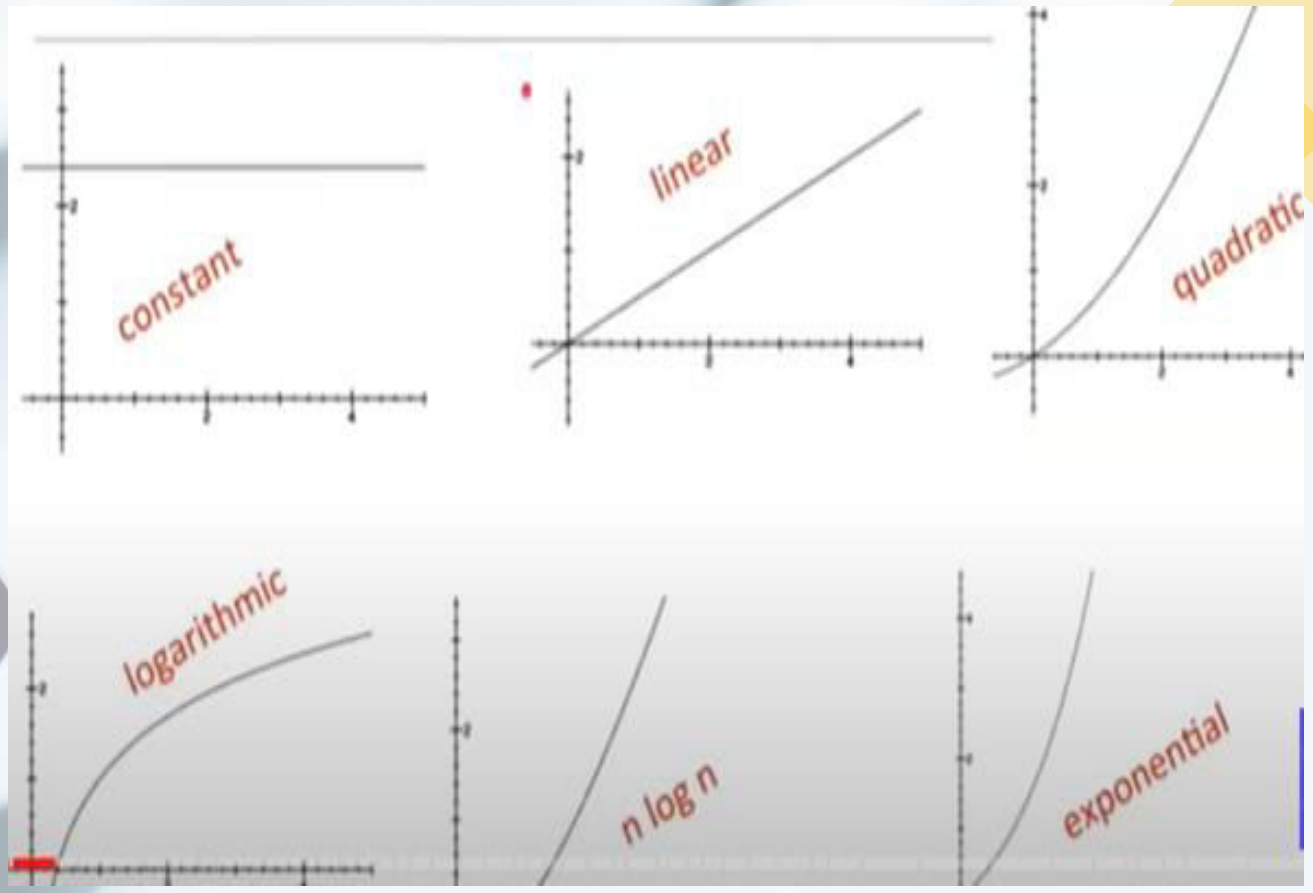
3. Worst Case : The maximum value of f(n) for any key possible input.

Analysis of Algorithm

- **1. Worst Case Analysis (Mostly used)**
- In the worst-case analysis, we calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed.
- **2. Best Case Analysis (Very Rarely used)**
- In the best case analysis, we calculate the lower bound on the running time of an algorithm. We must know the case that causes a minimum number of operations to be executed.
- **3. Average Case Analysis (Rarely used)**
- In average case analysis, we take all possible inputs and calculate the computing time for all of the inputs.

Category of algorithms

1. Constant time O(1)
2. Linear time O(n)
3. Logarithmic time O(log n)
4. Polynomial time O(n^k) where K>1 algorithm
5. Exponential time O(k^n) where K>1 algorithm

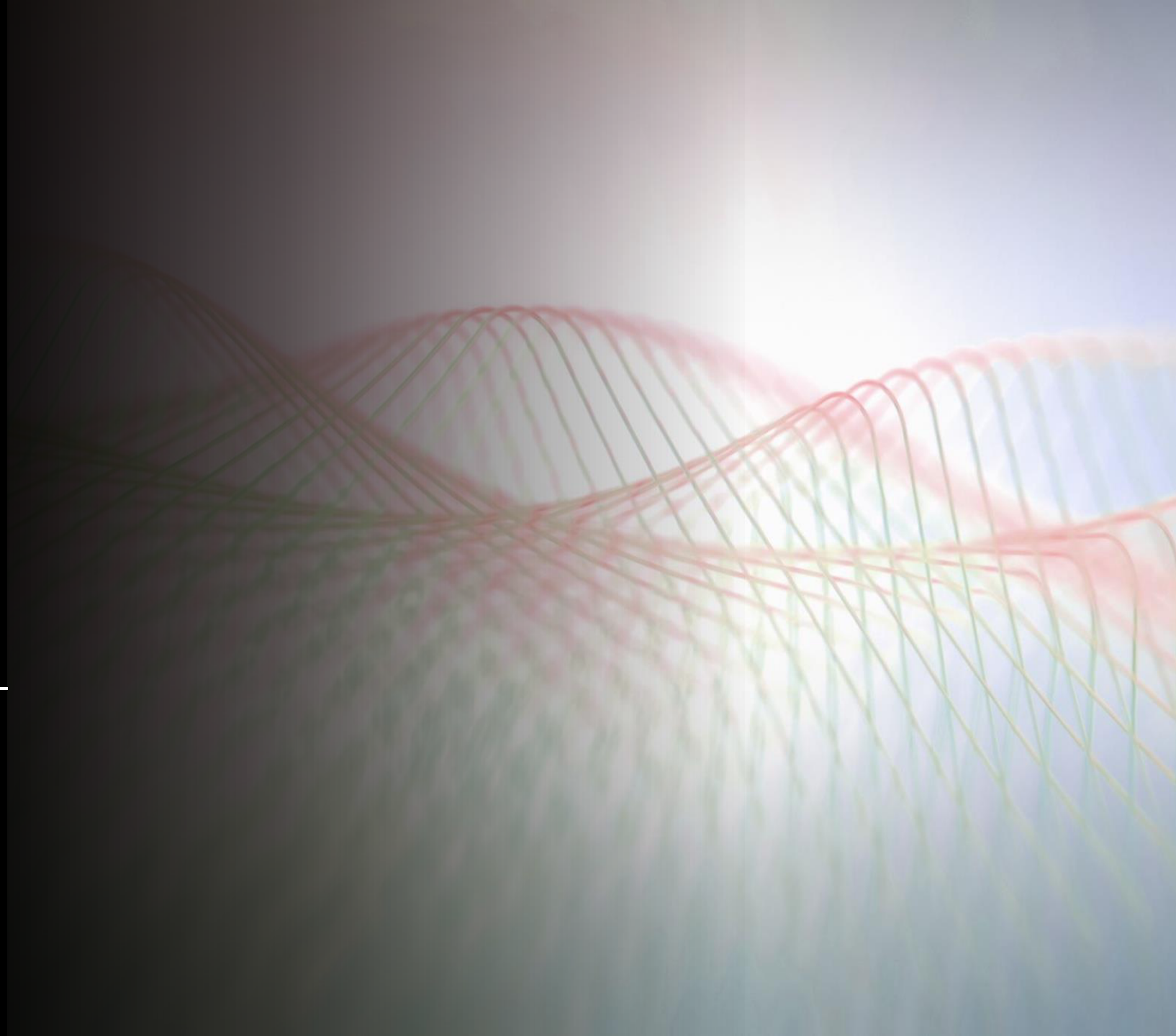


Any Question

Data Structure with Python

Mr. V.M.Vasava

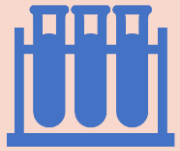
GPG,IT Dept.



Agenda

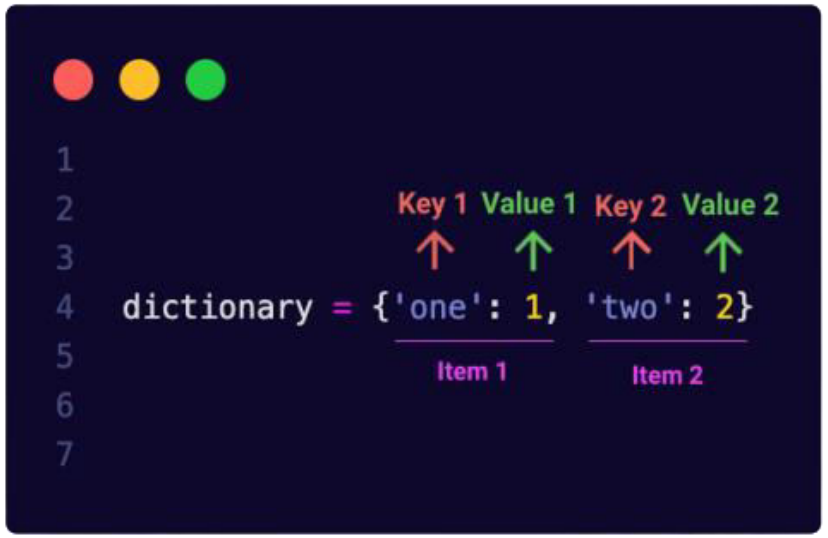
Dictionary & set

Basic Operations



Methods

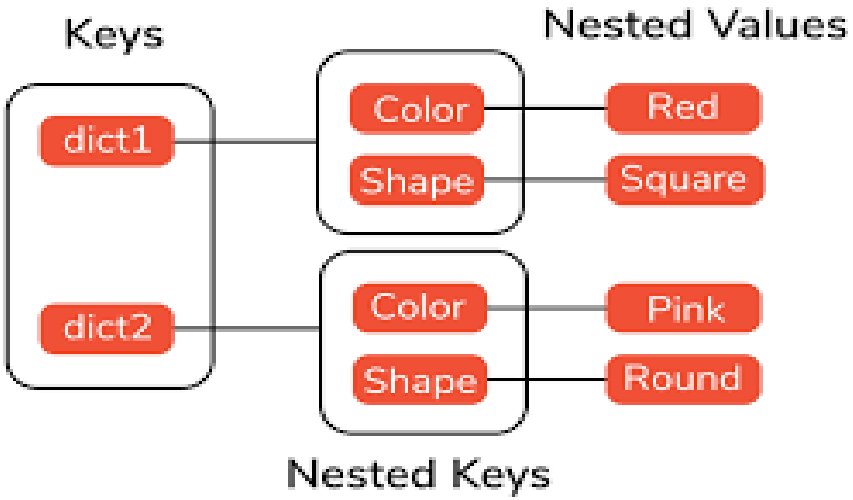
Dictionary




```
In [37]: a={1:"VMV",2:"Ashwini",3:"Nivedita"}

In [38]: a
Out[38]: {1: 'VMV', 2: 'Ashwini', 3: 'Nivedita'}

In [39]: b=dict({1: 'Java', 2: 'DS', 3:'DM'})

In [40]: b
Out[40]: {1: 'Java', 2: 'DS', 3: 'DM'}
```

- Python dictionary is an unordered collection of items. Each item of a dictionary has a **key/value** pair.
- Dictionaries are optimized to retrieve values when the key is known..
- With the curly bracket({}) and built-in dict() class, create empty dictionaries data(key-value pairs) with separated commas.
- Keys will be a single element
- Values can be a list or list within a list, numbers, etc.
- It is a **mutable data structure**.

Nested Dictionary



1. A dictionary can contain dictionaries, this is called nested dictionaries.
2. **Syntax:**

```
nested_dict = { 'dictA': {'key_1': 'value_1'},
                'dictB': {'key_2': 'value_2'}}
```

```
In [41]: box = {1: {'color': 'red', 'shape': 'square'},
    ...:        2: {'color': 'pink', 'shape': 'round'}}

In [42]: print(box)
{1: {'color': 'red', 'shape': 'square'}, 2: {'color': 'pink', 'shape':
'round'}}
```

Dictionary Operations

Definition operations

{ }	Creates an empty dictionary

Mutable operations

[ ]	Adds a new pair of key and value to the dictionary, in case that the key already exists, updates the value
del	Removes an entry from a dictionary
update	Combines the entries of two dictionaries

Inmutable operations

len	Returns the number of entries in a dictionary
keys	Returns a view of all keys in a dictionary
values	Returns all the values stored in a dictionary.
items	Returns all the keys and their associated values as a sequence of tuples.
in	Tests whether a key exists in a dictionary
get	Returns the value of a key or a configurable default
setdefault	Returns the value if the key is in dictionary; otherwise, sets the value for the key to the default and returns the value

Create Dictionary

1. { }

- Creates an empty dictionary or a dictionary with some initial values.

```
In [1]: y = {}

In [2]: y
Out[2]: {}

In [3]: x = {1: "one", 2: "two", 3: "three"}

In [4]: x
Out[4]: {1: 'one', 2: 'two', 3: 'three'}
```

2. Mutable operations

- These operations allow us to work with dictionaries, but altering or modifying their previous definition.
- A). **[ ]** -Adds a new pair of key and value to the dictionary

```
In [5]: y['one'] = 1

In [6]: y
Out[6]: {'one': 1}

In [7]: y['two'] = 2

In [8]: y
Out[8]: {'one': 1, 'two': 2}
```

Dictionary Operations

b). del

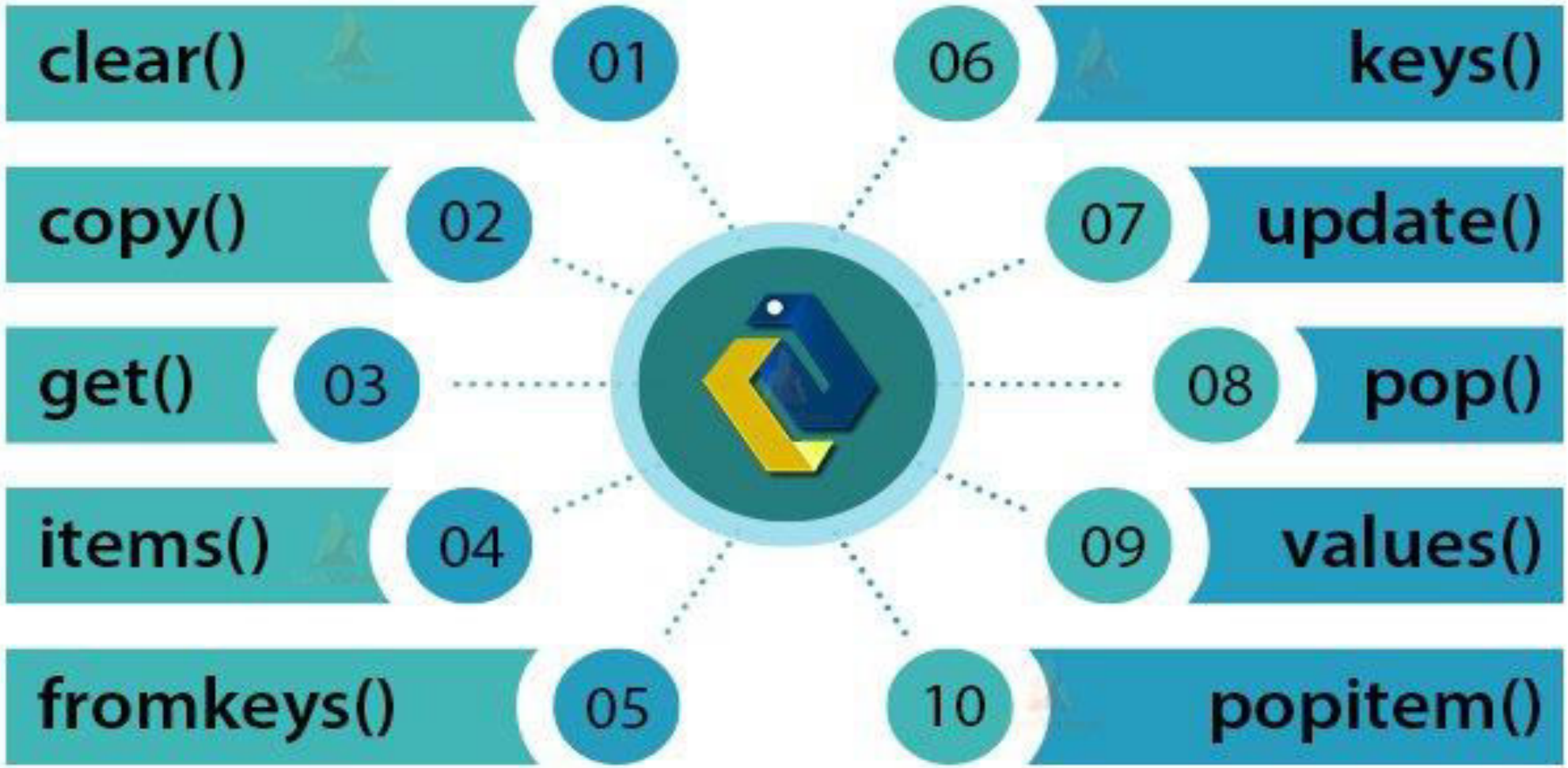
- Del statement can be used to remove an entry (key-value pair) from a dictionary.

c).update

- This method updates a first dictionary with all the key-value pairs of a second dictionary.

```
In [8]: y
Out[8]: {'one': 1, 'two': 2}

In [9]: del y['two']

In [10]: y
Out[10]: {'one': 1}

In [11]: x = {'one': 0, 'two': 2}

In [12]: x.update(y)

In [13]: x
Out[13]: {'one': 1, 'two': 2}
```

Python Dictionary Methods



Methods	Description
clear()	The clear method removes all the entries in the dictionary.
copy()	Returns a shallow copy of the dictionary
fromkeys()	Creates a dictionary from the given sequence
get()	Returns the value for the given key
items()	Return the list with all dictionary keys with values
keys()	Returns a view object that displays a list of all the keys in the dictionary in order of insertion
pop()	Returns and removes the element with the given key
popitem()	Returns and removes the key-value pair from the dictionary
update()	Updates the dictionary with the elements from another dictionary
values()	Returns a list of all the values available in a given dictionary

Example

- **get() Method** return the value for the given key if present in the dictionary.

Syntax : **Dict.get(key, default=None)**

✓ key: The key name of the item you want to return the value from

✓ Value: (Optional) Value to be returned if the key is not found. The default value is None.

```
In [12]: a={'A':10,'B':20,'C':30}

In [13]: a['B']
Out[13]: 20

In [14]: a.get('B')
Out[14]: 20

In [15]: a.get('D')
```

```
In [42]: a.get('D',"not found")
Out[42]: 'not found'

In [43]: dept = {'IT' : {'is' : 'best'}}

In [44]: res = dept.get('IT', {}).get('is')

In [45]: res
Out[45]: 'best'
```

Example

- Syntax: **dict.clear()**
- **The clear() method doesn't return any value.**
- **Syntax: dict.copy()**
- *This method doesn't modify the original, dictionary just returns copy of the dictionary.*

```
In [19]: gp={'A':10,'B':9,'C':7}

In [20]: type(gp)
Out[20]: dict

In [21]: gp.clear()

In [22]: gp
Out[22]: {}
```

```
In [23]: gp={'A':10,'B':9,'C':7}

In [24]: gp
Out[24]: {'A': 10, 'B': 9, 'C': 7}

In [25]: a=gp.copy()

In [26]: a
Out[26]: {'A': 10, 'B': 9, 'C': 7}
```

Example

- **fromkeys() :**
- This method is used to create a new dictionary from a sequence containing all the keys and common value, which will be assigned to all the kerys.
- Syntax : **fromkeys(seq, val)**

seq : The sequence to be transformed into a dictionary.

val : Initial values that need to be assigned to the generated keys. Defaults to None.

```
In [28]: seq = {'a', 'b', 'c'}

In [29]: d=dict.fromkeys(seq)

In [30]: d
Out[30]: {'c': None, 'a': None, 'b': None}

In [31]: d1=dict.fromkeys(seq, 1)

In [32]: d1
Out[32]: {'c': 1, 'a': 1, 'b': 1}

In [33]: l=[1,2]

In [34]: d2=dict.fromkeys(seq, l)

In [35]: d2
Out[35]: {'c': [1, 2], 'a': [1, 2], 'b': [1, 2]}
```

```
In [2]: d1=dict.fromkeys("vilas",615)

In [3]: d1
Out[3]: {'v': 615, 'i': 615, 'l': 615, 'a': 615, 's': 615}

In [4]: d=dict.fromkeys((1,2,3))

In [5]: d
Out[5]: {1: None, 2: None, 3: None}
```

Example

- **items()** method is used to return the list with all dictionary keys with values.
- Syntax: **dictionary.items()**
- Parameters: This method takes no parameters.
- **Returns:** A view object that displays a list of a given dictionary's (key, value) tuple pair.

```
In [50]: a
Out[50]: {'A': 10, 'B': 9, 'C': 7}

In [51]: a.items()
Out[51]: dict_items([('A', 10), ('B', 9), ('C', 7)])

In [52]: del[a['C']]

In [53]: a
Out[53]: {'A': 10, 'B': 9}

In [54]:
```

Example

- **key()** –This method returns all the keys of the dictionary.
- Syntax: **dict.keys()**
- **update() method** updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.
- Syntax: **dict.update([other])**

```
In [6]: std={'Name':'vmv','sal':2000,'Dept':'IT'}

In [7]: brn={'Name':'vmv','Desg':'Lect'}

In [8]: std.update(brn)

In [9]: std
Out[9]: {'Name': 'vmv', 'sal': 2000, 'Dept': 'IT', 'Desg': 'Lect'}
```

```
In [75]: a
Out[75]: {'A': 10, 'B': 20, 'C': 30}

In [76]: a.keys()
Out[76]: dict_keys(['A', 'B', 'C'])

In [77]: for i in a:
    ...:     if (j == 1):
    ...:         print('2nd key using loop : ' + i)
    ...:

In [78]: print(i)
C
```

```
In [80]: a
Out[80]: {'A': 10, 'B': 20, 'C': 30}

In [81]: b={'D':40,'E':50}

In [82]: a.update(b)

In [83]: a
Out[83]: {'A': 10, 'B': 20, 'C': 30, 'D': 40, 'E': 50}

In [84]:
```

Example

Values()

- The values() method returns all the values from the dictionary.
- Syntax: **dictionary_name.values()**

setdefault() returns the value of a key (if the key is in dictionary). Else, it inserts a key with the default value to the dictionary.

- **Syntax:**
- **dict.setdefault(key, default_value)**

key – Key to be searched in the dictionary.

default_value (optional) – Key with a value default_value is inserted to the dictionary if key is not in the dictionary.

If not provided, the default_value will be None.

```
In [85]: marks = {'Physics':67, 'Maths':87}

In [86]: print(marks.values())
dict_values([67, 87])

In [87]: marks.setdefault(' ', 32)
Out[87]: 32

In [88]: print(marks)
{'Physics': 67, 'Maths': 87, ' ': 32}

In [89]:
```

Example

- 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.

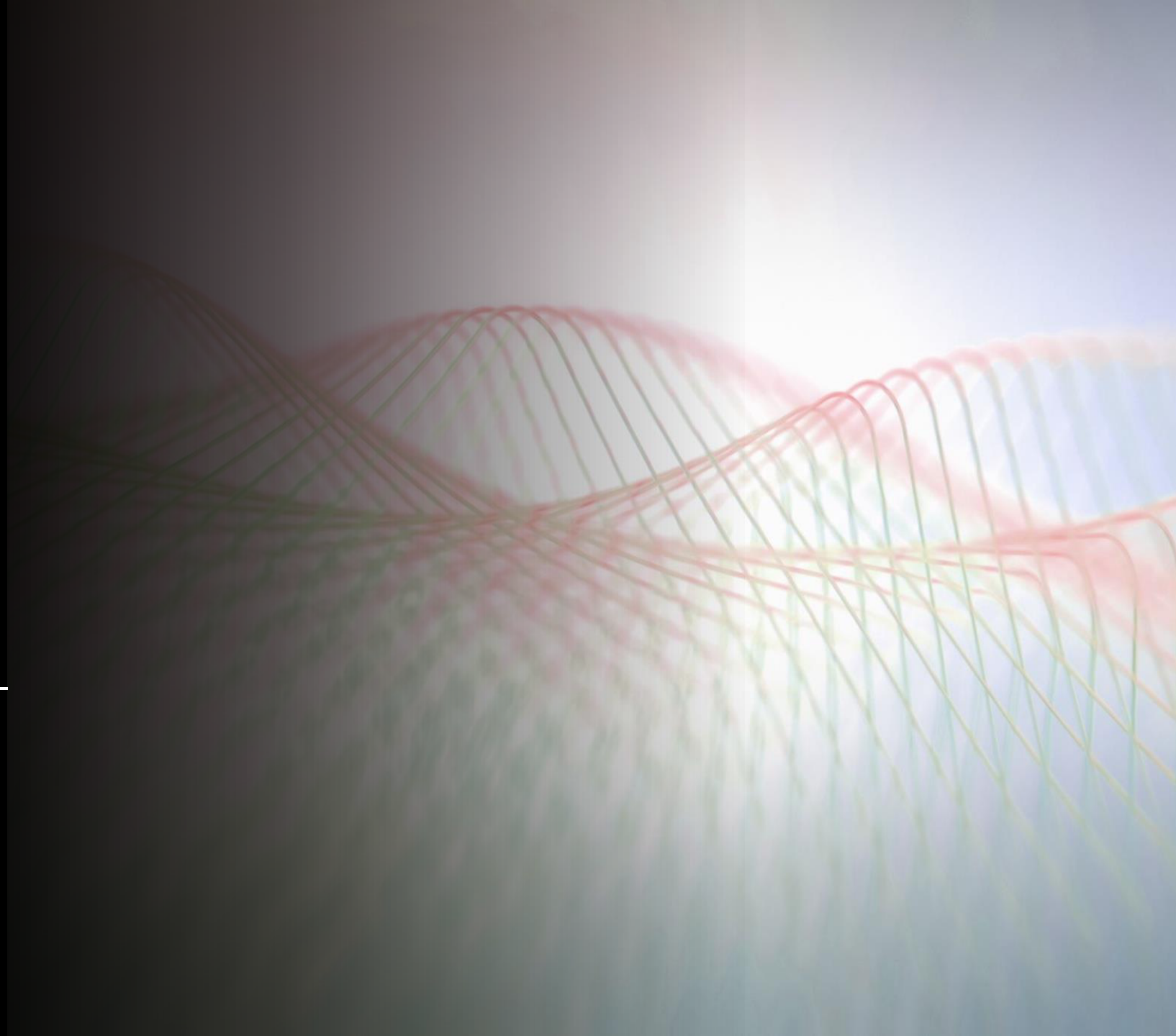
```
In [16]: squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

In [17]: print(squares.pop(4))
16

In [18]: print(squares.popitem())
(5, 25)

In [19]:
```

Any Questions ???

Data Structure with Python

Mr. V.M.Vasava

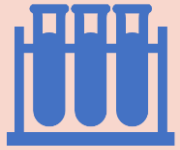
GPG,IT Dept.



Agenda

List & Tuple

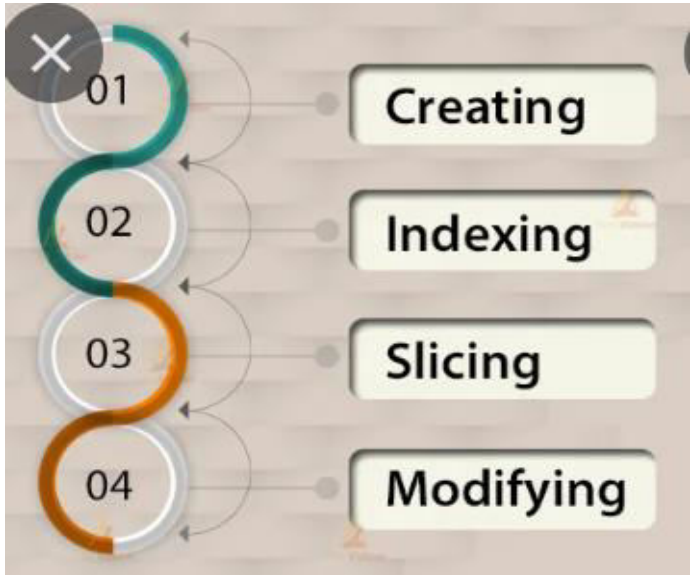
Basic Operation of List & Tuple



Methods of List & Tuple

List in python

List



- A list can contain a series of values.
- List variables are declared by using brackets **[ ]**.
- A list is **mutable**, which means we can modify the list.
- Lists are used to store multiple items in a single variable.
- For example:

L = [20, 'Jessa', 35.75, [30, 60, 90]]

L[0] L[1] L[2] L[3]

- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Changeable**: List is mutable and we can modify items.
- ✓ **Heterogeneous**: List can contain data of different types
- ✓ **Contains duplicate**: Allows duplicates data

Creating List

- Lists are create with two ways:

1. square brackets []
2. list() function.

z =	[3,	7,	4,	2]
index	0	1	2	3
negative index	-4	-3	-2	-1

- Returned **a List** created from a passed argument as a sequence type(string,list,tuple)

```
In [19]: z=[]    #create empty list

In [20]: z=[3,7,4,2]  #create list with values

In [21]: print(z[0]) #Access list with positive index
3

In [22]: print(z[-1]) #Access list with negative index
2

In [23]: z[2]="vmv" #update list with value

In [24]: print(z)
[3, 7, 'vmv', 2]

In [25]: z=list() #create list with list() function

In [26]: z=[[1,"vmv",20.50],[2,"PAP",40.50]]
```

```
In [2]: list("Hello")
Out[2]: ['H', 'e', 'l', 'l', 'o']

In [3]: list((1,2,3))
Out[3]: [1, 2, 3]

In [4]: list({1:"one",2:"two"})
Out[4]: [1, 2]
```

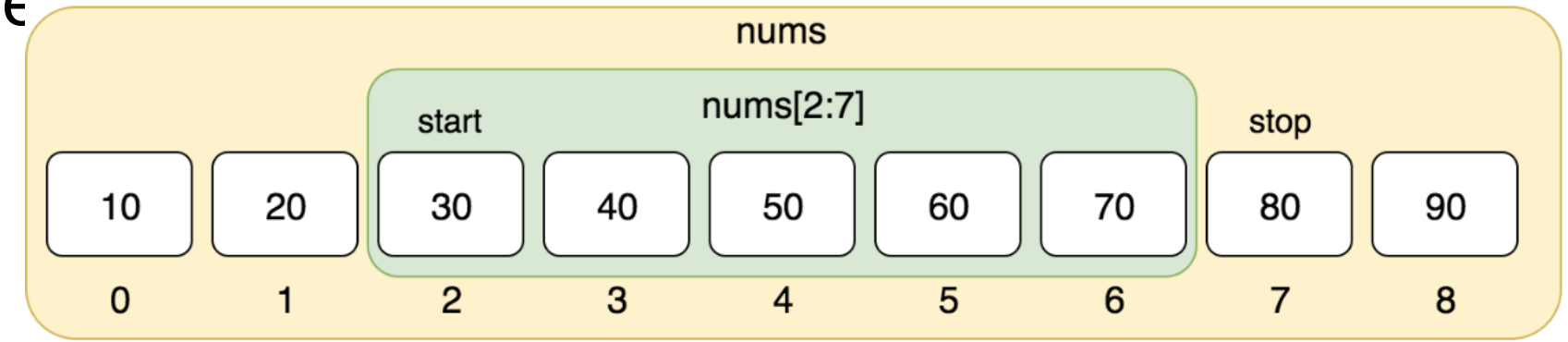
Slice in List

- The format for list slicing is **[start:stop:step]**.

  start is the index of the list where slicing starts.

  stop is the index of the list where slicing ends.

  step allows you to select **nth** item within the range **start** to **stop**




```
In [5]: nums=[10,20,30,40,50,60,70,80,90]

In [6]: nums[2:7]
Out[6]: [30, 40, 50, 60, 70]
```

Slice in List

- **Get all the Items**

```
In [3]: l=[1,2,3,4,5]

In [4]: l[:]
Out[4]: [1, 2, 3, 4, 5]
```

- **Get all the Items After a Specific Position**

```
In [6]: l
Out[6]: [1, 2, 3, 4, 5]

In [7]: l[2:]
Out[7]: [3, 4, 5]
```

- **Get all the Items Before a Specific Posi**

```
In [8]: l
Out[8]: [1, 2, 3, 4, 5]

In [9]: l[:2]
Out[9]: [1, 2]
```

- **Get the Items at Specified Inter**

```
In [10]: l
Out[10]: [1, 2, 3, 4, 5]

In [11]: l[::2]
Out[11]: [1, 3, 5]
```

Slice , concatenate, Repetition two list

- Taking n last elements of a list

-Negative indexes allow us to easily take n-last elements of a list:

-mix negative and positive indexes in start and stop positions

```
In [12]: nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]

In [13]: nums[-3:]
Out[13]: [70, 80, 90]

In [14]: nums[1:-1]
Out[14]: [20, 30, 40, 50, 60, 70, 80]

In [15]: nums[-3:8]
Out[15]: [70, 80]

In [16]: nums[-5:-1]
Out[16]: [50, 60, 70, 80]
```

```
>>> a=[1,2,3]
>>> b=["vmv","IT"]
>>> a+b
[1, 2, 3, 'vmv', 'IT']
>>> print(a*3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

List Methods

Methods	Description
append()	adds an element to the end of the list
extend()	adds all elements of a list to another list
insert()	inserts an item at the defined index
remove()	removes an item from the list
pop()	returns and removes an element at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the number of items passed as an argument
reverse()	reverse the order of items in the list

Append, Extend

Append
- ✓ Adds an item to the end of the list.
- ✓ Does not return any value.
- ✓ Modifies the original list.

Extend –Adding multiple elements at the end.

append	extend
The append method will add an item to the end of the list.	The extend method will extend the list by appending all the items from the iterable.
The length of the list will be increased by 1.	The length of the list will be increased depends on the length of the iterable.
It will update the original list itself.	It will update the original list itself.
Return type is None.	Return type is None.

```
In [7]: L=[1,2,3]

In [8]: L.append(4)

In [9]: L
Out[9]: [1, 2, 3, 4]

In [10]: L1=L.append(5)
```

```
In [21]: a1 = [1, 2]

In [22]: b = [3, 4]

In [23]: a1.extend(b)

In [24]: print(a1)
[1, 2, 3, 4]

In [25]: a2.append(b)
    ...: print(a2)
[1, 2, [3, 4]]

In [26]:
```

Insert Method

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

Syntax: list.insert(i, elem)

- Here, elem is inserted to the list at the ith index. All the elements after elem are shifted to right.
- index - the index where the element needs to be inserted
- element - this is the element to be inserted in the list

```
In [13]: L
Out[13]: [1, 2, 3, 4, 5]

In [14]: L.insert(2,30)

In [15]: L
Out[15]: [1, 2, 30, 3, 4, 5]

In [16]: L.insert(-10,50)
```

List functions

1. Index Method :The index method returns the first index at which a value occurs.
2. If given element not in the list ,generates Value Error.

Syntax: *list_name.index(element, start, end)*

- ***element*** *– The element whose lowest index will be returned.*
- ***start*** *(Optional) – The position from where the search begins.*
- ***end*** *(Optional) – The position from where the search ends.*

```
In [7]: z
Out[7]: [3, 7, 4, 2]

In [8]: print(z.index(2))
3
```

```
In [18]: z
Out[18]: [3, 7, 4, 5, 6, 7, 8]

In [19]: print(z.index(7,2,6))
5
```

```
In [7]: z
Out[7]: [3, 7, 4, 2]

In [8]: print(z.index(2))
3

In [9]: print(z.index(4,2))
2
```

List Method

Count Method

- It counts the number of times a value occurs in a list
- ***Syntax:*** *list_name.count(object)*
- ***Parameters: object:*** *is the item whose count is to be returned.*
- *If the given item is not in the list, it returns 0.*

```
In [8]: vowels = ['a', 'e', 'i', 'o', 'i', 'u']

In [9]: count = vowels.count('i')

In [10]: print('The count of i is:', count)
The count of i is: 2
```

```
In [5]: random = ['a', ('a', 'b'), ('a', 'b'), [3, 4]]

In [6]: count = random.count(('a', 'b'))

In [7]: print("The count of ('a', 'b') is:", count)
The count of ('a', 'b') is: 2

In [8]:
```

Pop Vs Remove

The pop() method removes the item at the given index from the list and returns the removed item.

list.pop(index)

The remove() method removes the first matching element (which is passed as an argument) from the list. But doesn't returned the deleted element.

list.remove(element)

remove()	pop()
To remove based on the element.	To remove based on index
It doesn't return any value	It returns the popped out element.
If the item is not present, the we get ValueError	If the index is not in the range, then we get IndexError

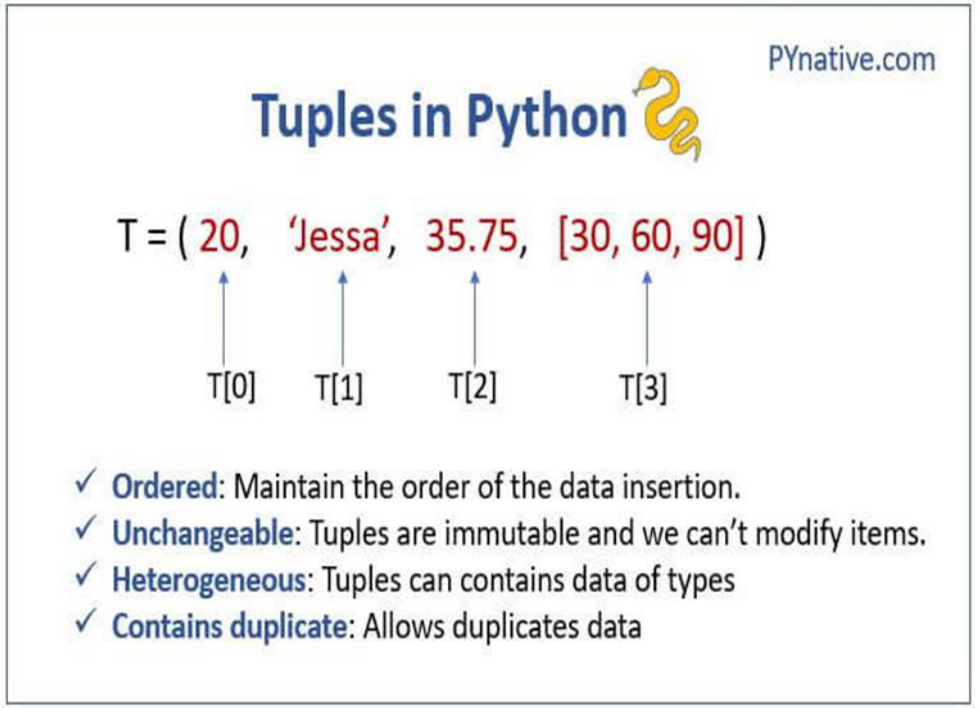
```
In [37]: a1
Out[37]: [1, 2, 4]

In [38]: a1.remove(4)

In [39]: a1
Out[39]: [1, 2]

In [40]: a1.pop()
Out[40]: 2
```

Tuples in python

- **Tuple**

Tuple is an ordered collection of elements enclosed within ().

Tuple



- Tuples are ordered collections of elements that are unchangeable.

The data stored in a tuple are heterogeneous.

- Data in a tuple are stored with comma-separated and is enclosed in a bracket **( )**.
- A Tuple is **immutable**, which means we can't modify the tuple.
- For example:

```
In [7]: T=(2,5,4.5,'Nidhi')

In [8]: T
Out[8]: (2, 5, 4.5, 'Nidhi')

In [9]: T[0]=10
Traceback (most recent call last):

  Cell In[9], line 1
    T[0]=10

TypeError: 'tuple' object does not support item assignment


In [10]: tup1=(2,2,3.5,"vmv")
```

Create tuple in python

()

- (1,2,3) --Multiple values
- (1,) --Single Value
- () --Empty tuple
- T=1,2,3

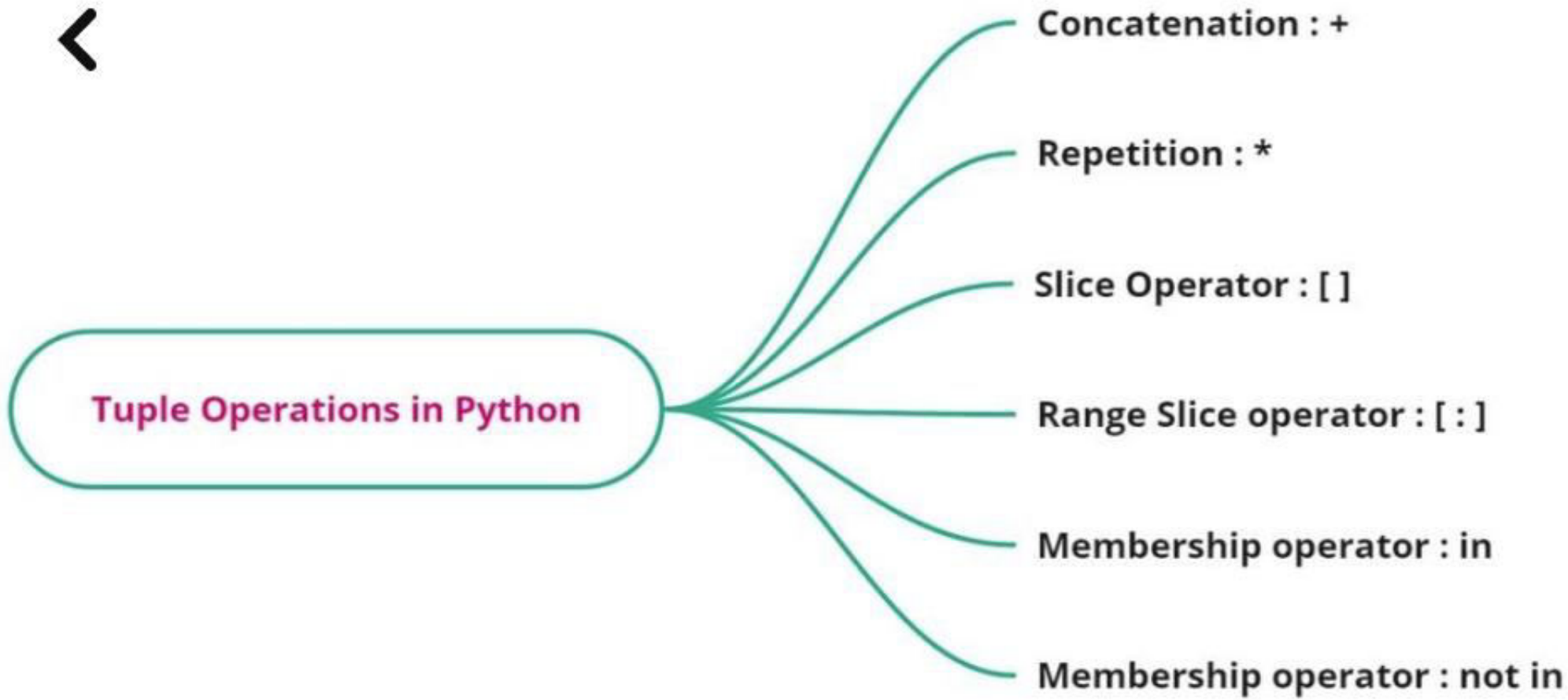
Tuple()

- tuple()
- **tuple([1,])**

Example

```
In [17]: t=()

In [18]: print(t,type(t))
() <class 'tuple'>

In [19]: t=(1,2.0,"vmv",True)

In [20]: print(t,type(t))
(1, 2.0, 'vmv', True) <class 'tuple'>

In [21]: t=(1,)   #single tuple

In [22]: print(t,type(t))
(1,) <class 'tuple'>

In [23]: a=tuple()    #Empty tuple

In [24]: print(t,type(t))
(1,) <class 'tuple'>
```

Tuple Operation



Operations

1. Tuple Concatenation

We can concatenate the tuples using the ‘+’ operator.

2.Tuple Repetition

- Tuple repetition means repeating the elements of the tuples multiple times. This can be achieved using the ‘*’ operator.

```
In [22]: t1=(1,2,3)

In [23]: t2=(3,4,5)

In [24]: print("concate",t1+t2)
concate (1, 2, 3, 3, 4, 5)

In [25]: print("repetition",t1*3)
repetition (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

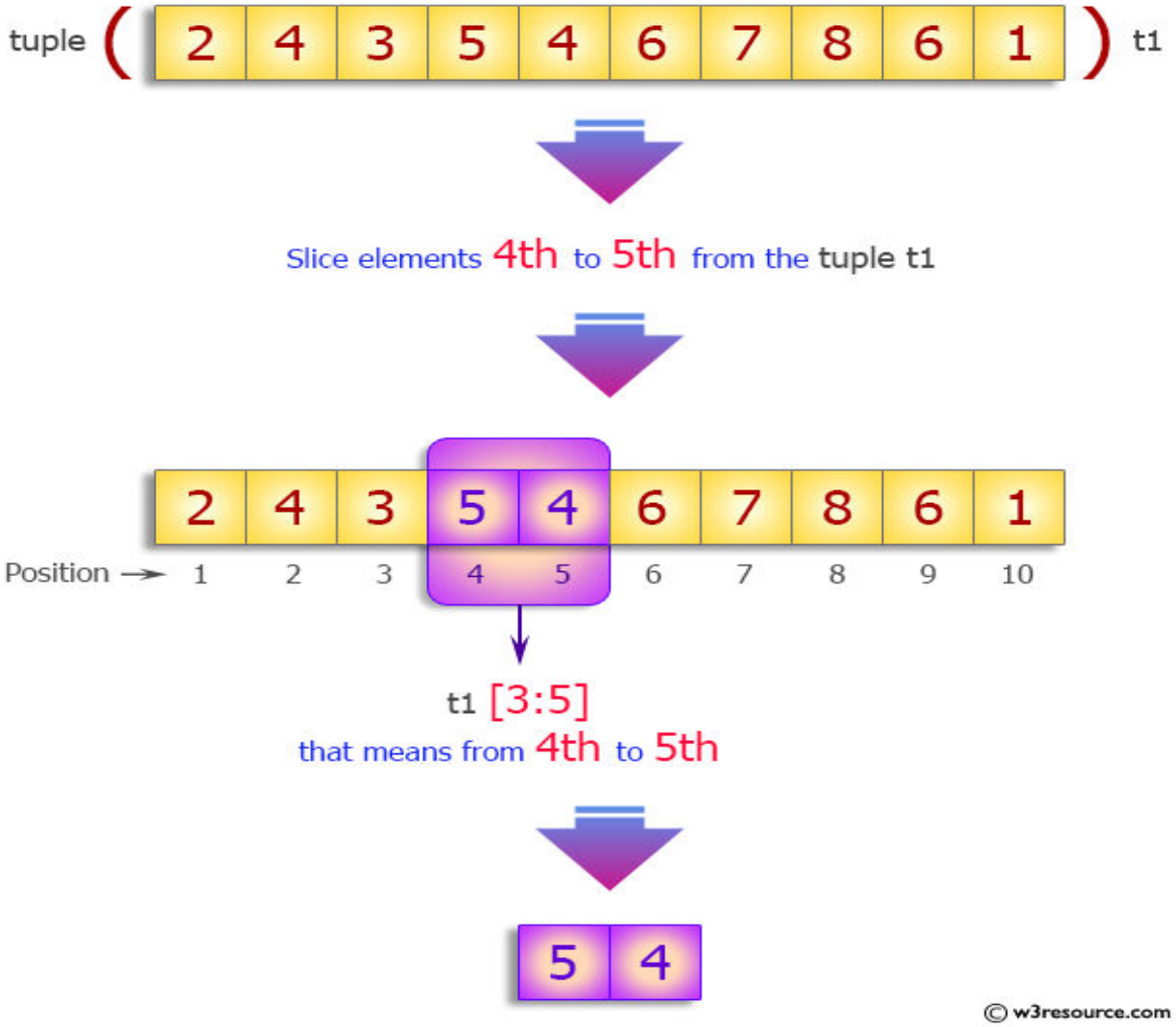
Operations of Tuple

3.Membership Operator

- Using the ‘in’ operator, we can check if a particular element is present in the Tuple.
- It returns the Boolean value True if the element is present in the tuple and returns False if the element is not present.

```
In [31]: t1=(1,2,3)

In [32]: print(2 in t1)
True

In [33]: print(4 not in t1)
True
```

Slice in Tuple



- Using the slice operator (:), we can access a range of elements from the tuple.

```
In [14]: t1=(2,4,3,5,4,6,7,8,6,1)

In [15]: t1[3:5]
Out[15]: (5, 4)

In [16]: t1[:-3]
Out[16]: (2, 4, 3, 5, 4, 6, 7)

In [17]:
```

Tuple Methods

Methods	Description
count()	returns count of the element in the tuple
index()	returns the index of the element in the tuple
any()	Returns True if any element present in a tuple and returns False if the tuple is empty
min()	Returns smallest element (Integer) of the Tuple
max()	Returns largest element (Integer) of the Tuple
len()	Returns the length of the Tuple
sorted()	Used to sort all the elements of the Tuple
sum()	Returns sum of all elements (Integers) of the Tuples

Example

```
In [32]: t1
Out[32]: (1, 2, 1, 2)

In [33]: t1.count(2)
Out[33]: 2

In [34]: t1.index(2)
Out[34]: 1

In [35]: sum(t1)
Out[35]: 6

In [36]: min(t1)
Out[36]: 1

In [37]:
```

Any Questions ???