* Data structure :-

A data structure is a way of organizing, storing and managing data so that we can use it efficiently

in programing, data structures help us:

- store data properly
- Access it quickly
- modify or update it easily.

python provides both built in data structures and allows us to imple -ment user-defined data structure.

Built in data structures:-

These are provided directly by python.

- List : ordered, mutable collection of elements.

  ex:- num = [1,2,3,4]

- Tuple : ordered, immutable collection of elements.

  ex:- coordinates = (10,20)

- set : un ordered collection of unique elements.

  ex:- fruits = {"apple", "banana", "cherry"}

- Dictionary : collection of key-value pairs.

  ex:- stu = {"name":"John", "age":21}.

* List in python:-

A list is a multivalued variable that can store multiple items in a single variable.

it's heterogneous (can store different data types) and mutable (can be changed after creation).

mutable → you can change, add or remove elements after creating the list

ordered → the element have a fixed ordered (index-based)

Allow duplicates :- list can contain duplicate values.

Heterogeneous — can hold different types of data (int, float, string etc)

* syntax :- 1. using square brackets

```
my_list = [10,20,30,40]
```

2. using the list() constructor.

```
My_list = list([10,20,30,40])
```

ex:-
```
details = [101, "Ravi", 89.5, True]
print (details).
```

o/p :- [101, 'Ravi', 89.5, True]

1. Creating a list :-

you create a list by placing values inside square brackets [], seperated by commas.

ex:-
```
fruits = ["apple", "banana", "cherry"]
print (fruits)
```

o/p :- ['apple', 'banana', 'cherry']

- fruits → list name (variable)
- ["apple", 'banana', 'cherry'] → list elements.
- square brackets [] tell python it's a list.

* List can Hold diff data type:-

list can store number, strings, boolean or even a mix of all.

```
num = [10, 20, 30]          # integers
names = ["Alice", "Bob", "charlie"]   # strings
mixed = [10, "Hello", 3.5, True]      # mixed data types.
```

o/p :-
```
[10, 20, 30]
['Alice', 'Bob', 'charlie']
[10, 'Hello', 3.5, True].
```

* Creating an empty list :-

If you want a list with no elements yet :-

```
empty list = []
print (empty_list)
```

o/p :- []

you can add elements later using .append() :-

```
empty_list = []
empty_list.append("apple")
print(empty_list)
```

o/p :- ['apple']

* Creating a list from Another Type.

you can convert other data (like string or tuples) into a list using the list() function.

From string :-

```
word = "hello"
letters = list(word)
print(letters)
```

o/p :- ['h', 'e', 'l', 'l', 'o']

From tuple :-

```
t = (1,2,3)
number = list(t)
print(numbers)
```

o/p :- [1,2,3]

* Nested list (list inside a list)

you can also create a list inside another list

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]
```

o/p :- [[1,2,3],[4,5,6],[7,8,9]]

* Accessing an inner element :

```
print(matrix[0][1])   # 2
```

* creating a list using Range

you can create a list of numbers quickly using range() and list()

```
numbers = list(range(1,6))
print(num)
```

o/p :- [1,2,3,4,5]

* square brackets — fruits = ["apple", "banana"] — most common way
* Empty list — list1 = [] — creates blank list
* list func — list("abc") — converts to list
* Nested list — [[1,2,3],[3,4]] — list inside another list
* range() — list(range(1,5)) — number in range.

Ex:-
2. Accessing elements (using index)

every item in a list has a position num (index)

python start - 0 indexing.

```
fruits = ["apple", "banana", "cherry"]

print (fruits[0])
print (fruits[1])
print (fruits[2])
```

o/p :- apple
banana
cherry

* Negative indexing

count from the end

```
print (fruits[-1])
print (fruits[-2])
```

o/p :- cherry
banana

* Slicing a list :-

you can get part of a list using slicing

```
num = [10, 20, 30, 40, 50]
print (num[1:4])
print (num[:3])
print (num[2:])
```

o/p :- [20, 30, 40]
[10, 20, 30]
[30, 40, 50]

* changing list items.

list are mutable - you can modify them.

```
fruits = ["Nagi", "Lakshi", "kaki"]
fruits[1] = "Manasa"
print (fruits).
```

o/p :-
["Nagi", "Manasa", "kaki"]

* Adding items :-

```
fruits = ["apple", "banana"]
fruits.append ("cherry")          # Add to the end
fruits.insert (1, "orange")       # Add at index 1
print (fruits)
```

o/p :- ['apple', 'orange', 'banana', 'cherry']

* Removing items

```
fruits = ["apple", "banana", "cherry", "mango"]
fruits.remove ("banana")   # Remove by value
fruits.pop (1)             # Remove by index.
del fruits [0]             # delete specific element
print (fruits)
```

o/p :- ['cherry', 'mango']

* Looping through a list

```
fruits = ["apple", "banana", "cherry"]
for item in fruits:
    print (item)
```

o/p :- apple
banana
cherry

* checking if an item exits

```
fruits = ["apple", "banana", "cherry"]
if "banana" in fruits:
        print ("yes, banana is in the list").
```

o/p :- yes, banana is in the list.

* Combining two lists :-

```
list 1 = [1, 2, 3]
list 2 = [4, 5, 6]
combined = list 1 + list 2.
print (combined).
```

o/p :-
[1, 2, 3, 4, 5, 6].

* List functions :-

fun	Description	ex	o/p.
len()	num of item	len (fruits)	3.
max()	largest val	max ([2, 5, 8])	8
min()	smallest val	min ([2, 5, 8])	2
sum()	sum of nom	sum ([1, 2, 3])	6
sorted()	sort list	sorted([3, 1, 2])	[1, 2, 3]
pop()	Remove items by index	fruits.sort()	

* Nested lists (list inside a list)

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]
print (matrix[0])
print (matrix[1][2])
```

o/p:-
- [1,2,3]
- 6.

* Reverse:-

Reverse the list.

List . Reverse ().

Examples:-

* `L1 = [34, "Nagi", 27.9, True, 34, 1+8j]`

```
L1
```

o/p:- 34, 'Nagi', 27.9, True, 34, (1+8j)

* `for i in L1:` # indexing.

```
    print (i)
```

o/p:- 34
Nagi
27.9
True
34
(1+8j)
Nagi

* `for i in range enumerate (L1):` # enumerate gives both index and value from list.
- tuple format - (index, val)

```
    print (i)
```

o/p:- (0, 34)
(1, 'Nagi')
(2, 27.9)
(3, True)
(4, 34)
(5, (1+8j))
(6, 'Nagi')

* `L1. append ('sudheer')` # To add value in last.

```
L1
```

o/p:- [34, 'Nagi', 27.9, True, 34, (1+8j), 'sudheer']

* `L1. insert (3,100)` # insert an element at a specific position in the list

```
L1
```

o/p:- [34, 'Nagi', 27.9, 100, True, 34, (1+8j), 'sudheer'] (index = 3, val = 100)

* L1[4] = "false" # To Replace the index position val.

L1

o/p [34, 'Nagi', 27.9, 100, 'false', 34, (1+8j), 'Nagi', 'sudheer']

* 11.pop (1) # To delete val.

o/p: 'Nagi'

* L1. remove (34) # To Remove the val.

L1.

o/p: [27.9, 'false', 34, (1+8j), 'sudheer']

* L1 = [1, 2, 3, 4, 5] # To concatenate the list.

L2 = [4, 5, 6, 7, 8]

L1 = L1+L2

print (L1)

o/p: [1, 2, 3, 4, 5, 4, 5, 6, 7, 8]

* L1 = [1, 2, 3, 4, 5] # To Remove the duplicates and keep in original.

L2 = [4, 5, 6, 7, 8]

L3 = []

for i in L1:

    if i not in L2:

        L3. append (i)

for i in range L2:

    if j not in L3:

        L3 . append (j)

L3

o/p: [1, 2, 3, 4, 5, 6, 7, 8]

* L4 = []

for i in range L1: # To Remove Repeated value. Keep

    if i in L2:

        L4 . append (i)

print (L4)

o/p: [4, 5]

* L1 = [1, 2, 3, 4, 5]

L2 = [4, 5, 6, 7, 8]

L3 = []

for i in L1:

    if i not in L2:

```
        l3.append(i)
for j in l2:
    if j not in l1:
        l3.append(j)
```

To skip the repeated val.

o/p:- [1,2,3,6,7,8]

l3:

* Create even, odd, prime num list from 1 to 20 num.

```
even = []
odd = []
pn = []
for i in range(1,21,1):
    if(i%2==0):
        even.append(i)
    else:
        odd.append(i)
    if(i!=1):
        for j in range(2,i,1):
            if(i%j==0):
                break
        else:
            pn.append(i)
print(f"{even}\n{odd}\n{pn}")
```

o/p:- [2,4,6,8,10,12,14,16,18,20]
[1,3,5,7,9,11,13,15,17,19]
[2,3,5,7,11,13,17,19]

* Slicing:-

var[sart posit : stop post : step size]

default start post is 0

default stop position is upto last

default step size is 1.

* list1 = [34, "Nagi", 27.9, 100, 'false', 34, (1+8j), 'Nagi', 'karona']
list1

o/p:- [34, "Nagi", 27.9, 100, 'False', 34, (1+8j), 'Nagi', 'karona']

* list1[0:4:1]

o/p:- ['Nagi', 'false', 'Nagi'] 27.9, 100]

* # extract odd position element.

list1[1::3]

o/p:- ['Nagi', 'false', 'Nagi']

* list1[-3:] # extract last 3 elements.
Negative ('-') indexing

o/p:- [(1+8j), 'Nagi', 'karuna']

* list1[:]

o/p:- [34, 'Nagi', 27.9, 100, 'false', 34, (1+8j), 'Nagi', 'karuna']

* list1[::-1] # change list last to first negative index

['karuna', 'Nagi', (1+8j), 34, 'false', 100, 27.9, 'Nagi', 34]

* palindrome:-

```
a = input ("enter a value")
print (a)
l= list (a)
rev = ''.join (l[::-1])
print (rev)
if (a==rev):
    print ("it's a palindrome")
else:
    print ("not a palindrome")
```

o/p:- enter a value MOM
MOM
MOM
it's a palindrome.

* Tuple :-

It is a multivalued hetrogenous variable, its immutable, ordered and allow duplicates.

It is represented by tuple() or ().

* multivalued :- It can store multiple values in a single variable.

ex:- t = (10,20,30)

* Heterogeneous :- it can contain elements of diff data types.

ex:- t = (10, 'hello', 3.5, True).

* ordered :- the element have a fixed order, and the order will not change.

ex:- t[0] will always give the first element.

* immutable :- once created, you cannot modify (add, remove or change) elements.

ex:- t = (1,2,3)

t[0] = 5 # Error - 'tuple' object does not support item assignment.

* Allows duplicates :- Tuple can contain repeated elements.

ex:- t = (1,2,2,3).

* Representation:

- using parentheses :- t = (1,2,3)
- using tuple() fun :- t = tuple ([1,2,3])

A tuple is like a list that cannot modify (you can't add, remove, or change elements once created)

* How to create a Tuple.

- using parentheses()

```
my_tuple = (10,20,30,40)
print (my_tuple)
```

- using the tuple() function

```
my_tuple = tuple ((10,20,30))
print (my_tuple).
```

* Example of mixed data types.

```
mixed_tuple = (10, "apple", 3.14, True)
print (mixed_tuple)
```

o/p:-
(10, 'apple', 3.14, True)

* single element Tuple.

If you want to create a tuple with only one item, you must add a comma after it.

```
t = (10)
print (type (t1))      # <class 'int'>

t2 = (10,)
print (type (t2))      # <class 'tuple'>
```

o/p:- (10,) → tuple
(10) → integer

* Accessing Tuple elements.

you can access tuple items using index number (like a list).

```
fruits = ("apple", "banana", "cherry")
print (fruits [0])     # apple
print (fruits [-1])    # cherry (-ve index).
```

o/p:- apple
cherry.

* slicing a tuple:-

you can get a part of the tuple using slicing.

```
num = (10,20,30,40,50)
print (num [1:4])
print (num [:3])
print (num [2:])
```

o/p:- (20,30,40)
(10,20,30)
(30,40,50)

* Tuple is immutable (can't change).

you cannot modify, add, or delete elements in a tuple.

```
fruits = ("apple", "banana", "cherry")
fruits [1] = "orange"
```

o/p:- Type error : 'tuple' object does not support item assignment.

* But you can convert tuple → list → modify → convert back.
if you need to change values:

```
fruits = ("apple", "banana", "cherry")
# convert to list
temp_list = list (fruits)
temp_list [1] = "orange"
# convert back to tuple.
fruits = tuple (temp_list)
print (fruits)
```

o/p :-
('apple', 'orange', 'cherry')

* Tuple operation and functions.

function	Description	example.
len()	Returns the num of item	len (fruits)
count (x)	Returns how many times x appears	t.count (2).
index (x)	Returns the index of 1st occurance x	t. index ("apple")
min ()	Returns small val	min (t)
max ()	Return large val	max (t)
sum ()	Adds all numeric element	sum (t).

Ex:-

```
num = (10, 20, 30, 40, 50, 20)
print ("Tuple:", numbers)
print ("Length:", len (numbers))
print ("count of 20:", num.count (20))
print ("index of 30:", num.index (30))
print ("sum:", sum (num))
print ("Max:", max (num))
print ("Min:", min (num))
```

o/p :- Tuple : (10, 20, 30, 40, 50, 20)

Length : 6
count Of 20 : 2
index of 30 : 2
sum : 170
Max : 50

Ex:- t1 = (34, 'Nagi', 27.9, 100, 'False', 34, (1+8j), 'Nagi', 'karuna')

tupple (t1)

olp:- (34, 'Nagi', 27.9, 100, 'False', 34, (1+8j), 'Nagi', 'karuna')

* t1[2] # indexing

olp:- 27.9.

* t1.append (86).

olp:- error - 'tupple' obj has no attribute 'append'.

* t1.insert (5, 'Ng')

op:- error

* t1.pop ()

op:- error

* tupple (t1)

olp:- (34, 'Nagi', 27.9, 100, 'False', 34, (1+8j), 'Nagi', 'karund)

* l1 = list (t1)

l1[4] = True

t1 = tuple (l1)

print (t1)

(34, 'Nagi', 27.9, 100, True, 34, (1+8j), 'Nagi', 'karuna')

* SET :-
A set is a collection of unique and unordered items.

means:
. # does not allow duplicates.

* multivalued :- # can store multiple value in a single variable.

ex:- S = {10, 20, 30}

* Hetrogeneous:- # can store diff data types together.

ex:- S = {10, "hello", 3.14, True}

* Mutable :- you can add or remove elements after creation.

ex:- S = {1, 2, 3}
S.add(4)
print(S)

o/p:- {1, 2, 3, 4}

* unordered:-
The elements don't have a fixed position or index.
you can't access items using indexes like S[0].

* No Duplicates:- Repeated elements are automatically removed.

ex:- S = {1, 2, 2, 3, 3}
print(S).

o/p:- {1, 2, 3}

* Representation:
- using curly braces {} → S = {1, 2, 3}
- using set() function → S = set([1, 2, 3])

Note:- {} alone creates an empty dictionary, not a set.
So for an empty set, always use set().

Ex:-
S = {10, 20, 30, 20, 40}
print(S) # {40, 10, 20, 30} (unordered, no duplicates),
S.add(50)
print(S) # {40, 10, 50, 20, 30}
S.remove(10)
print(S). # {40, 50, 20, 30}

o/p:- {40, 10, 20, 30}
{40, 10, 50, 20, 30}
{40, 50, 20, 30}.

* How to create a set.

* using curly braces { }

```
my_set = {10, 20, 30, 40}
print (my_set)
```

* using the set() function.

```
my_set = set ([10, 20, 30, 40])
print (my_set)
```

* Example with duplicates.

```
numbers = {10, 20, 30, 10, 20}          o/p :- {10, 20, 30}.
point (numbers)          # set duplicates are automatically remove
```

* Mixed data type.

```
mixed_set = {10, "apple", 3.14, True}          o/p :- {'apple', 10, 3.14}
print (mixed_set).
```

The order may be change every time, you print - because sets are unordered.

* Accessing set elements :-

you cannot use indexing like set[0].

But you can use a for loop to access each element.

```
fruits = {"apple", "banana", "cherry"}.
for fruit in fruits:
    print (fruit)
```

* Adding and Removing elements.

To add one item :-

```
fruits = {"apple", "banana", "cherry"}
fruits.add ("orange")
print (fruits).
```

o/p :- {"apple", 'banana', 'cherry'}

Add multiple items :-

```
fruits.update (["kiwi", "mango"])
print (fruits)
```

Remove an item :-

```
fruits.remove ("banana")   # error if not found.
print (fruits)
```

Discard an item (no error if not found).

```
fruits.discard ("grape").      # no error if not found.
```

Remove a random item :-

```
fruits.pop()
```

clear all items :-

```
fruits.clear().
```

* set operations (very use ful to Data Analysis).

set are great for mathematical operation like union, intersection, etc.

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

operation	Description	example	Result
union	combines all unique element	A.union(B)	{1, 2, 3, 4, 5, 6, 7, 8}
intersection	common elements in both sets	A.intersection(B)	{4, 5}
difference	elements in A but not in B	A.difference(B)	{1, 2, 3}.
symmetric difference	elements not common to both.	A.symmetric_difference(B)	{1, 2, 3, 6, 7, 8}.

Ex :-

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

```
print ("A:", A)
print ("B:", B)
print ("union:", A.union(B))
print ("intersection:", A.intersection(B))
print ("difference:", A.difference(B))
print ("symmetric difference:", A.symmetric_difference(B))
```

O/P:- A : {1,2,3,4,5}
B : {4,5,6,7,8}
union : {1,2,3,4,5,6,7,8}
intersection : {4,5}
difference : {1,2,3}
symmetric difference : {1,2,3,6,7,8}

Ex:- S1 = {34, 'Nagi', 27.9, 100, 'False', 34, (1+8j), 'Nagi', 'karuna'}
S1

* S1.add (123)
S1
O/P:- {(1+8j), 100, 27.9, 34, 'False', 'Nagi', True, 'karuna'}

* S1.discrad ('True')
S1.
O/P:- {(1+8j), 100, 123, 27.9, 34, 'Nagi', True, 'karuna'}

* S1.pop()
S1
O/P:- {(1+8j), 100, 123, 27.9, 'False', 'karuna', 'priya'}

* S1.remove ('False')
S1.
O/P:- {(1+8j), 100, 123, 27.9, 'karuna', 'Nagi'}

* S1.add (True)
S1.
{(1+8j), 100, 123, 27.9, 'karuna', True, 'Nagi'}

* list 1 = [1,1,1, 2,2, 3,3,3, 4,4, 5, 6, 8, 9] # To Remove duplicate values
set (list 1)

o/p:- {1, 2, 3, 4, 5, 6, 8, 9}

* list 1 = list (set (list 1))
list 1

o/p:- [1, 2, 3, 4, 5, 6, 8, 9]

* DICTIONARY :-

A dictionary is a multivalued, heterogeneous data type that stores data in key-value pairs.

syntax:- my_dict = {key : value}

(or)

my_dict = dict ()

* It is heterogeneous can store different types of data (int, str, list)
* It is multivalued contains multiple key-value pairs. change, add, remove
* key are immutable (cannot be changed, must be unique).
* values are mutable (can be changed)
* it is unordered (in python), but insertion-ordered.

ex:-

```
stu = {"name": "Nagi", "age": 20, 'marks': 85}
print (stu)
```

o/p:- {'name': 'Nagi', 'age': 20, 'marks': 85}

* d1 = dict (name = "Nagi", age = 20, city = "Andhra")

* Accessing elements :-

```
print (d1["name"])
print (d1.get ("city"))
```

o/p:- "Nagi"
"Andhra"

* Adding or updating.

```
d1["age"] = 21          # update existing key
d1["country"] = "India" # Add new key
```

O/P:- {'name': 'Nagi', 'age': 21, 'city': 'Andhra', 'country': 'India'}

* Deleting elements

```
del d1["city"]      # Remove key
d1.pop("age")       # Remove key and Return value.
d1.clear()          # Remove all items.
```

Ex:-

```
person = {"name": "Nagi", "age": 25,
          "skills":["python", "power BI", "SQL"]}
# Access
print(person["skills"][0])
# update
person["age"] = 20
# Add new key
person["city"] = "chennai"
# Display dictionary
print(person)
```

O/P:- {'name': 'Nagi', 'age': 20, 'skills': ['python', 'power BI', 'sql'], 'city': 'chennai'}

Ex:-

```
d1 = {'name': 'Nagi', 'DOB': '02/06/2004', 'ph no: 9390533260'}
```

* d1.keys()

O/P:- dict_keys(['name', 'DOB', 'ph no'])

* d1.values()

O/P:- dict_values(['Nagi', '02/06/2004', '9390533260'])

STRINGS :-

A string is a sequence of characters enclosed in quotes. you can use single quotes (' '), double quotes (" ") or triple quotes (""" """ / ''' ''').

Ex:-

```
name = 'Nageswari'
greeting = "Hello"
sentence = """This is a multiline string"""
```

String characteristics

1. strings are ordered.

Each character has an index starting from 0.

```
word = "python"
print (word[0])
print (word[3])
```

o/p :- p
h

* strings are immutable.

once created, you cannot change their content directly.

```
word = "python"
new_word = language + word[1:]
print (new_word)
```

o/p :- language python

* string operations.

1. Concatenation

```
first = "Hello"
second = "world"
result = first + " " + second
print (result)
```

o/p :- Helloworld.

2. Repetition

```
word = 'Hi'
print (word * 3)
```

o/p :- Hi Hi Hi

3. slicing

```
text = 'python'
print (text[0:3])
```

o/p :- pyt
hon

* String Methods

Method	Description	Example
upper()	converts to uppercase	'hello'.upper() → 'HELLO'
lower()	Converts to lowercase	'HELLO'.lower() → 'hello'
strip()	Remove spaces	' hi '.strip → 'hi'
replace(a,b)	Replace text	'cat'.replace('c','b') → 'bat'
split()	split into list	'a,b,c'.split(',') → ['a','b','c']
Join()	Join list with a separator	'-'.Join(['a','b']) → 'a-b'
Find()	Find index of substring	'hello'.find('e') → 1
Count()	count occurances.	'banana'.count('a') → 3

String Formatting:-

1. using f-string

```
name = "Nageswari"
age = 21
print(f"My name is {name} and I am {age} years old.")
```

2. Using format

```
print("My name is {} and I am {} years old.".format("Nageswari",21))
```

Ex:-

```
str2 = 'besant123@gmail.com'
v=[]
c=[]
n=[]
s=[]
for i in str2:
    if i.isalpha():
        if i in 'aeiou':
            v.append(i)
        else:
            c.append(i)
    elif i.isdigit():
```

```
    else:
        s.append(i)
print(f"{v}\n{c}\n{n}\n{s}")
```

O/P:- ['e', 'a', 'a', 'i', 'o']

['b', 's', 'n', 't', 'g', 'm', 'l', 'c', 'm']

['1', '2', '3', '4']

['@', '.']

* str = 'PYTHON SUBJECT'

```
upc = 0
lwc = 0
for i in str3:
    if i.isupper():
        upc += 1
    else:
        lwc += 1
print(f"upper count = {upc}\n lower count = {lwc}")
```

O/P:- upper count - 8

lower count - 6