Unit - 03 ≫ STRING

String :- A string is a sequence of characters enclosed within ~~single~~ ~~or~~ Single or double or tripple quotes.
string are immutable it means that they can not we change after creation.

Creation A string

Single line string
a = " Hello"

Multiline string
a = " Hello"
b = " How are you"
c = " I am Fine"

Accessing characters :- individual characters using negative or positive indexing.

Example :-

```
S = " Python programming"
print (S [0,-1]).
```

Indexing / slicing and / Substring :-

- Indexing :- S[i] ≫ give the characters at index i ~~[illegible]~~
- Negative indexing S[-1] ≫ Is the last character
- Slicing s[start : end] gives substring from intex 'start' upto 'end'

* S[:k] :- ~~code~~ is from start k - 1

जैसे आप किसी फाइल का पता

* S[K:] :- is from K to end.

• S[:] :- Is a copy entire string

Slicing S[start : end : stop]

P y t h O N
| | | | | |
0 1 2 3 4 5

S = "python"

```
print (S[0])    =>   'p'
print (S[-1])   =>   'n'
print (S[1:4])  =>   'yth'
print (S[:3])   =>   'pyt'
print (S[3:])   =>   'on'
```

Note :-

S[::2] → 'pto' picks every second character. →
S[::-1] → Reverse picks the string. → n
'pinohtyp'

Note :-

Raw string :- where ~~state~~ escape characters are not processed.

Example :-

P = " C:|new Folder|test
o/p → new Folder test.

P = r" C:|new Folder|test

o/p ⇒ C:|newFolder|test.

Escape		Meaning.
'\n'	→	New line
'\t'	→	tab
'\\'	→	literal backslash
'\'' '\"'	→	quote characters inside string.
'\r', '\b', '\f' etc	→	other control characters.

common operations on string :-
(दो STRING को जोड़ना)
<u>concatination</u> :- Use pluse (+) sign to join strings

Ex :- $n_1 + n_2$
∴ n_1 = Ankit
∴ n_2 = Kumar
∴ $n_1 + n_2$
= Ankit + Kumar
= Ankit Kumar.

<u>Repetation</u> :- 's * n repeats 's' n times.
<u>Ex</u> :- n_1 = "ankit"
~~[illegible]~~

<u>Repetation</u>:-
$n_1 * 3$
<u>Ankit</u> Ankit $\overset{n}{\text{A}}$kit

Membership:- Sub in S' returns 'true/False' if sub is a substring. of S

'S = " Hello Python"

'Hel'

Use Full string Methods:-

Method	disc	Example
upper~~lower~~()	— convert all characters to upper case	"hello" ⟶ "HELLO"
lower ()	— convert all characters to lower case	"HELLO" ⟶ "hello"
title ()	convert first letter of each word to upper cas	"welcome to python" ⟶ ⟶ "Welcome To Python"
capitalize()	capitalizes First character only	"python is fun" ⟶ "Python is fun"
Swap case()	Swap upper case to lower case	"PyThON" ⟶ "pYtHoN"

Find(sub):- Returns index of First occurance, -1 if not found

Example:- ~~Find f"~~ S= python

find ("th") ⟶ 2

rFind(sub):- Finds from the right side

s= ".Python Programming"

rFind ("mm") ⟶ 12

ⓔ S₁ = Python Python

rFind ("th") ⟶ 8.

Index (Sub):- Same as find function, but raises ~~a~~ error if not found.

Count (sub):- Counts occurrences of sub string.
Example:- s = " Aman "
print (s. count ("a")).

Starts with (sub):- checks if string starts with substring.
s = " Ayush "
print (s. Startswith(" Ay"))

Ends with(sub):- checks in strings ends with substring.
s = "Dhananjay"
print (s. endswith ("an")).

~~Replace (sub)~~ :-
Replace (old, new):- Replaces substring with another.
Example:- " I like Python"
print(s. replace("Python", pps)).

Method		Description		Example
'lstrip()	→	Removes left spaces.	→	a = "_hello" a. lstrip → "hello"
r strip()	→	Removes right spaces	→	a = " hello_" a. rstrip → "hello"
join (iterable)	→	Joins elements using a seprator	→	a = " ". ~~join~~ join ~~join~~ ["Python", "is", "Fun"] a.join [Python is Fun]
split(sep)	→	split string into list	→	a = "A,B,C" a. split(".") = ['A', 'B', 'C']
is alpha()	→	true if all are letters	→	a = '" abc "' a. isalpha [] → True.
~~isdi~~ isdigit()	→	true if all are digit	→	a = '" 1,2,3 "' a. isdigit () → True.
is alnum():-		true if all letter or digits	→	a = '" abc 123 "' a. isalnum [] → True.
is lower ()	→	true if all characters are lower	→	a = '" abc "' a. islower [] → True.
is upper ()	→	true if all character are upper	→	a = '" PYTHON "' a. is upper [] → True.
Center (width, char)	→	Centers text		a = "Hi" a. center (6, '*') => ** Hi **

ljust(width. char) → left aligns text → a = "Hi"
a.ljust (6, "—")
⇒ 'Hi ----'

rjust(width, char) → Right align text → a = "Hi"
a.rjust (6, "—")
⇒ '----Hi'

zfill (width) → ~~Bauts~~ Pads with zero. → a = '42'
a.zfill(5) = 00042

Build in function

	Function	Description	Example.
①	len (list).	Return number of elements	list = [10, 20, 30, 40, 50] print(len (list))
ⓐ	Max (list)	Returns maximum value	a = max (list) print a
③	Min(list)	Return Minimum Value	a = min (list) print (a)
④	Sum(list) →	Returns sum of Numeric element	a = sum [list] prin(a)
⑤	sorted(list) →	Return sorted copy of list	list = [10, 30, 50, 20, 40] a = sorted [list] print [a).

⑥ append (x) → Add single element at end → List. append [10]
⑦ Extend [iterable] → Add multiple elements → list. extentd([1, 2, 3])
⑧ Insert (i, x) → insert x at index i → list. insert([2, 100]).
⑨ Remove (x) → Remove first accurance of x → List. Remove (8).
⑩ Pop([i] → Remove and return element → List. pop ([1])
⑪ Clear() → Remove all elements → List. clear().
⑫ index. (x) → Return index of x → List. index [3)
⑬ Count (x → Count occurances of x → List. count (2.
⑭ Sort() Sort list in accending order → List. sort ( )
⑮ Reverse() → Reverse the list → List. reverse()
⑯ Copy() → Return a copy of the list → List. copy()

```
num = [10,20, 30, 40, 50]
print (num [0])
print ( Num[-1])
            output = 10 , 50
```

(ii) Slicing (used to access a range of elements)

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

Updating and Modifying list :-

```
List = [10,20, 30, 40,50]
List [1] = 25
print (list)
      Output - [10,25,30,40, 50]
```

Creating list :- Build in Fun

Function	description	Example

List

LIST :- A list collection of ordered, mutable (changeable) and heterogeneous elements. it allows storing multiple values in single variable.

Example :-

List = [ ] empty list.

Characteristics of list

ⓐ Ordered :- Elements have are define sequence.
ⓑ Mutable :- You can modify them after creation.
ⓒ Different Data Types :- integer float char, string etc.
ⓓ Nested :- A list inside another list

(i) empty list :-

List = [ ]

(ii) List of integers →

List 1 = [10,20,30,40,50,60]

(iii) List of Mixed Data types →

List 2 = [25, "Hello", 'C', 3.14, True]

(iv) Nested List :-

List 3 = [[1,2], [3,4], [5,6], [7,8]]

Accessing Element :-

(a) indexing :-
- indexing starts from zero.
- Negative index starts from minus one (-1).

Tuples :-   ( ) Paranthesis :-

Tuple is an ordered immutable collection of element

-> It is used to store multiple items in a style

-> Once created cannot modify (Address, remove or change its items)

-> Tuples are ~~father~~ faster that list and used when the data should not change

Syntax

Tuple_name = (item1, item 2, ...n)

Note :- If user only one element you must use , (comma)

a = (5,) → Tuples

a = (5) → Integer.

Creating A Tuples :-

```
T1 = [10,20,30, 40,50]
print(T1).
```

Accessing tuple Elements :-

```
t1 = (10,20, 30, 40, 50)
print (t1[0])
print (t1[-1])
```

Slicing Element :-

```
t1 = [10,20,30,40,50]
print (t1[1:3]).
```

Immutalibliity :-

```
t = (10,20, 30, 40, 50)
t[0] = 90 → Not allowed  xx error xx
```

modifying, adding element, removing element list operation :-

① Concatenation
```
a = [1, 2, 3]
b = [4, 5, 6]
c = [a + b]
print (c)
```

(ii) Repetition
```
a = [1, 2]
print (ax3)
```

(iii) Membership test.
```
name - ["Amit", "Sunil", "Ganesh"]
print ("sumit" in name)
print (" Array" not is name)
```

(iv) Copying list :-
```
a = [1, 2, 3]
b = a
b.append [4]
print (a)
print (b)
```

```
a = [1, 2, 3]
c = a.copy
c.append (4)
print (a)
print (b)
```

~~Add~~ Advantages of list :- • Dynamic in size
- Store multiple Data type together
- Make Buid in Function make them easy to use.
- allow slicing, looping and comparison limitation
- ~~of list~~ limitation of list
- slower compare to array in other language
- cannot large numeric data set efficiently.

Method/Function Description ~~Example~~

~~Append~~

Traversing ~~Always~~ a list using a loop :-

For

```
a = [10,20,30 ,40,50]
    For i in a
    print (i).
```

while

```
a = [10,20, 30,40]
    b = len(a)
    n=0

while (n<=b)
    print (a[n])
    n -- n+1
```

WAP in python to sum avg of list element.

```
a = [10,20,30, 40,50]
    b = sum(a)
    print (b)
    n = len(a)
    Avg = b/n
    pring (avg).
```

② num = [10,20,30]
num.append(40)
num.remove(30)
print (num)
o/p = [10,30,40]

③ a = [1,2,3,4,5]
a[1:3] = [20,30]
print(a)
o/p → [1,20,30,4,5]

Unpacking in ~~Tup~~ Tuple :-

```
{student = ("Amit", 21, "B-Tech")
        name, age, course = student. }

{ marks = [85, 90, 88]
  marks.append(95)
  marks[1] = 92 }

print("student Details")
print ("Name", name)
print ("Age", age)
print ("Course", course)
print ("Marks", marks).
```

Function :- A function is defined using the def keyword in python, followed by the function name and paranthesis.

Syntax :- ~~<name of~~

$$\begin{tikzcd} \text{def} \arrow[d] & \text{<name of fun>} \arrow[d, dash] & \text{( )} \arrow[d, dash] \\ \text{keyword} & \text{Name of the function} & \text{Paranthesis.} \end{tikzcd}$$

Unpacking in string :-

```
s = " Hello"
a, b, c, d, e = s
print(a)
print(b)
print(c)
print (d)
 print (e).
```

Mutable Sequences :- Mutable sequence can be modified after creation.

* you can change , remove element.

Common operation on mutable sequences.

① indexing.
② slicing
③ modification
④ opend
⑤ Remove.

Modifying List :-

```
fruits = ["apple", "Banana", "orange"]

Fruits [1] = " Mango"

print (Fruits)
```

Unpacking :- • unpacking means ~~values~~ assigning value ~~for~~ from a sequence (tuple, list, string) to variables in a single step.

- It allows multiple assignments at once
- The number of variables on the left must ~~mark~~ match the number of values in the sequences.

Example :-

```
stu = ("Amit", "B-Tech", "20")
name, course, age = stu
print (name)
print (Course)
print ( age )
```

Stu = ["Amit", "B-Tech",

unpacking of Tuple

```
n = [1, 2, 3, 4]
a, b, c, d = n
print(a, b, c, d)
```

unpacking of list

Extend unpacking.

```
num = (10, 20, 30, 40, 50)
 a, *b, C = num
 print(a)   = 10
 print(b)   = [20,30,40]
 print (c)  = 50
```

① Types of func^n

① def disp():
print("Hello! ~~Bansal to~~ Wellcome Bansal")

disp():

② ~~def(~~

② def disp(name)
print("Hello", name)
disp("Bansal")

③ ~~def(~~ def add (a,b)
return a+b
result = add(5,10)
print(result).

④ def get_pi()
return 3.1415
print("Value of Pi", get_pi())

⑤ def cal (a,b) → यह फनकशन को डिफाइन करेगा ~~फिर~~ फिर
return a+b, a-b, a*b → रिटर्न करेगा उसके बाद
sum, diff, pro = cal (10,2) फिर यहा पर रिटर्न कराये गये Function की unpacking karayenge
print("Sum", Sum)
print("Differenc", diff) → uske बाद सब बारी-बारी से
print("product", pro) execute होगा.

⑥ def ~~even~~ even(num):
if num% 2 = = 0:
print(" num is even")
else
print("no is odd")

even(10) } output
~~odd~~(7)

User defined function :- User defined function in python are custom functions created by programmer to perform specific task.
They enhance code reuseability, modularity module and readability

Example :-

```
def total ()
```

Part of function

① Define the function
① Paraemetter / Arguments
③ Function Body.
④ Return statement
⑤ Function call.

Parameters/Arguments

```
def add (a,b) →           definition of func
result = a+b  →           funcⁿ body
sumval = add(5,3) →       return statement
print ("Sum" = ", sumval)  Funcⁿ call
=8
```

Execution of a function :-

① Python read the ~~of~~ function definition but does not execute it immediatly.
⑪ When the function is called, control passes to the funcⁿ body
③ The statements inside the functions execute.
④ After ~~s~~ the execution the control returns to the calling point.

- Paranthesis may contain parameters / arguments. which are input to the function.
- The function body which contains the core to executed. below the function defnition line
- Optionally a function can be written a value using return statement.

Types of Function :-

$$\begin{tikzcd}
 & \text{Types of Function} \arrow[dl] \arrow[dr] & \\
\text{"Built in func}^n\text{"} & & \text{"user defind func}^n\text{"}
\end{tikzcd}$$

Built if Func^n :- Built in Func^n are predefined func^n available in python without requiring any inclose incode statement.

They provide fundamental funcality lovering banieus aspects of programming from basic operations to data type conversion.

abs ()   range( )
pow()   type( )
round()   set ( )
sum()
min ()
max()
len ()

③
```
def out ()
    X = "outer"
    def in ():
        X = "  inner"

    in ()
    print ("After in ():", x)

    out ()
    print (" After out ():", x).
```

Example:-

```
def disp (name, message = " Good morning")
    print (" Hello", name + ", " message)

    disp ("Amit")
    disp (" Amitesh", " How are you")
```

Output

```
Hello Amit Good Morning
Hello Amitesh How are you.
```

Scope rules in Python :- scope determines ~~th~~ where a variable is accessible ~~bit~~ within the program.

local:- inside a ~~fun~~ current function
Enclosing:- in enclosing functions (Nested functions).
Global:- Decleared at the top level of script
Built in:- Reserved named in python (len, print, etc.)

(i)
```
x = 10
def disp ()
    x = 5
print ("inside fun" =", x)
    disp ()
print (" outside func^n = ", x)
```

(ii)
```
~~cnt~~ ~~cnt~~ = 10
def increment () :
    global cnt
        cnt + = 1
    print("inside func^n=", cnt)
    increment ()
print ("outside func^n=" cnt)
```

KEYWORDS ARGUMENTS :- When calling a function you can pass arguments using the parameters name, This means the code more readable and the order of arguments optional

Key Points :- ① you specify the parameter name with a it value
② Order of arguments does not mattere when using keyword.
③ use full a way a funcⁿ has many parameter.

Example

```
def student(name, age, course):
    print ("Name = ", name)
    print ( " Age :-", age)
    print ( " Course :", course)
~~Student~~
student ('age = 20","name = ankit", " course = CSE/BTech", ) invalid.

Student ("name = "Ankit", "age = 20", "course = B-Tech ") valid.
```

. Default Argument :- Default arguments are parameter assume default value if a value is not provided in the function call.

Key Points :-
① simplyfies function call when some parameter usually have save value
②