**String Handling**

**Day-01**

**25-03-2025**

**===============**

1: What is the string?

-------------------------

-> String is a literal.

-> can store into a variable

-> the group of characters which enclosed with single quotes

or double quotes or triple quotes is called as "String".

s1 = '@'

s2 = 'python'

s3 = ":"

s4 = "Programming"

s5 = '''a'''

s6 = '''Language'''

print(type(s1),type(s2),type(s3),type(s4),type(s5),type(s6))

Note:

-----

Like java, in python there is no character datatype

In python the character notation also consider as "string".

2. Multi-line String:

----------------------

-> triple quote can be used.

text = '''

Python is Easy language

We can use python in automation

and also for development of various applications

from various domains.

'''

print(text)

print(type(text))

3. Do we able to define Dynamic string?

---------------------------------------------

-> there are two ways:

1) input()

2) eval()

1) using input():

-----------------

Syntax:

string-variable = input("Enter some string:")

or

string-variable = str(input("text")) ==> not consider to define

Note:

-----

While defining the dynamic string using input(), the quotes are optional.

If you can define with quotes, the quotes are also consider as separate characters.

2) using eval():

----------------

eval() used to convert the input()

to take any collection like list, tuple, dictionary etc.

Ex: a = [10,20,30,40,50];

Syntax:

eval(input("Enter some string:"))

note:

-----

while defining the dynamic string using eval(), without the quotes string definition never understood.

# dynamic string

s1 = input("Enter user name:")

s2 = eval(input("Enter password:"))

print("The user name = ",s1)

print("The password = ",s2)

4. String Comparison:

-------------------------

Python-2:

cmp()

python-3:

using relational operators only for the string comparison

# dynamic string

# assume, system credentials are:

# username = "admin"

# password = "admin123"

username= input("Enter user name:")

password = eval(input("Enter password:"))

if username == "admin" and password == "admin123":

print("Ashok IT welcomes you!")

else:

print("Check your credentials")

print(username < "admin123")

5. Length of the string:

-------------------------

i) using len()

--------------

Syntax:

len(string-variable)

ii) using for loop

------------------

Syntax:

for variable/iterable in string-variable:

loop body

# length of the string

s1 = "Python is Easy programming language"

print("The length of the string = ",len(s1))

# using for loop

count = 0

for i in s1:

count += 1

print("The number of characters = ",count)

6. Accessing of the string:

---------------------------

-> getting of individual characters of the string

-> getting of group of characters of the string

-> for the string accessing,

we have two approaches:

1) indexing

2) slicing

1) indexing:

--------------

-> using indexing, we can get only one character at a time

-> indexing can be two types:

1) positive indexing

2) negative indexing

-> when we need to access the characters of the string from left to right (forward access), we can use "positive indexing".

-> positive index can be ranged from:

0 to len(string) - 1

Ex: string has 20 characters

range of index ==> 0 to 19

s1 = "python"

index 0 ==> 'p'

index 1 ==> y

index 2 ==> t

index 3 ==> h

index 4 ==> o

index 5 ==> n

-> when we want to access the characters of the string from right to left (reverse access) we can use "negative indexing".

-> negative indexing range:

-1 to -len(string)

Ex: string ==> 20 characters

rage ==> -1 to -20

s1 = "python"

index -1 ==> 'n'

index -2 ==> 'o'

index -3 ==> 'h'

index -4 ==> 't'

index -5 ==> 'y'

index -6 ==> 'p'

Syntax for the indexing:

----------------------------

string-variable[index-number]

s1 = "Python"

# positive index/forward access

print("First = ",s1[0])

print("Second = ",s1[1])

print("Third = ",s1[2])

print("Forth = ",s1[3])

print("Fifth = ",s1[4])

print("Sixth = ",s1[5])

# negative indexing/reverse access

print("Last = ",s1[-1])

print("Previous = ",s1[-2])

print("Previous = ",s1[-3])

print("Previous = ",s1[-4])

print("Previous = ",s1[-5])

print("Previous = ",s1[-6])

# print(s1[6]) # index error

**Day-02**

**26-03-2025**

**=====================**

**Accessing of Strings:**

**==============**

1) Indexing ==> When we want to access the individual character of the string

Syntax:

string-name[index-value]

here:

index-value ==> +eve or -eve

+eve ==> 0 to len(string) - 1 ==> left to right (forward access)

-eve ==> -1 to -len(string) ==> right to left (reverse access)

2) Slicing

-----------

-> slice ==> piece or part of the string or sub-string

Syntax:

string-name[start-index: stop-index: step]

here:

start-index ==> the starting point for the slicing

stop-index ==> the ending-point in the slicing

but from the stop-index value, the PVM can always consider the less-than one only

Ex: stop-index = 10

slicing can end with 9th character (10-1)

step ==> difference from start index to next index

Ex: 0 th character

next-character : 2nd index

here difference ==> 2

Note:

-----

the default value for the start-index = 0

the default value for the stop-index = len(string)-1

the default value for the step ==> 1

# slicing

string = "Learning Python is Easy"

# start index < stop index: step must be +eve

print(string[0:8]) # no step value, default value = 1

print(string[:8]) # no start and no stop, default start = 0, default step = 1

print(string[0:]) # no stop, default value = last character

print(string[:]) # no start and no stop, default of start = 0 and stop = last

print(string[0:11:2]) # Lann y

print(string[10::3]) # yoiEy

print(string[:16:5]) # Liy

# we can define the step as +eve or -eve

# when start index > stop index: step must be -eve

print(string[10:0:-1])

# string reverse

print(string[::-1])

# when we have to define the step as '-1' without start and stop values:

# PVM can change the direction of string access

# by default, the direction of string access is "forward"

# when the step is '-eve' and no start and stop present:

# slicing can perform in reverse

**8. String Math Operations:**

**==================**

String Concatenation:

------------------------

-> to define with '+'

-> concatenation ==> joining of two strings into one

Syntax:

str1 + str2 + str3 + .....

Note:

-----

string concatenation is possible on only strings

Ex: "x" + 12 ==> "x12" ==> possible in java

"ZZ" + 12 ==> error.

# string concatenation

str1 = "Python "

str2 = "is "

str3 = "Learning "

str4 = "is"

str5 = " Easy"

print(str1 + str2 + str3 + str4 + str5)

print(str1 + str(21))

String Repetition

------------------

-> When we want to repeat the same string for several times, we can use "String Repetition".

-> '\*'

Syntax:

string-data \* n

here:

n = number of times

str1 = "Python "

print(str1 \* 10)

**9. How to remove spaces from the string:**

**==========================**

Note:

-----

Space is one of the character of the string

-> if a string have spaces at the beginning or at the ending, we can able to remove

but if a string have spaces in between the string data, no removal of spaces possible.

Ex: " python " => we can remove spaces

"py th on" ==> can't remove spaces

strip():

-------

using this we can remove the white spaces from the beginning or ending of the string.

Syntax:

string-data-name.strip()

string = "Python is Easy"

string1 = " Python"

string2 = "Python "

print(string.strip())

print(string1)

print(string1.strip())

print(string2)

print(string2.strip())

**10. How to find the sub-strings:**

**=====================**

-> finding of the first occurrence of the given sub-string in the string

find():

------

Syntax:

string-data-name.find(sub-string)

find() can return the first occurrence of sub-string from the main string

if the specified sub-string is the member of given string

otherwise: return '-1'

string = "python is easy to learn"

index = string.find('y')

i1 = string.find('ea')

i2 = string.find("Java")

print(index)

print(i1)

print(i2)

**11. How to replace the string with another string:**

**================================**

replace():

----------

Syntax:

string-name.replace('old-string','new-string')

data = "Python is hard to understand"

print(data)

print(id(data))

data = data.replace('hard','easy') # re-definition

# string immutable data

print(data)

print(id(data))

**12. How to join sub-strings into one string:**

**---------------------------------------------------**

join():

-------

Syntax:

'separator'.join (list of sub-strings)

l1 = ['Python','is','Easy','Programming','Language']

l2 = '26','03','2025'

s1 = ''.join(l1)

s2 = ' '.join(l1)

s3 = '.'.join(l2)

s4 = '-'.join(l2)

s5 = '/'.join(l2)

print(l1)

print(s1)

print(s2)

print(l2)

print(s3)

print(s4)

print(s5)

**Day-03**

**27-03-2025**

**===============**

**String Validations:**

**=============**

-> the string is in the valid format or not , to check "string validations" can help.

Ex:

Ecommerce application:

Registration form

user name:

mobile

mail id

address

etc.

1) startswith:

--------------

Job portal:

Naukri

url of linkedin

https://www.username-linkedin1203af1122

-> whether the string has started with given set of characters or not, to check "startswith()" can be used.

Syntax:

string-data.startswith("set-of-characetrs")

url = "http://www.ashokit.in"

if url.startswith("https://"):

print("Testcase passed")

else:

print("Need to refer the internal team because the url is not in the format.")

2) endsWith():

---------------

-> when we need to check whether the string is ended with specified group of characters or not.

Syntax:

string-data.endswith("group-of-characters")

gmail = "ravivraoinfs@gmail.com"

if gmail.endswith("@gmail.com" or "@xyz.com" or "@xyz.in"):

print("User mail id is valid.")

else:

print("The application accepting the mail ids as the mentioned guidelines only.")

3) isalpha():

-------------

Whether the given string is defined with only alphabets or not.

Syntax:

string-data.isalpha()

4) isalnum():

-------------

whether the given string is with all alphabets and digits or not.

Syntax:

string-data.isalnum()

Note:

-----

when the string with only alpha or with only digits or with only alpha and digits, then isalnum() can return "True". Otherwise "False".

5) islower()

-------------

if the string is with only lower case alphabets, islower() can return "True". Otherwise it returns "False".

Syntax:

string-data.islower()

6) isupper():

------------

When the string with upper case alphabets

isupper() ===> True

otherwise:

isupper() ==> False

Syntax:

string-data.isupper()

7) isdigit():

-------------

if the string with digits:

isdigit() ==> True

otherwise:

isdigit() ==> False

Syntax:

string-data.isdigit()

username = "Ravi kumar"

un1 = "ravikumar"

un2 = "ravi112233"

un3 = "12345"

print(username.isalpha())

print(un1.isalpha())

print(un2.isalnum())

print(username.isalnum())

print(un1.isalnum())

print(un3.isalnum())

print(username.islower())

print(un1.islower())

print(un2.islower())

print(username.isupper())

print("RAVI".isupper())

print("RAVI@427".isupper())

print("8977029779".isdigit())

print("918977029779".isdigit())