Strings (Data type): It is a python Data type.

Strings are sequence of characters which are present inside a double or single quotes. (" ", ' ').

Here the characters can be

- alphabets (A-Z or a-z)
- numbers (0-9)
- special symbols (#, $, ₹, *, ␣, ...)

ex:- "ab7nH#"

'ab7nH#'

Properties of strings:

1) **Strings are immutable**

Mutable - whenever we are creating something and after the creation also we can change it or modify it then it is mutable.

Immutable - after creation if we cannot modify it or change it then it is immutable.

So once the strings are created they cannot be changed.

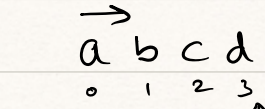
2) Inside a string each character **will have an Index value:-**

Index values are nothing but addresses of each character.

ex:- "abcd" → four characters a, b, c, d.

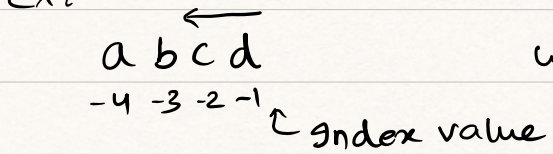
Posibive Index value: whenever we go from left to right the index value starts with '0' and up to '∞'.

ex:-



Negative Index value:- whenever we go from right to left the index value starts with '-1' and up to '-∞'.

ex:-



So two types of Index value

- positive Index value (+ve)
- Negative Index value (-ve)

3) **Accessing**: we can access individual character/ element in the string because they have different index value.

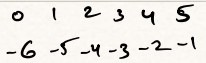
two types
- Indexing
- slicing

(a) **Single character accessing**

Indexing is a technique by using which we can access single character /element at a time.

here we use square brackets [Index]

ex:- Var1 = "abc12@"



So Var1[2]

↳ c (+ve indexing)

Var1[-4]

↳ c (-ve indexing)

x = "a 1 2 @ . * $ C 6 8"

0 1 2 3 4 5 6 7 8 9 ← +ve indexing

-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 ← -ve indexing

x[6]

↳ $

(+ve)

x[-4]

↳ $

(-ve)

+ve number is used for +ve indexing.

-ve number is used for -ve indexing.

* type() is an inbuilt function which can check if a given variable is a string or not.

(b) multiple element accessing: (sequential elements)

we can access multiple elements only when they are in a sequential order.

Slicing :- Slicing is a technique by using which we can access multiple sequential elements.

Square brackets with colon (:) inside:

[SI : EI]

SI → Starting index

EI → Ending index

* In slicing Ending index character is not included.

+ve indexing

ex:-

	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← -ve indexing
x = "	a	1	2	@	.	*	$	C	6	8 "	
+ve indexing →	0	1	2	3	4	5	6	7	8	9	

x[2:5]

Starting index ↙ ↘ Ending index

→ 2@. (stops before fetching '*' character of index 5)

x[6:]

→ blank represents the end of the string

→ $C68

-ve indexing: Python will go from left to right so for -ve indexing we have to come up with SI & EI.

ex:-

x[-8:-5]

→ 2@.

x[-4:]

→ here we leave blank because we cannot use '0'.
'0' is +ve indexing.

→ $C68

In slicing we can use step also

Var[SI:EI:step)

by default step is 1.

ex:-

	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← -ve indexing
x = "	a	1	2	@	.	*	$	C	6	8 "	
+ve indexing →	0	1	2	3	4	5	6	7	8	9	

x[0:5:2] → step

↙ Starting index

↘ Ending index

→ a2. (steps by 2)

x[: :-1] i.e x[-10: : -1]

→ blank represents the end of the string

→ 8 6 c $ * . @ 2 1 a

→ reverse of the string

(or)

```
for y in x[::-1]
    print(y)
```

```
8
6
c
$
*
.
@
2
1
a
```

4) **Empty string** : String is called an empty string when there is no character inside the double quotes or single quotes.

ex:- " " or ' ' → Empty strings

" ␣ " or ' ␣ ' → x Space is a special character.

Inbuilt function

3) **len( )** : length function

This is used to get how many elements or characters are there in a string.

```
ex:- var4 = "abcdef123@$"
     len(var4)
       ↳ 11
```

* Empty string when len( ) is equal to '0'.

Inbuilt functions

1) type( )
2) id( )
3) len( )
4) ord( )
5) bin( )
6) chr( )
7) print( )
8) input( )

Constructors

int( )
float( )
complex( )
str( )

String methods

These are the methods or functions which we can apply on strings.

To apply these methods we use '.' dot operator.

* dot operator will first check if the variable is string or not then only it will apply the string method.

1) lower() : lower method

This method will convert all the upper case letters in to lower case letters inside a string.

ex:- x = "ABC"

x.lower()

↳ "abc"

- here "abc" is a temporary result unless assigned to a different variable. The variable still retains original value.

So x is still "ABC"

- So strings are not being changed but the result is manipulated for execution purpose. Hence strings are immutable.

ex:- x = "ABCdef123"

x.lower()

↳ "abcdef123"

x = "ABCdef123"

So if we do

x = x.lower()

↳ "abcdef123"

x = "abcdef123" (here a new x string is created by assignment)

2) upper() : upper method

upper is a method which will convert all the lower case letters in to upper case letters.

ex:- x = "ABcdef123"

x.upper()

↳ "ABCDEF123"

3) capitalize() : capitalize method

This method will convert the first character of a string in to upper case

- if alphabet it will convert to upper case
- if number it will ignore and not do conversion to the string.

ex:- x = "abcd123"

x.capitalize()

↳ "Abcd123"

y = "1abcd123"

y.capitalize()

↳ "1abcd123"

↓
did not capitalize this

* capitalize will convert the first character to upper case and rest will retain their case.

ex:- x = "python Java oracle"

x.capitalize()

↳ "Python Java oracle"

4) swapcase(): swapcase method

This method will convert all the upper case letters to lower case and all the lower case letters to upper case inside the string.

ex:- x = "ABCdef123"

x.swapcase()

↳ "abcDEF123"

5) title(): title method

This method will convert the first character of each word in to upper case inside a string.

↳ space between characters creates words.

ex:- x = "python java oracle"

x.title()

↳ "Python Java Oracle"

x.upper()

↳ "PYTHON JAVA ORACLE"

x.capitalize()

↳ "Python java oracle"

* title() will convert first character in each word to upper case and all the rest of characters will be converted to lower case.

ex:- x = "pyTHon java Oracle"

x.title()

↳ "Python Java Oracle"

* title() unlike capitalize here the first character after the numericals and special characters will be converted.

ex:- x = "12python 23java 45oracle"

x.title()

↳ "12Python 23Java 45Oracle"

x.capitalize()

↳ "12python 23java 45oracle"

6) count() : count method

count ("sub-string")

Count is a method by using which we can count how many number of times a single character or a sequential characters are occuring inside the string.

ex:- x = "aaadfcaag"

x.count("a") ↳ 5

x.count("aa") ↳ 2

```
x.count("d")
↳ 1
```

Count using slicing :- count("sub-string", SI, EI)

SI → starting index, EI → Ending index

So we can do slicing to get exact count.

ex :- y = " Py Pydef Pysad "
(0 1 2 3 4 5 6 7 8 9 10 11)

```
y.count("Py")
↳ 3

y.count("Py", 0, 12)
↳ 3

y.count("Py", 0, 5)
↳ 2

y.count("Py", 0, 2)
↳ 1
```

7) **replace()** : replace method

replace(what we want to replace, by what we want to replace)

It is a string method by using which we can replace a single or sequential characters with other characters.

ex :- x = "abaca"

Now if we want to replace "a" in x variable with "@".

```
x.replace("a", "@")
↳ "@b@c@"
```

```
x.replace("ab","#")
  ↳ "#aca"
```

(or)

replace (what, by what, how many times)

```
x.replace("a","@",1)
  ↳ "@baca"
x.replace("a","@",2)
  ↳ "@b@ca"
```

8) **index():** index method (here +ve indexing is used)

index("Element of string")

It is a string method by using which we can find the index value of an element / character occurying at the first instance.

ex:-

```
x = "abaced"
    012345
x.index("a")
  ↳ 0
```

(from left to right we check index of first "a")

(or)

index("sub-string", SI, EI)

SI ↓ starting index; EI ↳ Ending index

```
x.index("a",1,)
  ↳ 2
```

↳ empty so till end of string.

* Substring - small part of a string.

* value error : If element is not found we get a value error.
* we can first use count() to check the no.of occurances.

```
x.count('a')
↳ 2
```

we can use for loop for instances like this.

9) find() : find method

find() method works same as index method. It is used to find the index value of a element/ character at the first occurance.

```
find ("substring", SI, EI)
```

* index() will give an error if the substring is not found.
* find() will not give an error instead will give '-1'. so here no error and code will not break. (Good)

ex:- x = " a b c d e f b s "
0 1 2 3 4 5 6 7

```
x.find("h")
↳ -1

x.index("h")
↳ error and code will break.
```

10) split( ) : split method

split method is a string method by using which we can split the string in to substrings.

Splitting needs a criteria to be specified as seperator.

Default seperator is space (␣).

ex:- x = "ab␣cd␣ef␣gh"

here if we give criteria as "space(␣)"

x.split("␣")

↳ "ab", "cd", "ef", "gh"

(or)

x.split(" ") → if nothing is mentioned default is space

↳ "ab", "cd", "ef", "gh"

y = "ab@cd@ef@gh"

y.split("@")

↳ "ab", "cd", "ef", "gh"

z = "abababaaba"

z.split("a")

↳ "", "b", "b", "b", "b", ""

↓ Empty string ↓ Empty string

* whenever the seperator is present at the starting or the ending of the string then our sub-strings will have starting or ending empty strings.

split("seperator", max.no.of times it should split)

ex:- x = " py ja py ja py "

x.split(" ", 2)

↳ " py", " ja", " py ja py"

11) strip() : Strip method

strip is a string method by using which we can remove or strip the element from starting and ending position only.

ex:- x = " abc "

x.strip(" ")

↳ "abc"

y = " a b c "

y.strip()

↳ " a b c"

* only starting and ending element will be stripped.
* default will be space ( ) to remove if nothing is mentioned.

12) rstrip() : right strip method

Here it goes from left to right and only removes the last instance.

y = " a b c "

y.rstrip()

↳ " a b c"

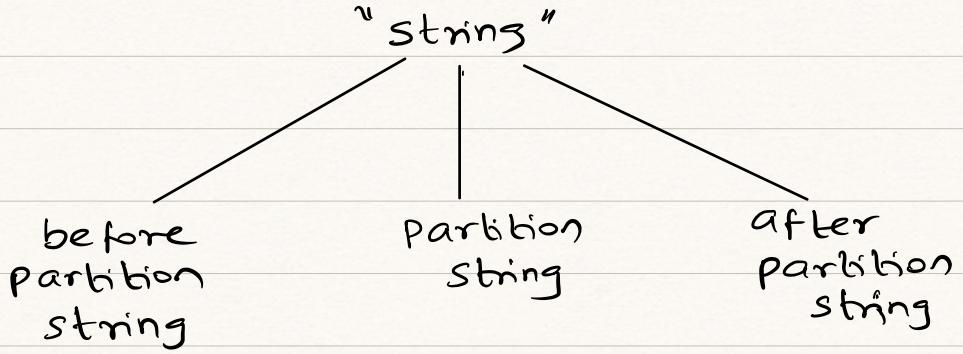
13) **lstrip()** : left strip method

Here it goes from right to left and only removes starting character.

```
y.lstrip()
  ↳ "a b c "
```

14) **partition()** Partition method (diff from split)

Partition is a string method by using which we can divide the string in to three sub-strings.



ex:- x = "abc"

```
x.partition("b")
  ↳ "a","b","c"
```

```
y = "abcbbac"
y.partition("b")
  ↳ "a","b","cbbac"
```

It will partition at the first instance and divide in to three parts.

15) maketrans( )

When we want to replace a single element or a sequential set of elements we use replace() method.

But if we want to replace non-sequential elements in a string with a new set of elements we use maketrans( ) method first followed by translate() method.

maketrans( ) method is a string method that will always create a mapping table for us.

ex:- x = "a1b2cdf"

x.maketrans("abc", "123")

Mapping table

a	1
b	2
c	3

first mapping table is created

Here we map the elements to be replaced with the new elements.

Here the output will be Ascii values mapped.

ex:- x = "abdcacbdefg"

x.maketrans("abc", "123")

↳ {97:49, 98:50, 99:51}

↑ Ascii values

ASCII value is a unique value given to each alphabet/symbols by python for the machine to understand.

Inbuilt function

4) **ord()** - will return the ASCII values of the alphabet/number/special character

5) **bin()** - will convert the integer or ASCII values in to binary value. i.e 0's & 1's.

6) **chr()** - will return character/number/special character from ASCII value.

ex:- x = "a"

ord("a")

↳ 97

bin(97)

↳ "0b1100001" ← 0b means binary value

chr(97)

↳ "a"

16) **translate()**

translate() method is used after maketrans() method. This method will take the mapping table created by maketrans() method and replace the old elements with the new elements based on the mapping table.

ex:- x = "abdcacbdefg"

x.maketrans("abc", "123")

↳ {97:49, 98:50, 99:51}

table = x.maketrans("abc", "123")

x.translate(table)

↳ "12d3132defg"

Concatenation: Combining two or more strings in to a single string is called concatenation. So we just use add (+).

ex:- x = "aa"

y = "bb"

z = "cc"

x+y+z

↳ "aabbcc"

but if one is numeric

ex:- x = 1

y = "aa"

x+y

↳ error (concatenation is not possible)

* when we want to concatenate a string with a numeric value we use a special method called format().

* In python the placeholder is defined as '{}'.

A placeholder is simply a variable that we will assign data to at a later date. i.e the data is input dynamically during the execution of the program.

17) format()

The format() method formats the specified values and inserts them inside the string's placeholder.

It takes any data seperated by the comma then

Step 1: formatting

Step 2: will put all data in the placeholder {}.

* format() method always returns a string output.

ex:-

```
name = "person 1"
age = 23
height = 169.7

bio_data = "my name is {} and my age is {} my
           height is {}"
bio_data.format(name, age, height)
                0     1    2
```

↳ "my name is person 1 and my age is 23 my height is 169.7"

(or we can write)

bio_data = "my age is {1} my name is {0}
and my height is {2}"

bio_data.format(name, age, height) ← (index's)
0 1 2

↳ "my age is 23 my name is
Person 1 and my height is 169.7"

This concludes the string manipulative methods.

Assignment 1

Create 6 variables
name, height, weight, age, phone_num, gender
and create a bio_data.

String check methods :

These methods check for conditions to see if it is True or False.

18) endswith()

It is a string method by using which we can check if the string is ending with a particular element or sequence of elements.

ex:- x = "abcd@"

x.endswith("@")
↳ True

x.endswith("d@")
↳ True

x.endswith("a")
↳ False

19) startswith()

It is a string method by using which we can check if the string is starting with a particular element or sequence of elements.

ex:- x = "abcd@"

x.startswith("ab")
↳ True

x.startswith("e")
↳ False

For startswith( ) and endswith( ) we can use SI & EI to check only a sub-string.

startswith ("sub-string", SI, EI)

endswith ("sub-string", SI, EI)

here SI & EI will help in getting a sub-string in which we will check for the condition

ex:- x = "abcd@"

x.startswith ("a", 1, 3) (substring "ab")

↳ True

x.endswith ("@", 3,)

↳ True

20) isupper( )

It is a string method where it will check if all the characters / Elements in the string are in the upper case.

- True if all are upper
- False if even one is smaller.

ex:- x = "ABCDEF"

x.isupper( )

↳ True

y = "ABdefXY"

y.isupper( )

↳ False

21) islower()

It is a string method where it will check if all the characters / Elements in the string are in the lower case.

- True if all are lower case
- False if even one is upper case

ex:- x = "abcdef"

x.islower()

↳ True

y = "aBcdef"

y.islower()

↳ False

* If we want to apply multiple methods make sure the output of first method is string.

ex:- x.lower().islower() ✓

x.count().islower() ✗

↳ this has integer output

22) istitle()

It is a string method that will check if first character of each word in the string is upper case or not. Rest of the elements need to be lower case.

ex:- x = "Alpha Beta Gamma"

x.istitle()

↳ True

```
y = "ALpha Beta Gamma"
y.istitle()
    ↳ False
z = "Alpha beta Gamma"
z.istitle()
    ↳ False
```

23) **isdigit()**

It is a string method that will check if all the elements in a string are numbers or not.

- True if all are numbers
- False if even one is not number

ex:-
```
x = "1234567"
x.isdigit()
    ↳ True
y = "123 567"
y.isdigit()
    ↳ False
z = "123a67"
z.isdigit()
    ↳ False
```

24) **isalpha()**

It is a string method that will check if all the elements in a string are alphabets (A-z, a-z)

- True if all are alphabets

└ False if even one is not alphabet

ex:- x = "ab AB cd CD"

x.isalpha()

↳ True

y = "ab1 AB_d"

y.isalpha()

↳ False

25) **isalnum()**

It is a string method that checks if all the Elements in the string are alphabets or numbers.

- True if all are alphabets or numbers
- False if it contains anything other than alphabets or numbers.

ex:- x = "abc1xyz"

x.isalnum()

↳ True

y = "aBC_d24"

y.isalnum()

↳ False (space is a special character)

z = "abc@ABC"

z.isalnum()

↳ False

26) isspace()

It is a string method that will check to see if all the elements are spaces or not.

- True if all are spaces
- False if anything but spaces.

ex:- x = " "

x.isspace()

↳ True

y = "a "

y.isspace()

↳ False

27) isidentifier()

This is a string method to check if the string can be used as an identifier. so this will check if the identifier followed all the naming conventions and rules.

ex:- x = "abcDe123"

x.isidentifier()

↳ True

y = "_xyz"

y.isidentifier() (space is special character)

↳ False

String manipulative methods

lower()
upper()
capitalize()
swapcase()
title()
count()
replace()
index()
find()
split()
strip()
rstrip()
lstrip()
partition()
maketrans()
translate()
format()

methods where indexing/slicing is used:

count("sub-str", SI, EI)
find("sub-str", SI, EI)
index("sub-str", SI, EI)
startswith("sub-str", SI, EI)
endswith("sub-str", SI, EI)

String checking methods

endswith( )

startswith( )

isupper( )

islower( )

istitle( )

isdigit( )

isalpha( )

isalnum( )

isspace( )

isidentifier( )

Inbuilt functions

1) Print()

Print() function is used to display the exact output returned by the machine.

If we do not use print function then

ex:- 1+1
2+2 } for these outputs are not printed
3+3
↳ 6

we can print(variable or data)

Print(1+1)
↳ 2

Print(2+2)
↳ 4

Print(3+3)
↳ 4

or Print(1+1, 2+2, 3+3)
↳ 2 4 6

* In python all the data is called as "Object".

In Print statement we can give n number of objects seperated by a comma or we can print any statement also.

ex:- x=1

```
y=2
z=3
print(x,y,z)
    ↳ 1 2 3
print(x+y*z)
    ↳ 7
```

Additional parameters

print(value(s), sep=seperator, end=end, file, flush)

default sep = space (" ")

default end = "\n" (new line insertion)

file = sys.stdout which is the file where print function writes all the output that is on the screen.

* we can change the seperator and end depending on how we want the output to be displayed.

ex:- print('hi', 'bye', 2)
↳ hi␣bye␣2

print('hi', 'bye', 2, sep=",")
↳ hi, bye, 2

print('hi') Same as print('hi', end="\n")

print('welcome') ⟶ print('welcome', end="\n")

print('bye') ⟶ print('bye', end="\n")

↳

hi

welcome

bye

```
print('hi', end="\t")
print('welcome', end="\t")
print('bye', end="\t")
```

↳ hi␣␣␣welcome␣␣␣bye

```
print('hi', 'welcome', sep=',', end='\t')
print('bye')
```

↳ hi,welcome␣␣␣bye

8) input()

If we create a variable and assign a static value to it then it will be a static variable. But instead if we want to give dynamic input we will use input() function.

```
x = input(" ")
```

ex:-

```
x = input("Type your name: ")
```

* Issue with input() function is it will always return string data.
* So if we want data in other data types we will need to use data type constructors where logically applicable.

Constructors

4) String Constructor : str( )

String constructor constructs a string from a integer, float, complex as well as string.

ex:- x=1

str(x)

↳ "1"

y=100.1

str(y)

↳ "100.1"

z = 2+3j

str(z)

↳ "2+3j"

* Can we convert string in to integer?
  A string can be converted in to integer if and only if all the elements are numbers.

* Conversion of string to other data types we have to check if logically possible.

Assignment 2:

Edit Assignment 1 code to take dynamic inputs.

Escape characters

There are some characters which we cannot insert inside the single or double quote.

ex:- single quote (') or double quote (")

Escape characters is a technique by using which we can insert characters that cannot be normally inserted in the string.

For this we use "\" (backward slash) followed by the character we want to insert.

\" → Escape character

ex:-

x = "ab""

↳ error

x = "ab\""

↳ ab\"

Print(x) (Print needs to be used to get exact output)

↳ ab"

Three important Escape characters;

\n → new line insertion

\t → tab space insertion

\b → back space insertion

} these do not insert n, t, b but have important meaning.

ex:-

```
x = "ab\nada"
```

↳ ab\nada

```
Print(x)
```

↳ "ab
ada"

} Print function gives the exact output

```
x = "ab\tada"
print (x)
```

↳ "ab␣␣␣␣ada"

```
x = "ab␣\bada"
print (x)
```

↳ "abada"

Boolean Data type

- True
- False

This data type has only two outputs True or False.

Machines cannot understand True or False.

So

bool() constructor

For numeric values

bool(0)
↳ False

bool(non-zero)
↳ True

bool(-1) → True
bool(+1) → True

For strings ← empty string

bool("")
↳ False

bool("....") (non-empty string)
↳ True

others ← special condition.

bool(None) → False
bool(True) → True
bool(False) → False