Python String Manipulation

String Literals

A collection of characters which are surrounded by either single quotation marks or double quotation marks are known as String in Python. The sequence of characters may include alphabets, numbers, special characters, white spaces and backslash.

Note: Strings are Immutable (Unchangeable)

'hello' is the same as "hello".

‘Pin 333001’ is the same as “Pin 333001”

‘abc.kvs@gov.in’ is the same as “abc.kvs@gov.in”

You can display a string literal with the `print()` function:

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Creating String

A string can be create as similar to other variable creation in Python. For creation of String we just assign a string value to a variable name followed by an equal sign.

```
S=”Hello KVS”

Print(s)

Msg=”Who was Developed \”Python\” ?”

Print(Msg)
```

Output: Who was Developed” Python” ?

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Output:

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Creating an Empty String

1. **An empty string can be created by assigning "" OR '' to a variable.**

   s="" OR s=''

   Here s is string type Variable.

   Print(s) Output: ''

   An empty string can be created by str() constructor also.

   s=str()

   Print(s) Output: ''

Creating a String from Existing Sequence
Creating a String using str() Constructor

A string can be created from other sequence or by using str() constructor.

Example:

s=str(258)

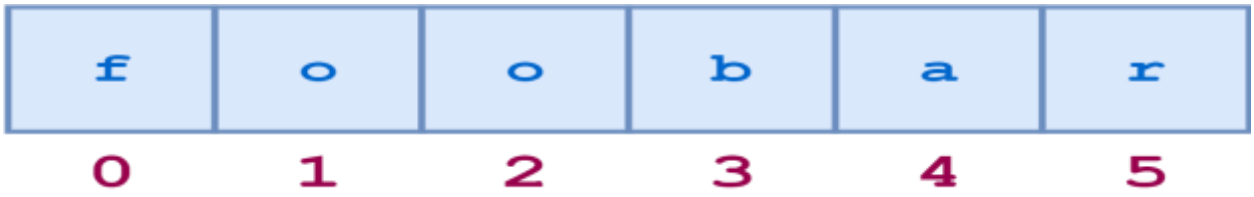
Print(s) Output: '258'

Indexing a String

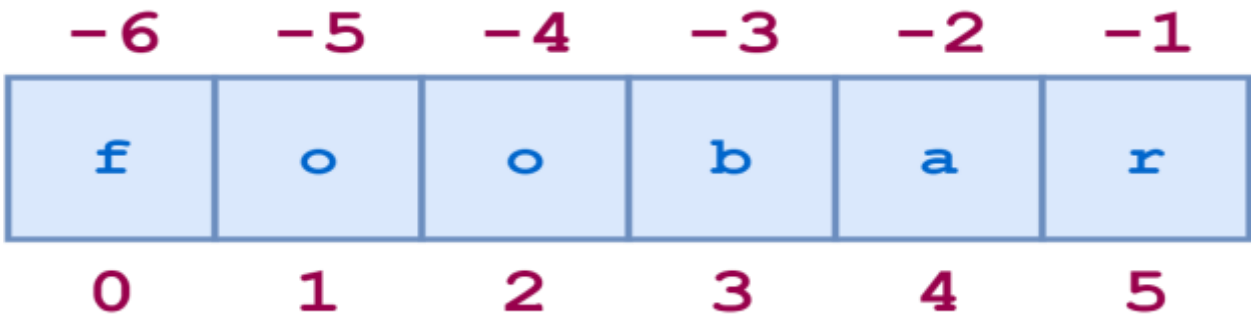
In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets (`[]`).

String indexing in Python is zero-based: the first character in the string has index `0`, the next has index `1`, and so on. The index of the last character will be the length of the string minus one.

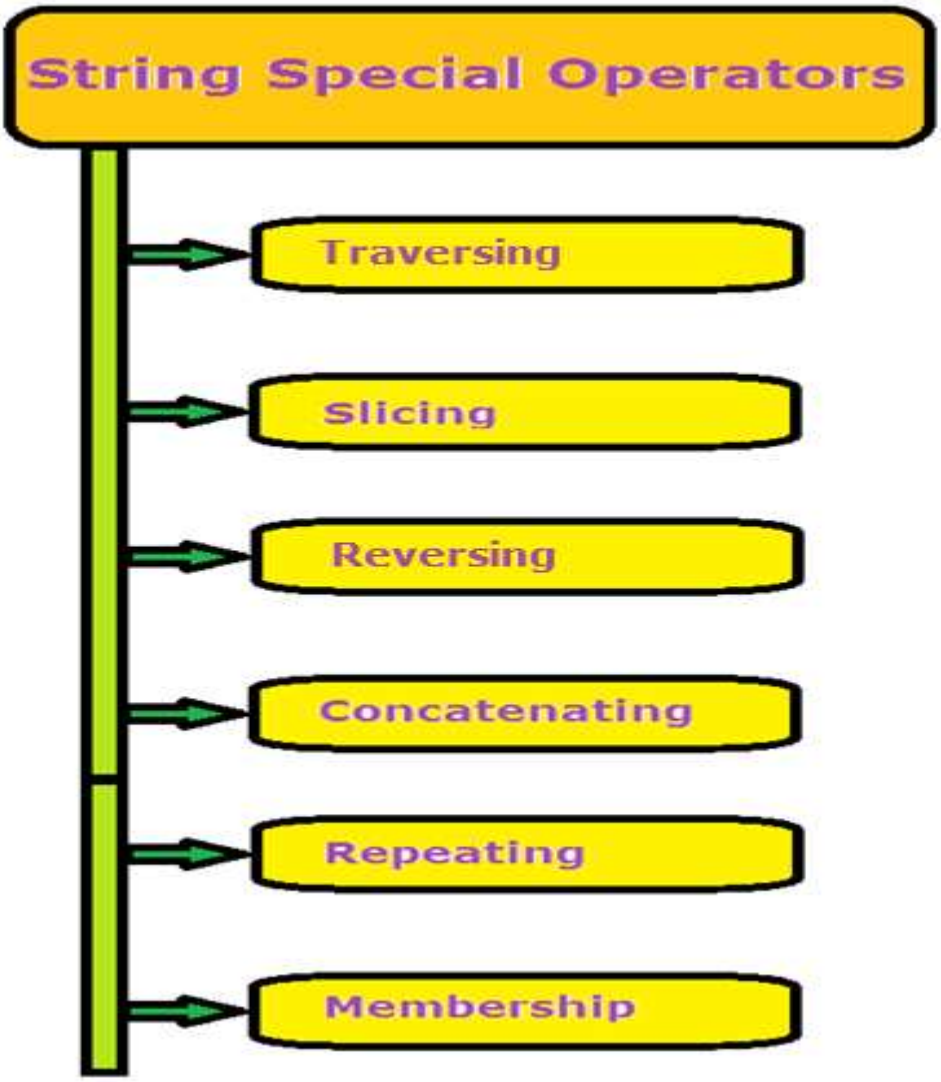
For example, a schematic diagram of the indices of the string `'foobar'` would look like this:



String indices can also be specified with negative numbers, in which case indexing occurs from the end of the string backward: `-1` refers to the last character, `-2` the second-to-last character, and so on. Here is the same diagram showing both the positive and negative indices into the string `'foobar'`:



Special String Operators



Traversing a String

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Get the character at position 1 (remember that the first character has the position 0):

```
B= “Hello, World!”
Print( B[1] )                    # Output:  e
Print( B[-2] )                   # Output: d
```

Display All String using loop

```
B= “Hello, World!”
for ch in B:
    print(ch, end=””)        # Output:   Hello, World!
```

String Slicing

Python also allows a form of indexing syntax that extracts substrings from a string, known as string slicing.

Format of string slicing:
String_Variable [Start Index: End Index: Step Value]
Where start, end index and step value separated with colon (:)
Start index is by default Zero (0)
End Index is by default end of string.
Step Value is by default 1
Note:

1. **Start index always include in result**
2. **End index (if given) always exclude in result**
3. **End index (if not given) then last index of string include**

Get the characters from position 2 to position 5 (not included):

b = "Hello, World!" print(b[2:5])	llo

Use negative indexes to start the slice from the end of the string:

Get the characters from position 5 to position 1, starting the count from the end of the string:

b = "Hello, World!" print(b[-5:-2])	Orl
print(b[2:8:2])	low
print(b[2:8:-1])	No Out Put
print(b[::2])	HloWrd
print(b[5:0:-1])	olle
print(b[10:0:-2])	drWol
print(b[::-1])	!dlroW olleH
print(b[::-2])	!lo olH

Reversing String

A string in Python can be reversed by using following ways.

1. **By using Traversing (Negative index Traversing)**
2. **By using loop**

```
Str=”Indian Army”
print(str[-1::-1])
```

Output: ymrA naidnI

By Using Loop:

```
str=”Indian Army”
size=len(str)
size=Size-1
While (size>=0):
    print(str[size], end=””)
    size=size-1
```

Output: ymrA naidnI

Concatenating String

Concatenating string mean joining two or more strings with the help of + operator and make a new string.

Example:

A=”Hello”

B=”Jhunjhunu”

New_Str=A + B

Print(New_Str)

Output: HelloJhunjhunu

New_Str=A + “ “ + B

Print(New_Str)

Output: Hello Jhunjhunu

New_Str= “Hello”+ 100

Output: It will give error as

Unsupported operand type(s) for +: int and str

Replicating String

The replicating operator * is used to repeating multiple times of a string to create multiple copies of given string.

Example:

A=”Hello”

Print(A*3)

Output: HelloHelloHello

S = 3 * “Hello-“

Print(S)

Output: Hello-Hello-Hello-

Membership Operator in String

Membership operator is Boolean type operator. It used to check whether a sub string is present in main string or not.. There are two membership operators are used in string.

1. **in operator: If sub string is present in main string then it will return True otherwise it will return False result.**
2. **not in operator: If sub string is not present in main string then it will return True otherwise it will return False result.**

Example:

Main_str= “This is main string”

Sub_str=”ain”

Result=Sub_str in Main_str

Print(Result)

Output: True

Print(“Ain” in Main_str) # Note A is Capital in Ain.

Output: False

Print(“Ain” not in Main_str)

Output: True

Print(“ain” not in Main_str)

Output: False

Relational / Comparison Operator

The relational operators of Python can also apply on strings also. The relational operators are also the Boolean operators and produce either True or False result.

String comparison is different from numeric values comparison. Python compares the Strings using their ASCII (American Standard Code for Information Interchange). Python compares the ASCII values of first letter of each string, if they are equal then next elements of string compare and so on, until it finds differ ASCII Values.

Letter	ASCII Value
A	65
B	66
Z	90
a	97
a	98
z	122

In Python by ord('character') function, the ASCII value of any character can be find. Similarly chr(ASCII Value) can used to find Character of given ASCII value.

Example:

S1="Jhunjhunu"

S2="Jhunjhunu"

Print(S1 ==S2) **Output: True**

Print(S1 < S2) **Output: False**

Print(S1 >=S2) **Output: True**

Print(S1 > S2) **Output: False**

Print(S1 != S2) **Output: False**

Strings are Immutable

Strings are immutable in nature in Python. Its mean the content of string cannot be changed once it is created. In other words, the assignment operator does not work to change the content of string.

Example:

Str=”Simple”

Str[1]=”a”

Print(str)

Output: TypeError: 'str' object does not support item assignment

Strings Built in Functions

Note: **All string methods returns new values. They do not change the original string.**

The `len()` function returns the length of a string: a="Hello, World!" print(len(a))	13
The `strip()` method removes any whitespace from the beginning or the end: a = " Hello, World! " print(a.strip()) # returns "Hello, World!"	Hello, World!
`lower()` / upper( ) a = "Hello, World!" print(a.lower()) print(a.upper())	hello, world! HELLO, WORLD!
The `replace()` method replaces a string with another string: a = "Hello, World!" print(a.replace("H", "J"))	Jello, World!
The `split()` method splits the string into substrings if it finds instances of the separator: a = "Hello, World!" print(a.split(",")) # returns ['Hello', ' World!']	['Hello', ' World!']

Check String

To check if a certain phrase or character is present in a string, we can use the keywords `in` or `not in`.

Check if the phrase "ain" is present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)

x = "ain" not in txt
print(x)
```

True

False

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
c = a + " " + b
print(c)
```

HelloWorld

Hello World

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

output: TypeError: must be str, not int

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

output: My name is John, and I am 36

quantity = 3 itemno = 567 price = 49.95 myorder = "I want {} pieces of item {} for {} dollars." print(myorder.format(quantity, itemno, price))	I want 3 pieces of item 567 for 49.95 dollars.
You can use index numbers {0} to be sure the arguments are placed in the correct placeholders: quantity = 3 itemno = 567 price = 49.95 myorder = "I want to pay {2} dollars for {0} pieces of item {1}." print(myorder.format(quantity, itemno, price))	I want to pay 49.95 dollars for 3 pieces of item 567

Method	Description
capitalize()	Converts the first character to upper case and rest in lower case. str=" Fit India is hit India" print("str.capitalize()=",str.capitalize()) >> Fit india is hit india
casefold()	Converts string into lower case str=" Fit India is hit India" print("str.casefold()=",str.casefold()) >> fit india is hit india
center()	Returns a centered string Str=”kvsjjn” Print(str.center(10) >> kvsjjn (set 10 character length and center the str) Print(str.center(10,”$”) >>$$kvsjjn$$
count()	Returns the number of times a specified value occurs in a string Str= “fit India is hit India” Print(str.count(“i”)) >> 5
endswith()	Returns true if the string ends with the specified value txt = "Hello, welcome to my world." x = txt.endswith("my world.") print(x) >>True
find()	Searches the string for a specified value and returns the index of where it was found(first occurance). It not found then returns -1 txt = "Hello, welcome to my world." x = txt.find("welcome") print(x) >> 7

format()	**Formats specified values in a string** **age=100** **nm="PL Adwani"** **msg="Hello, I am {} is {} year old."** **str=msg.format(nm,age)** **print(str)** **>> Hello, I am PL Adwani is 100 year old.**

index()	**Searches the string for a specified value and returns the index of where it was found. If string not found then generate "ValueError: substring not found"**
	`txt = "Hello, welcome to my world."`
	`x = txt.index("welcome")`
	`print(x)`
	`>> 7`

isalnum()	**Returns True if all characters in the string are alphanumeric**
isalpha()	**Returns True if all characters in the string are in the alphabet**
isdecimal()	**Returns True if all characters in the string are decimals**
isdigit()	**Returns True if all characters in the string are digits / numeric**
isidentifier()	**Returns True if the string is an identifier otherwise False** **str="kvs_jjn"** **print(str.isidentifier())** **>>True**
islower()	**Returns True if all characters in the string are lower case**
isnumeric()	**Returns True if all characters in the string are numeric / digit**
isspace()	**Returns True if all characters in the string are whitespaces**
istitle()	**Returns True if the string follows the rules of a title** **str="Kvs jjn"** **print(str.istitle())** **>> False** **str="Kvs Jjn"** **print(str.istitle())** **>> True**
isupper()	**Returns True if all characters in the string are upper case**
lower()	**Converts a string into lower case**

lstrip()	**Returns a left trim version of the string**
	`txt = "     banana     "`
	`x = txt.lstrip()`
	`print("In all fruits", x, "is my favorite")`
	`>> In all fruits banana     is my favorite`

replace()	**Returns a string where a specified value is replaced with a specified value**

	txt = "I like bananas" x = txt.replace("bananas", "apples") print(x) print(txt) >> I like apples I like bananas x = txt.replace("a", "apples") **print(x)** **>> I like bapplesnapplesnappless**
rsplit()	**Splits the string at the specified separator, and returns a list**
rstrip()	**Returns a right trim version of the string**
split()	**Splits the string at the specified separator, and returns a list**
strip()	**Returns a trimmed version of the string**
title()	**Converts the first character of each word to upper case**
upper()	**Converts a string into upper case**

Assignments:

1. **Write a program to find reverse of string.**
2. **WAP to check that string is palindrome or not.**
3. **WAP to find number of words in string.**
4. **WAP to find number of digits in string.**
5. **WAP to find number of alphabets in string.**
6. **WAP to print special characters in string.**
7. **WAP to find Uppercase (Capital) letters in string.**
8. **WAP to find Lowercase (Small) letters in string.**
9. **WAP to replace all lowercase in uppercase and vice versa.**
10. **WAP to count number of articles (a,e,I,o,u) used in string.**
11. **WAP to count word “this” in string.**
12. **WAP to print alternate characters of string.**
13. **WAP to print only those words which start with ‘t’.**
14. **WAP to print only those words which end with ‘y’.**
15. **WAP to replace the word “India” with “Bharat”.**