Strings & String Functions

What are Strings ?

- A **String** is a series / sequence of Unicode characters enclosed within double/single/triple quotation marks to represent **Strings** and **Strings** are immutable.
- A String may contain letters,digits,and various special characters such as +,-,*,/ and $,string constants also referred as string literals.

Declaring & Initializing String Variables

```
>>> s1 = "'This is a string one“
>>> s2 = ‘This is a string two’
>>> s3=“  #empty string
>>> type(s1)                <class 'str'>
```

- Multi-line strings can be denoted using triple quotes, ''' or """.
- Even triple quotes can be used in Python but generally used to represent **multiline strings** or **docstrings**.

```
>>> s4 = '''a multiline
            string"‘
>>>
```

How to access characters in String

- We can access individual characters using indexing and a range of characters using slicing operator **[ ]** or **The Subscript Operator**
- Index starts from 0. Trying to access a character out of index range will raise an IndexError.
- The index must be an integer. We can't use float or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a string by using the slicing operator with colon **[:]**.
- String='HELLO'

0	1	2	3	4
H	E	L	L	O
-5	-4	-3	-2	-1

```
s = 'Hello world'
print('s = ', s)

#first character
print('s[0] = ', s[0])

#last character
print('s[-1] = ', s[-1])

#slicing 2nd to 5th character
print('s[1:5] = ', s[1:5])

#slicing 6th to 2nd last character
print('s[5:-2] = ', s[5:-2])
```

```
s = Hello world
s[0] = H
s[-1] = d
s[1:5] = ello
s[5:-2] =  wor
```

If we try to access index out of the range or use decimal number, we will get errors.

```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range

# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

How to change or delete a string?

- Strings are immutable. This means that elements of a string cannot be changed once it has been assigned.
- We can simply reassign different strings to the same name.

```
my_string = 'Hello world'
my_string[5]='j'

Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    my_string[5]='j'
TypeError: 'str' object does not support item assignment
```

- We cannot delete or remove characters from string ,deleting the string is possible use the keyword **del**.

```
my_string = 'Hello world'
del my_string[5]

TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

Python String Operations

- There are many operations that can be performed with string which makes it one of the most used datatypes in Python.

Concatenation of Two or More Strings

- Joining of two or more strings into a single one is called concatenation.
- The + operator does this in Python. Simply writing two string literals together also concatenates them.
- The * operator can be used to repeat the string for a given number of times.

```
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

- Writing two string literals together also concatenates them like + operator.
- If we want to concatenate strings in different lines, we can use parentheses.

- Writing two string literals together also concatenates them like + operator.
- If we want to concatenate strings in different lines, we can use parentheses.

```
>>> # two string literals together
>>> 'Hello ''World!'
'Hello World!'

>>> # using parentheses
>>> s = ('Hello '
...      'World')
>>> s
'Hello World'
```

Iterating Through String

- Using **for** loop we can iterate through a string. Here is an example to count the number of 'l' in a string.

```
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count,'letters found')
```

String Membership Test

- We can test if a substring is exists within a string or not , for that we have to use **in** keyword.

```
print('r' in 'rgukt')# True

print('he' not in 'hello')#False
```

- **How to print reverse string using for loop?**

```
s1='Hello' # Assign String here
s2='' #empty string
l=0 # length
for i in s1:
    l=l+1
#finaly legth of the string l=5
for i in range(1,l+1):
    s2+=s1[l-i]
print("The Reverse String of ",s1,"is",s2)
```

Python String Formatting

Escape Sequence

If we want to print a text like - **He said, "What's there?“**

we can neither use single quote or double quotes.

This will result into SyntaxError as the text itself contains both single and double quotes.

```
print("He said, "What's there?"")
...
SyntaxError: invalid syntax
>>> print('He said, "What's there?"')
...
SyntaxError: invalid syntax
```

- One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences.

- An escape sequence starts with a backslash and is interpreted differently. If we use single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes.
- Here is how it can be done to represent the above text

```
# using triple quotes
print('''He said, "What's there?"''')

# escaping single quotes
print('He said, "What\'s there?"')

# escaping double quotes
print("He said, \"What's there?\"")
```

- We can even format strings with % Operator

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

list of all the escape sequence

Here are some examples

```
>>> print("C:\\Python32\\Lib")
C:\Python32\Lib

>>> print("This is printed\nin two lines")
This is printed
in two lines

>>> print("This is \x48\x45\x58 representation")
This is HEX representation
```

Escape Sequence	Description
\newline	Backslash and newline ignored
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

Raw String to ignore escape sequence

- Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place **r** or **R** in front of the string.
- This will imply that it is a raw string and any escape sequence inside it will be ignored.

```
>>> print("This is \x61 \ngood example")
This is a
good example
>>> print(r"This is \x61 \ngood example")
This is \x61 \ngood example
```

The format() Method for Formatting Strings

- The format() method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
- We can use positional arguments or keyword arguments to specify the order.

```
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

Built-in functions to Work with Python

- Various built-in functions that work with sequence, works with string as well.
- The **enumerate()** function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.
- The **len()** function returns the length (number of characters) of the string.

```
s='RGUKT'
#enumerate()
list_enumerate=list(enumerate(s))
print('list(enumerate(s))=',list_enumerate)
#list(enumerate(s))= [(0,'R'),(1,'G'),(2,'U'),(3,'K'),(4,'T')]

#character count
print('length of the string =',len(s))
#length of the string = 5
```

- The method **center()** makes str centred by taking *width* parameter into account. Padding is specified by parameter ***fillchar***. Default filler is a space.

  Syntax : - str.center(width[, fillchar])

```
str = "RGUKT IIIT"
str1= str.center(30,'a')
str2= str.center(30)
print(str1)
print (str2 )
```

- The method **count()** returns the number of occurrence of Python string *substr* in string *str*. By using parameter *start* and *end* you can give slice of *str* .
- Return Int Value

  Syntax : - str.count(substr [, start [, end]])

```
str = "This is a count example"
sub = "i"
print(str.count(sub, 4, len(str)))
sub = "a"
print (str.count(sub) )
```

- This method **endswith()** returns True if the string ends with the specified substring, otherwise return False ,
- Return Bool Value
- The use of *start* and *end* to generate slice of Python string *str*.

Syntax :- str.endswith(suffix[, start[, end]])

```
str = "RGUKT IIIT RKVALLEY"
sub="IIIT"
l=len(str)
print(l)
print(str.endswith(sub,2,6))
print(str.endswith(sub,10))
sub='sdfs'
print(str.endswith(sub))
```

- The **find()** method used to find out whether a string occur in given string or its substrings.
- Return the lowest index in S where substring sub is found
- If given Python string is found, then the **find()** method returns its index.

  If Python string is not found then -1 would be returned.

Syntax : - str.find(str, beg=0 end=len(string))

```
str = "RGUKT IIIT RKVALLEY"
sub1="IIIT"
sub2='RKV'
print(str.find(sub1))
print(str.find(sub1,20))
print(str.find(sub2))
```

- The method **isalnum()** is used to determine whether the Python string consists of alphanumeric characters, false otherwise

  Syntax :- str.isalnum()
- The method **isalpha()** return true if the Python string contains only alphabetic character(s), false otherwise.

  Syntax :- str.isalpha()
- The method **isdigit()** return true if the Python string contains only digit(s),false otherwise.

  Syntax :- str.isdigit()
- The method **islower()** return true if the Python string contains only lower cased character(s), false otherwise

  Syntax :- str.isdigit()

```
s='rgukt1234'
print(s.isalnum())
s1='#$@#%'
print(s1.isalnum())
s2='PinCode'
print(s2.isalpha())
s3='rgukt1234'
print(s3.isalnum())
s4='1234'
print(s3.isdigit())
s5='rgukt1234'
print(s5.isdigit())
```

```
s='rguktrkv'
print(s.islower())
s1='pin343223'
print(s1.islower())
s2='PinCode'
print(s2.islower())
```

- The method **isspace()** return true
  if the Python string contains only
  white space(s).
  Syntax :- str.isspace()
- The method **istitle()** return true
  if the string is a titlecased
  Syntax :- str.istitle()
- The method **isupper()** return true if the string
  ontains only upper cased character(s),
  false otherwise.
  Sytax:- str.isupper()
- The method **ljust()**, it returns the string left justified.
  Total length of string is defined in first parameter
  of method *width* . Padding is done as defined
  in second parameter *fillchar* .(default is space)

```
s=" "
print(s.isspace())

s1="pin 2342"
print(s1.isspace())

s2="rgukt rkv"
print(s2.istitle())

s3="Rgukt rkv"
print(s3.istitle())

s4="Rgukt Rkv"
print(s4.istitle())

s5="RGUKT"
print(s5.isupper())

s6="RGUKT123"
print(s6.isupper())

s7="rGUKT"
print(s7.isupper())

s="rgukt"
print(s.ljust(10,"k"))
print(s.ljust(10))
print(s.ljust(3,"k"))
```

Syntax : - **str.ljust(width[, fillchar])**

- In above example you can see that if you don't define the *fillchar* then the method **ljust()** automatically take space as *fillchar*.
- The method **rjust()**, it returns the string right justified.

  Total length of string is defined in first parameter of method *width* . Padding is done as defined in second parameter *fillchar* .(default is space)

  Syntax:- **str.rjust(width[, fillchar])**

```
s="rgukt"
print(s.rjust(10,'j')) #jjjjjrgukt
print(s.rjust(10)) #'     rgukt'
print(s.rjust(3,'k')) #rgukt
```

- This function **capitalize()** first letter of string.

  Syntax:- **str.capitalize()**

```
s="rgukt rkv"
print(s.capitalize())
```

- The method **lower()** returns a copy of the string in which all case-based characters have been converted to lower case.

  Syntax:- **str.lower() ,str.upper()**

```
s="RGUKT Rkv 124"
print(s.lower())
print(s.upper())
print(s)
```

- The method **upper()** returns a copy of the string in which all case-based characters have been converted to upper case.

- The method **title()** returns a copy of the string in which first character of all words of string are capitalised.

  Syntax:- **str.title()**

- The method **swapcase()** returns a copy of the string in which all cased based character swap their case

  Syntax:- **str.swapcase()**

- This method **join()** returns a string which is the concatenation of given sequence and string as shown in example.

  seq = it contains the seqeunce of separated strings.
  str = it is the string which is used to replace the separator of seqcence

  Syntax:- **str.join(seq)**

```
s="rgukt rkv"
print(s.capitalize())

s="rgukt rkv 124"
print(s.title())
print(s)

s="RgukT rKv 124"
print(s.swapcase())
print(s)

s="RgukT", "rKv", "124"
print('$'.join(s))
```

- The method **lstrip()** returns a copy of the string in which specified char(s) have been stripped from left side of string. If char is not specified then space is taken as default.

  Syntax : - **str.lstrip([chars])**
- The method **rstrip()** returns a copy of the string in which specified char(s) have been stripped from right side of string. If char is not specified then space is taken as default.

  Syntax : - **str.rstrip([chars])**
- The method **strip()** returns a copy of the string in which specified char(s) have been stripped from both side of string. If char is not specified then space is taken as default.

  Syntax : - **str.strip([chars])**

```
s="    hello rgukt nuz      "
print(s.lstrip())
s="$$$$$$hello rgukt nuz$$$$$"
print(s.lstrip('$'))

s="    hello rgukt nuz      "
print(s.rstrip())
s="$$$$$$hello rgukt nuz$$$$$"
print(s.rstrip('$'))

s="    hello rgukt nuz      "
print(s.strip())
s="$$$$$$hello rgukt nuz$$$$$"
print(s.strip('$'))
```

- The method **splits** returns a list of all words in the string, delimiter separates the words. If delimiter is not specified then whitespace is taken as delimiter, paramter *num*

  Syntax :- str.split("delimiter", num)

- The method **max()** returns the max character from string *str* according to ASCII value.

  in first print statement y is max character, because ASCII code of "y" is 121. In second print statement "s" is max character, ASCII code of "s" is 115.

  Syntax :- max(str)

- The method **min()** returns the min character from string *str* according to ASCII value.

  Syntax :- min(str)

```
s="RGUKT-RKV RGUKT-RKV"
print(s.split())
print(s.split('-',1))

s="RGUKT-RKVRGUKT-RKV"
print(max(s))

print(min(s))
```

- The method **replace()** returns the string in which occurrences of string specified by parameter *old* have been replaced with string specified by parameter *new* .
- The parameter max defined how many occurrences have been replaced. If *max* is not specified then all occurrences will be replaced.

  Syntax :- str.replace(old, new[, max])
- The method **splitlines()** returns a list with all the lines in string, In *num* is specified then it would include line break. *num* greater than 1 it include line break.

  num -- This is any number, if present then it would be assumed that line breaks need to be included in the lines.

  Syntax :- str.splitlines(num)

```
s="My Name is Apple"
print(s.replace("Apple","Mango"))#My Name is Mango
print(s.replace(" ","-",3))#My-Name-is-Apple
```

```
s="My Name is Apple"
print(s.splitlines(0))

s="My\nName\nis\nApple"
print(s.splitlines(1))
print(s.splitlines(2))
print(s.splitlines())
```

- The method **startswith()** return true if a string *str* starts with the string specified by parameter str1.
- parameter *beg* and *end* are used to slice the string *str* .

Syntax :- str.startswith(str1, beg=0,end=len(string));

```
s="My Name is Apple"
print(s.startswith('My'))#True
print(s.startswith("Name",3,8))#True
print(s.startswith('is',4,len(s)))#False
```

- This method pads the string on the left with zeros to fill width.

Syntax :- str.zfill(width)

```
s="My Name is Apple"
print(s.zfill(20))#0000My Name is Apple
print(s)#My Name is Apple
```

- The **ord()** function returns ASCII code of the character.

```
ch='A'
print(ord(ch)) # 65
```

- The **chr()** function returns character represented by a ASCII number.

```
num=90
print(chr(num)) # Z
```

- **Write a program to print Alphabet using ASCII Numbers ?**

```
ch='A'
print(ord(ch)) # A=65,a=97

ch='Z'
print(ord(ch)) # Z=90,z=122

for i in range(65,91):
    print(chr(i),end=" ")
#A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
print()
for i in range(97,123):
    print(chr(i),end=" ")
#a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Comparing Strings

- Python allows you to compare strings using relational(or comparison) operator such as **>, <, < =, < =**,etc.
- **==** if two strings are equal ,it returns True → 'Abc' =='Abc' →True
- **!= or <>** if two strings are not equal, it returns True → 'AbC' != 'ABC' →True
- **>** if first string is greater than the second ,it returns True → "abc">"Abc" →True
- **<** if second string is greater than the first ,it returns True → "abC"<"abc" →True
- **>=** if first string is greater than or equal to the second ,it returns True → "aBC">="ABC"→True
- **<=** if second string is greater than or equal to the first ,it returns True → "ABc"<="ABc"→True

Note:

- These operators compare the strings by using the lexicographical order i.e using ASCII values of the characters.
- The ASCII values of A-Z is 65 -90 and ASCII code for a-z is 97-122 .

Practice Problems on Strings

- Write a program to find number of digits ,alphabets and symbols ?
- Write a program to convert lower case to upper case from given string?
- Write a program to print the following output?

  R

  R G

  R G U

  R G U K

  R G U K T

  R G U K

  R G U

  R G

  R

- Write a program to check whether given string is palindrome or not? Radar,liril,
- Write a program to find no_ words,no_letters,no_digits and no_blanks in a line?
- Write a program to sort list names in alphabetical order?

- To find the first character from given string ,count the number of times repeated and replaced with * except first character then print final string?
- To find the strings in a list which are matched with first character equals to last character in a string?
- Write a program that accepts a string from user and redisplays the same string after removing vowels from it?
- This is a Python Program to take in two strings and display the larger string without using built-in functions.?
- Python Program to Read a List of Words and Return the Length of the Longest One?
- Python Program to Calculate the Number of Upper Case Letters and Lower Case Letters in a String ?