Chapter-4: Python Strings Revisited

Concatenating, Appending and Multiplying Strings

- In python string data type is a sequence made up of one or more individual characters, where a character could be a letter, digit, whitespace or any other symbol.
- Python treats strings as contiguous series of characters delimited by single, double or even triple quotes.
- Python has a built-in string class named "str" that has many useful features.
- In python we can simultaneously declare and define string by creating a variable of string type.

Concatenating: The word concatenate means to join together and python allows operator (+) to concatenate two variables.

Source code

```
str1 = "hello"
str2 = "World"
str3 = str1 + str2
print("The conctenated string is:",str3)
```

Output

```
The conctenated string is: helloWorld
```

Appending: append mean to add something at the end. In python we can add one string at the end of another string using the (+=) operator

Source code

```
str = "hello"
name = input("Enter your name:")
str += name
str += "\nWelcome to python programming"
print(str)
```

Output

```
Enter your name:shivaraj
helloshivaraj
```

Welcome to python programming

Multiplying or Repeating: python supports printing same variable output multiple times with the help of (*) operator

Source code

```
str = "hello "
print(str * 3)
```

Output

hello hello hello

Stings are Immutable:

- Python strings are immutable which means once they are created they cannot be changed.
- Whenever you try to modify an existing string variable, a new string is created.
- Every object in python is stored in memory and you can find out whether two variables are referring same object or not by using id () function.
- The id () function returns the memory address of that object.

Example: Program to demonstrate string references using the id() function

Source code

```
str1 = "hello"
print("str1 is:", str1)
print("ID of str1 is:",id(str1))
print("================================================"
)
str2= "World"
print("str2 is:", str2)
print("Id of str1 is:",id(str2))
str1 += str2
print("================================================"
)
print("str1 after concatenation is:",str1)
print("id of str1 is:", id(str1))
```

```
print("================================================"
)
str3 =str1
print("str3=", str3)
print("id of str3 is:", id(str3))
print("================================================"
)
```

Output

```
str1 is: hello
ID of str1 is: 1940537870408
================================================
str2 is: World
Id of str1 is: 1940537759480
================================================
str1 after concatenation is: helloWorld
id of str1 is: 1940537896048
================================================
str3= helloWorld
id of str3 is: 1940537896048
```

String Formatting Operator:

The format operator (%) allows users to constructs strings, replacing parts of the strings with the data stored in the variables.

Syntax:

"<FORMAT>" % (<VALUES>)

- The statement begins with a format string consisting of a sequence of characters and conversion specifications.
- Conversion specifications start with a (%) operator and can appear anywhere within the string.
- Following the format string is a (%) sign and then a set of values, one per conversion specification, separated by commas and enclosed in parenthesis.
- If there is a single value then parenthesis is optional.

Example: Program to use format sequence while printing a string

Source code

```
name = "Shivaraj"
age = 30

print("Name is %s and Age is = %d" %(name,age))
print("Name is %s and Age is = %d" %("Shruthi",27))
```

Output

```
Name is Shivaraj and Age is = 30
Name is Shruthi and Age is = 27
```

Formatting Symbols:

Format Symbol	Purpose
%c	Character
%d or %i	Signed decimal integer
%s	String
%u	Unsigned decimal integer
%o	Octal integer
%x or %X	Hexadecimal integer
%e or %E	Exponential notation
%f	Floating point notation
%g or %G	Short numbers in floating point or exponential notation

Example: Program to display powers of a number using formatting characters

Source Code

```
i = 1
print("%-4s%-5s%-6s%-8s%-13s%-15s%-17s%-19s%-21s%-23s" % \
    ('i', 'i**2','i**3','i**4','i**5','i**6','i**7','i**8','i**9','i**10'))

while i <=10:
   print("%-4d%-5d%-6d%-8d%-13d%-15d%-17d%-19d%-21d%-23d" %
    (i,i**2,i**3,i**4,i**5,i**6,i**7,i**8,i**9,i**10))
   i += 1
```

Output

```
i   i**2 i**3  i**4   i**5    i**6     i**7      i**8       i**9        i**10
1   1    1     1      1       1        1         1          1           1
2   4    8     16     32      64       128       256        512         1024
3   9    27    81     243     729      2187      6561       19683       59049
4   16   64    256    1024    4096     16384     65536      262144      1048576
5   25   125   625    3125    15625    78125     390625     1953125     9765625
6   36   216   1296   7776    46656    279936    1679616    10077696    60466176
7   49   343   2401   16807   117649   823543    5764801    40353607    282475249
8   64   512   4096   32768   262144   2097152   16777216   134217728   1073741824
9   81   729   6561   59049   531441   4782969   43046721   387420489   3486784401
10  100  1000  10000  100000  1000000  10000000  100000000  1000000000  10000000000
```

- In the above code, we have set the width of each column independently using the string formatting feature of python.
- The (-) minus after each (%) in the conversion string indicates left justification.
- The numerical values specify the minimum length.
- The %-15d means it is a left justified number that is at least 15 characters wide.

Built-in string methods and functions:

- Strings are an example of python objects.
- An object is an entity that contains both data as well as functions to manipulate that data.
- The functions are available to any instance of the object.
- Python supports built-in methods to manipulate strings.
- A method is just like a function and only difference between a function and method is that a method is invoked or called an object.
- If the variable is str is a string then you can call the upper () method as str.upper () function to convert all the character of str in upper case.

Function	Usage	Example
capitalize()	This function is used to capitalize first letter of the string	str = "hello" print(str.capitalize()) output: Hello
center(width,fillchar)	Returns a string with the original string centered to a total of width columns and filled with char in columns that do not have characters	str = "hello" print(str.center(10,'*')) output: **hello***
isalnum()	Returns true if string has at least 1 character and every character is	message = "JamesBond007" print(message.isalnum()) output:True

	either a number or an alphabet and false otherwise	
isalpha()	Returns true if string has at least 1 character and every character is an alphabet and false otherwise	message = "JamesBond007" print(message.isalpha()) Output:False
isdigit()	Returns true if string contains only digits and false otherwise	message = "007" print(message.isdigit()) output:True
islower()	Returns true if string has a at least 1 character and every character is a lowercase alphabet and false otherwise	message = "Hello" print(message.islower()) Output:False
len(string)	Returns the length of the string	message = "Hello" print(len(message)) Output:5
strip()	Removes all leading and trailing whitespaces in string	message = "abcHelloabc" print(message) print(message.strip('abc')) Output: abcHelloabc Hello
enumerate(str)	Returns an enumerate object lists the index and value of all the characters in the strings as pairs	message = "Hello World" print(message) print(list(enumerate(message))) Output: Hello World [(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'W'), (7, 'o'), (8, 'r'), (9, 'l'), (10, 'd')]
title()	Returns string in title case	message = "Hello World" print(message) print(message.title()) Output: Hello World Hello World
replace(old,new[,max])	Replaces all or max occurrences of old in string with new	message = "Hello World " print(message) print(message.replace("He","Sh")) Output:

		Hello World Shllo World
swapcase()	Toggles the case of every character(uppercase character becomes lowercase and viceversa)	message = "The World Is Full Of Ego People And We Should Be Always Careful" print(message) print(message.swapcase()) Output: The World Is Full Of Ego People And We Should Be Always Careful tHE wORLD iS fULL oF eGO pEOPLE aND wE sHOULD bE aLWAYS cAREFUL

Slice operation:

- A substring of a string is called a slice.
- The slice operation is used to refer to sub-parts of sequences and strings.
- You can take subset of string from the original string by using [ ] operator also known as slice operation.
- In python to know or to understand slice operation you should know the concept f positive and negative indexing which is shown below

```
P       Y       T       H       O       N

0       1       2       3       4       5       #INDEX FROM THE START

-6      -5      -4      -3      -2      -1      #INDEX FORM END
```

- The syntax of slice operation is s [start: end], where start specifies the beginning index of substring and end -1 is the index of the last character.

Example: program to demonstrate slice operation on string objects

Source code

```
str = "python"

#default to the entire string
print(str[:])
```

```
#default to the start of the string
print("str[:6]=",str[:6])

#characters starting at index 1and extending up to but not including index
5
print("str[1:5]=",str[1:5])

#default to the end of the string
print("str[1:]=",str[1:])

#an index that is too big is truncated down to the length of the string
print("str[1:20]=",str[1:20])
```

Output

```
python
str[:6]= python
str[1:5]= ytho
str[1:]= ython
str[1:20]= ython
```

- Python gives the flexibility to either access a string from first character or from the last character.
- If we access the string from the first character then we use a zero based index but when doing it backward the index starts with -1.

Example: program to demonstrate how characters in a string are accessed using negative index

Source code

```
str = "python"

#default to the entire string
print(str[:])

#last character is accessed
```

```
print("str[-1]=",str[-1])

#first charcter is accessed
print("str[-6]=",str[-6])

#second last and the last character is accessed
print("str[-2]=",str[-2])

#characters from second upto second last are accessed
print("str[-5:-2]=",str[-5:-2])
```

Output

```
python
str[-1]= n
str[-6]= p
str[-2]= o
str[-5:-2]= yth
```

Specifying stride while slicing strings:

In the slice operation you can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string.

The default value of stride is 1.

Example: program to use slice operation with stride

Source code

```
str = "Welcome to the world of python"

#default to the entire string
print(str[:])

#default stride is 1
print("str[2:10]=",str[2:10])

#same as stride 1
print("str[2:10:1]=",str[2:10:1])
```

```
#skips every alternate character
print("str[2:10:2]=",str[2:10:2])

#skips every fourth character
print("str[2:10:3]=",str[2:10:3])
```

Output

```
Welcome to the world of python
str[2:10]= lcome to
str[2:10:1]= lcome to
str[2:10:2]= loet
str[2:10:3]= lmt
```

Example with different slice operation with stride:

Source code

```
str = "Welcome to the world of python"

#default to the entire string
print(str[:])

#slice operation with just last(positive)argument
print("str[::3]=",str[::3])

#Program to print slice operation with just last(negative)argument
print("str[::-1]=",str[::-1])

#Program to print the string in reverse order thereby skipping every second character
print("str[::-2]=",str[::-2])

#Program to print the string in reverse order thereby skipping every third character
print("str[::-3]=",str[::-3])
```

Output

Welcome to the world of python
str[::3]= Wceohwloph
str[::-1]= nohtyp fo dlrow eht ot emocleW
str[::-2]= nhy odrweto mce
str[::-3]= nt r ttml

Regular Expression:

- A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression.
- Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression.
- Regular expressions can contain both special and ordinary characters. Most ordinary characters, like 'A', 'a', or '0', are the simplest regular expressions; they simply match themselves.
- Regular expressions are used for various string manipulations.
- Regular expressions are used as a special text string that is used for describing a search pattern to extract information from text such as code, files, log, spreadsheets and documents.
- Regular expressions are a special sequence of characters that helps to match or find strings in another string.
- Regular expressions can be accessed using the **re** module which comes as part of the standard library.

Some of the important methods of re module are

- match() function
- search() function
- sub() function
- findall() function
- finditr() function
- flag options

Match () function:

- The match function is used to match the RE pattern to string with optional flags.
- In this method, the expression "w+" and "\W" will match the words starting with letter 's' and thereafter, anything which is not started with 'g' is not identified.
- To check match for each element in the list or string, we run the for loop.

- Syntax: re.match(patterns,string,flags=0)

Example: program to demonstrate the use of match() function

Source code

```
import re
list = ["shivaraj sdm", "sameed sdm", "ashwith sdm"]
for element in list:
    z = re.match("(s\w+)\W(s\w+)", element)
    if z:
        print((z.groups()))
```

Output

```
('shivaraj', 'sdm')
('sameed', 'sdm')
```

Search () function:

- A regular expression is commonly used to search for a pattern in a text. This method takes a regular expression pattern and a string and searches for that pattern with the string.
- To use search () function, you need to import re first and then execute the code.
- The search() function takes the "pattern" and "text" to scan from our main string and returns a match object when the pattern is found or else not match if the pattern is not found.
- Syntax: re.search(pattern, string,flags=0)

Example: program to demonstrate the use of search() function

Source code

```
import re

patterns = ['python testing','bvoc19']
text = 'python testing is fun?'

for pattern in patterns:
    print('looking for "%s" in "%s" ->' % (pattern, text), end='')
```

```
if re.search(pattern,text):
        print('found a match!')
else:
        print('no match')
```

Output

```
looking for "python testing" in "python testing is fun?" ->found a match!
looking for "bvoc19" in "python testing is fun?" ->no match
```

- Here we look for two literal strings 'python testing','bvoc19', in a text string 'python testing is fun?'
- for 'python testing' we found the match hence it returns the output as "found a match", while for word 'bvoc19'we could not found in string hence it returns the output as "No match".

Findall () function:

- findall () module is used to search a string and returns list of matches of the pattern in the string. If no match is found, then the returned list is empty.
- syntax: matchlist.re.findall(pattern,input_str,flags=0)

Source code

```
abc = 'shine@google.com,naukri@hotmail.com, monster@yahoomail.com'
emails = re.findall(r'[\w\.-]+@[\w\.-]+', abc)
for email in emails:
   print(email)
```

Output

```
shine@google.com
naukri@hotmail.com
monster@yahoomail.com
```

Finditer () function:

- Return an iterator yielding Match Object instances over all non-overlapping matches for the RE pattern in string. The string is scanned left-to-right, and

matches are returned in the order found. Empty matches are included in the result.

Source code

```
abc = 'shine@google.com,naukri@hotmail.com,monster@yahoomail.com'
emails = re.finditer(r'[\w\.-]+@[\w\.-]+', abc)
for email in emails:
   print("match found at starting index:",email.start())
   print("match found at ending index:",email.end())
   print("match found at starting and ending index:",email.span())
```

Output

```
match found at starting index: 0
match found at ending index: 16
match found at starting and ending index: (0, 16)
match found at starting index: 17
match found at ending index: 35
match found at starting and ending index: (17, 35)
match found at starting index: 36
match found at ending index: 57
match found at starting and ending index: (36, 57)
```

Source code

```
import re
s1 = 'Blue Berries'
pattern = 'Blue Berries'
for match in re.finditer(pattern, s1):
   s = match.start()
   e = match.end()
   print ('String match "%s" at %d:%d' % (s1[s:e], s, e))
```

Output

```
String match "Blue Berries" at 0:12
```

Sub () function:

- The sub () function in the re module can be used to search a pattern in the string and replace it with another pattern.
- Syntax: re,sub(pattern,repl,string,max=0)
- The sub () function replaces all occurrences of the pattern in string with repl, substituting all occurrences unless any max value is provided and this method returns a modified string.

Source code

```
import re
string = 'she sells sea shells on the sea shore'
pattern = 'sea'
repl = 'ocean'
new_string = re.sub(pattern,repl,string,1)
print(new_string)
```

Output

```
she sells ocean shells on the ocean shore
```

Flag Options:

- The search (), findall () and match () functions of the module take options to modify the behavior of the pattern match (i.e. in the syntax of search (), findall () and match () functions they have used flags options and these flag options are as below).
- Flag variable is used as a signal in programming to let the program know that a certain condition has met. It usually acts as a Boolean variable indicating a condition to be either true or false.

Flags options	Description
re.I or re.IGNORECASE	Ignores case of characters, so “match”,”MATCH”,”mAtCh”, i.e. all are same. And Performs case-insensitive matching.
re.S or re.DOTALL	Enables dot(.) to match newline character. By default, dot matches any character other than the newline character.
re.M or re.MULTILINE	Makes the ^ and $ to match the start and end of each line. i.e. it matches even after and before line breaks in the string. By default, ^ and $ matches the start and end of the whole string.
re.L or	Makes the flag \w to match all characters that are considered letters

re.LOCALE	in the given current local string.
re.U or re.UNICODE	Treats all letters from all scripts as word characters.

re.I

re.IGNORECASE

Perform case-insensitive matching; expressions like [A-Z] will match lowercase letters, too. This is not affected by the current locale. To get this effect on non-ASCII Unicode characters such as ü and Ü, add the UNICODE flag.

re.S

re.DOTALL

Make the '.' special character match any character at all, including a newline; without this flag, '.' will match anything except a newline.

re.M

re.MULTILINE

When specified, the pattern character '^' matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character '$' matches at the end of the string and at the end of each line (immediately preceding each newline). By default, '^' matches only at the beginning of the string, and '$' only at the end of the string and immediately before the newline (if any) at the end of the string.

re.L

re.LOCALE

Make \w, \W, \b, \B, \s and \S dependent on the current locale.

re.U

re.UNICODE

Make the \w, \W, \b, \B, \d, \D, \s and \S sequences dependent on the Unicode character properties database. Also enables non-ASCII matching for IGNORECASE.

You can also find more resource @ below links:

https://www.tutorialspoint.com/python/python_reg_expressions.htm

https://docs.python.org/2/library/re.html

Programs:

Program to extract an email address from the given text

Source code

```
import re
pattern = r"[\w.-]+@[\w.-]+"
string = "Send your feedback at feedback@ujire.com"
match = re.search(pattern, string)

if match:
   print("Email to:", match.group())
else:
   print("No Match")
```

Output

Email to: feedback@ujire.com

Hint: if question is asked to find multiple address, use the re.finall() method instead of re.search() to extract all email address.

Program to extract each character from a string using regular expression

Source Code

```
import re
result = re.findall(r'.','Keep Going Good')
print(result)
```

Output

```
['K', 'e', 'e', 'p', ' ', 'G', 'o', 'i', 'n', 'g', ' ', 'G', 'o', 'o', 'd']
```

Program to extract each word from a string using regular expression

Source Code

```
import re
result = re.findall(r'\w+','Keep Going Good')
print(result)
```

Output: ['Keep', 'Going', 'Good']

Program to extract first word from a string using regular expression

Source Code

```
import re
result = re.findall(r'^\w+','Keep Going Good')
print(result)
```

Output

```
['Keep']
```

Program to extract last word from a string using regular expression

Source Code

```
import re
result = re.findall(r'\w+$','Keep Going Good')
print(result)
```

Output

['Good']

Program to extract pair of words from a string using regular expression

Source Code

```
import re
result = re.findall(r'\w\w','Keep Going Good')
print(result)
```

Output

['Ke', 'ep', 'Go', 'in', 'Go', 'od']

Program to extract first two characters from each word from a string using regular expression

Source Code

```
import re
result = re.findall(r'\b\w\w','Keep Going Good')
print(result)
```

Output

['Ke', 'Go', 'Go']

Program to extract complete date format from a given string using regular expression

Source Code

```
import re
result = re.findall(r'\d{2}-\d{2}-\d{4}','Hello,my name is shiva and my date of joining is 19-09-2019')
print("Date of Joining is:",result)
```

Output

Date of Joining is: ['19-09-2019']

Program to extract only current date, month and year from a given string using regular expression

Source Code

```
import re
result = re.findall(r'(\d{2})-\d{2}-\d{4}','Hello,my name is shiva and my date of joining is 19-09-2019')
print("Date of Joining is:",result)

#if you want only month you ca use paraenthsis as shown in the below code
#import re
result = re.findall(r'\d{2}-(\d{2})-\d{4}','Hello,my name is shiva and my date of joining is 19-09-2019')
print("Date of Joining is:",result)

#import re
result = re.findall(r'\d{2}-\d{2}-(\d{4})','Hello,my name is shiva and my date of joining is 19-09-2019')
print("Date of Joining is:",result)
```

Output

```
Date of Joining is: ['19']
Date of Joining is: ['09']
Date of Joining is: ['2019']
```

Program to extract only words that starts from vowel from a given string using regular expression

Source Code

```
import re
result = re.findall(r'\b[aeiouAEIOU]\w+','Hello,my name is shiva and my date of joining is 19-09-2019')
print(result)
```

Output: ['is', 'and', 'of', 'is']

Program to extract a mobile phone number and number should start with 7,8,9 from a given string using regular expression

Source Code

```
import re
list = ['9986123456','9986456789','9986456123','9740123456','995646525']
for i in list:
    result = re.findall(r'[7-9]{1}[0-9]{9}',i)
    if result:
        print(result)
```

Output

```
['9986123456']
['9986456789']
['9986456123']
['9740123456']
```

Write a program that uses a regular expression to pluralize a word

Source Code

```
import re
def pluralize(noun):
    if re.search('[sxz]$', noun):
        return re.sub('$','es',noun)
    elif re.search('[aeioudgkprt]h$',noun):
        return re.sub('$','es',noun)
    elif re.search('[^aeiou]y$',noun):
        return re.sub('y$','ies',noun)
    else:
        return noun + 's'

list = ["bush","fox","toy","cap"]
for i in list:
    print(i,'-',pluralize(i))
```

Output

bush - bushs
fox - foxes
toy - toys
cap - caps