Chapter 1

Strings

After studying this lesson, students will be able to:

- *Learn how Python inputs strings*
- *Understand how Python stores and uses strings*
- *Perform slicing operations on strings*
- *Traverse strings with a loop*
- *Compare strings and substrings*
- *Understand the concept of immutable strings*
- *Understanding string functions.*
- *Understanding string constants*

Introduction

In python, consecutive sequence of characters is known as a string. An individual character in a string is accessed using a subscript (index). The subscript should always be an integer (positive or negative). A subscript starts from 0.

Example

```
# Declaring a string in python
>>>myfirst="Save Earth"
>>>print myfirst
Save Earth
```

Let's play with subscripts

To access the **first character** of the string

```
>>>print myfirst[0]
S
```

To access the **fourth character** of the string

```
>>>print myfirst[3]
e
```

To access the **last character** of the string

```
>>>print myfirst[-1]
>>h
```

To access the **third last character** of the string

```
>>>print myfirst[-3]
r
```

Consider the given figure

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

Important points about accessing elements in the strings using subscripts

- Positive subscript helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.

  Example : A[0] or A[-5] will display 'H'
- Subscript 1 or –ve (n-1) displays the second element.

Note: Python does not support character data type. A string of size 1 can be treated as characters.

Creating and initializing strings

A literal/constant value to a string can be assigned using a single quotes, double quotes or triple quotes.

✡ Enclosing the string in single quotes

Example

```
>>>print ('A friend in need is a friend indeed')
A friend in need is a friend indeed
```

Example

```
>>>print(' This book belongs to Raghav\'s sister')
This book belongs to Raghav's sister
```

As shown in example 2, to include the single quote within the string it should be preceded by a backslash.

✡ Enclosing the string in double quotes

Example

```
>>>print("A room without books is like a body without a soul.")
A room without books is like a body without a soul.
```

✡ Enclosing the string in triple quote

Example

```
>>>life="""\" Live as if you were to die tomorrow.
      Learn as if you were to live forever.\"
      ---- Mahatma Gandhi """
>>> print life
      " Live as if you were to die tomorrow.
    Learn as if you were to live forever."
    ---- Mahatma Gandhi """
```

Triple quotes are used when the text is multiline.

In the above example, backslash (\) is used as an escape sequence. An escape sequences is nothing but a special character that has a specific function. As shown above, backslash (\) is used to escape the double quote.

Escape sequence	Meaning	Example
\n	New line	>>> print "Hot\nCold" Hot Cold
	Tab space	>>>print "Hot\tCold" Hot Cold

✡ By invoking raw_input() method

Let's understand the working of raw input() function

Example

```
>>>raw_input()
Right to education
'Right to education'
```

As soon as the interpreter encounters raw_input method, it waits for the user to key in the input from a standard input device (keyboard) and press Enter key. The input is converted to a string and displayed on the screen.

Note: raw_input( ) method has been already discussed in previous chapter in detail.

✡ By invoking input() method

Example

```
>>>str=input("Enter the string")
Enter the string hello
NameError: name 'hello' is not defined
```

Python interpreter was not able associate appropriate data type with the entered data. So a NameError is shown. The error can be rectified by enclosing the given input i.e. hello in quotes as shown below

>>>str=input("Enter the String") Enter the String "hello" >>> print str Hello	>>>str=input("Enter the String") Enter the String'hello' >>> print str hello

Strings are immutable

Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example.

Example

```
>>>str='honesty'
>>>str[2]='p'
TypeError: 'str' object does not support item assignment
```

Python does not allowthe programmer to change a character in a string. As shown in the above example, str has the value 'honesty'. An attempt to replace 'n' in the string by 'p' displays a TypeError.

Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

String traversal using for loop	String traversal using while loop
A='Welcome' >>>for i in A: print i W	A='Welcome' >>>i=0 >>>while i<len(A) print A[i]

e l c o m e	i=i+1 W e l c o m e
A is assigned a string literal 'Welcome'. On execution of the for loop, the characters in the string are printed till the end of the string is not reached.	A is assigned a string literal 'Welcome' i is assigned value 0 The len() function calculates the length of the string. On entering the while loop, the interpreter checks the condition. If the condition is true, it enters the loop. The first character in the string is displayed. The value i is incremented by 1. The loop continues till value i is less than len-1. The loop finishes as soon as the value of I becomes equal to len-1, the loop

Strings Operations

Operator	Description	Example
+ (Concatenation)	The + operator joins the text on both sides of the operator	>>> 'Save'+'Earth' 'Save Earth' To give a white space between the two words, insert a space before the closing single quote of the first literal.
* (Repetition)	The * operator repeats the	>>>3*'Save Earth '

	string on the left hand side times the value on right hand side.	'Save Earth Save Earth Save Earth '
in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters.	>>>A='Save Earth' >>> 'S' in A True >>>'Save' in A True >>'SE' in A False
not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of **in** operator discussed above)	>>>'SE' not in 'Save Earth' True >>>'Save ' not in 'Save Earth' False
range (*start*, *stop*[, *step*])	**This function is already discussed in previous chapter.**	
Slice[n:m]	The Slice[n : m] operator extracts sub parts from the strings.	>>>A='Save Earth' >>> print A[1:3] av The print statement prints the substring starting from subscript 1 and ending at subscript 3 but not including subscript 3

More on string Slicing

Consider the given figure

String A	S	A	V	E		E	A	R	T	H
Positive Index	0	1	2	3	4	5	6	7	8	9
Negative Index	-10	-9	-9	-7	-6	-5	-4	-3	-2	-1

Let's understand Slicing in strings with the help of few examples.

Example

>>>A='Save Earth'

>>> print A[1:3]

av

The print statement prints the substring starting from subscript 1 and ending at subscript 3 .

Example

>>>print A[3:]

'e Earth'

Omitting the second index, directs the python interpreter to extract the substring till the end of the string

Example

>>>print A[:3]

Sav

Omitting the first index, directs the python interpreter to extract the substring before the second index starting from the beginning.

Example

>>>print A[:]

'Save Earth'

Omitting both the indices, directs the python interpreter to extract the entire string starting from 0 till the last index

Example

>>>print A[-2:]

'th'

For negative indices the python interpreter counts from the right side (also shown above). So the last two letters are printed.

Example

>>>Print A[:-2]

'Save Ear'

Omitting the first index, directs the python interpreter to start extracting the substring form the beginning. Since the negative index indicates slicing from the end of the string. So the entire string except the last two letters is printed.

Note: Comparing strings using relational operators has already been discussed in the previous chapter

String methods & built in functions

Syntax	Description	Example
len()	Returns the length of the string.	>>>A='Save Earth' >>> print len(A) >>>10
capitalize()	Returns the exact copy of the string with the first letter in	>>>str='welcome'

	upper case	>>>print str.capitalize() Welcome
find(*sub*[, *start*[, *end*]])	The function is used to search the first occurrence of the substring in the given string. It returns the index at which the substring starts. It returns -1 if the substring does occur in the string.	>>>str='mammals' >>>str.find('ma') 0 **On omitting the start parameters, the function starts the search from the beginning.** >>>str.find('ma',2) 3 >>>str.find('ma',2,4) -1 **Displays -1 because the substring could not be found between the index 2 and 4-1** >>>str.find('ma',2,5) 3
isalnum()	Returns True if the string contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc.	>>>str='Save Earth' >>>str.isalnum() False The function returns False as space is an alphanumeric character. >>>'Save1Earth'.isalnum() True

isalpha()	Returns True if the string contains only letters. Otherwise return False.	>>> 'Click123'.isalpha()
		False
		>>> 'python'.isalpha()
		True

isdigit()	Returns True if the string	>>>print str.isdigit()

	contains only numbers. Otherwise it returns False.	false
lower()	Returns the exact copy of the string with all the letters in lowercase.	>>>print str.lower() 'save earth'
islower()	Returns True if the string is in lowercase.	>>>print str.islower() True
isupper()	Returns True if the string is in uppercase.	>>>print str.isupper() False
upper()	Returns the exact copy of the string with all letters in uppercase.	>>>print str.upper() WELCOME

lstrip()	Returns the string after removing the space(s) on the left of the string.	>>> print str
		Save Earth
		>>>str.lstrip()
		'Save Earth'
		>>>str='Teach India Movement'
		>>> print str.lstrip("T")
		each India Movement
		>>> print str.lstrip("Te")
		ach India Movement
		>>> print str.lstrip("Pt")
		Teach India Movement
		If a string is passed as argument to the lstrip() function, it removes those characters from the left of the string.

rstrip()	Returns the string after removing the space(s) on the	>>>str='Teach India Movement' >>> print str.rstrip()

	right of the string.	Teach India Movement

isspace()	Returns True if the string contains only white spaces and False even if it contains one character.	>>> str=' '
		>>> print str.isspace()
		True
		>>> str='p'
		>>> print str.isspace()
		False

istitle()	Returns True if the string is title cased. Otherwise returns False	>>> str='The Green Revolution'
		>>> str.istitle()
		True
		>>> str='The green revolution'
		>>> str.istitle()
		False

replace(old, new)	The function replaces all the occurrences of the old string with the new string	>>>str='hello'
		>>> print str.replace('l','%')
		He%%o
		>>> print str.replace('l','%%')
		he%%%%o

join ()	Returns a string in which the string elements have been joined by a separator.	>>> str1=('jan', 'feb' ,'mar')
		>>>str='&"
		>>> str.join(str1)
		'jan&feb&mar'

swapcase()	Returns the string with case changes	>>> str='UPPER'
		>>> print str.swapcase()
		upper
		>>> str='lower'
		>>> print str.swapcase()

		LOWER

partition(sep)	The function partitions the strings at the first occurrence of separator, and returns the strings partition in three parts i.e. before the separator, the separator itself, and the part after the separator. If the separator is not found, returns the string itself, followed by two empty strings	>>> str='The Green Revolution'
		>>> str.partition('Rev')
		('The Green ', 'Rev', 'olution')
		>>> str.partition('pe')
		('The Green Revolution', '', '')
		>>> str.partition('e')
		('Th', 'e', ' Green Revolution')

split([sep[, maxsplit]])	The function splits the string into substrings using the separator. The second argument is optional and its default value is zero. If an integer value N is given for the second argument, the string is split in N+1 strings.	>>>str='The$earth$is$what$we$all $have$in$common.'
		>>> str.split($,3)
		SyntaxError: invalid syntax
		>>> str.split('$',3)
		['The', 'earth', 'is', 'whatweall$have$in$common.']
		>>> str.split('$')
		['The', 'earth', 'is', 'what', 'we', 'all', 'have', 'in', 'common.']
		>>> str.split('e')
		['Th', ' Gr', '', 'n R', 'volution']
		>>> str.split('e',2)
		['Th', ' Gr', 'en Revolution']

Note: In the table given above, len( ) is a built in function and so we don't need import the string module. For all other functions ***import string*** statement is required for their successful execution.

Let's discuss some interesting strings constants defined in string module:

string.ascii_uppercase

The command displays a string containing uppercase characters.

E**xamp**le

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.ascii_lowercase

The command displays a string containing all lowercase characters.

Example

```
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

string.ascii_letters

The command displays a string containing both uppercase and lowercase characters.

```
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.digits

The command displays a string containing digits.

```
>>> string.digits
'0123456789'
```

string.hexdigits

The command displays a string containing hexadecimal characters.

```
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

string.octdigits

The command displays a string containing octal characters.

```
>>> string.octdigits

'01234567'
```

string.punctuations

The command displays a string containing all the punctuation characters.

```
>>> string.punctuations

'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}-'
```

string.whitespace

The command displays a string containing all ASCII characters that are considered whitespace. This includes the characters space, tab, linefeed, return, formfeed, and vertical tab.

```
>>> string.whitespace

'\t\n\x0b\x0c\r '
```

string.printable

The command displays a string containing all characters which are considered printable like letters, digits, punctuations and whitespaces.

```
>>> string.printable

'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!
"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}- \t\n\r\x0b\x0c'
```

Note: Import string module to get the desired results with the commands mentioned above.

Programs using string functions and operators

1. Program to check whether the string is a palindrome or not.

```
defpalin():
    str=input("Enter the String")
    l=len(str)
    p=l-1
    index=0
    while (index<p):
        if(str[index]==str[p]):
            index=index+1
            p=p-1
    else:
        print "String is not a palidrome"
        break
    else:
    print "String is a Palidrome"
```

2. Program to count no of 'p' in the string pineapple.

```
def lettercount():
    word = 'pineapple'
    count = 0
    for letter in word:
        if letter == 'p':
        count = count + 1
    print(count)
```

Regular expressions and Pattern matching

A regular expression is a sequence of letters and some special characters (also called meta characters). These special characters have symbolic meaning. The sequence formed by using meta characters and letters can be used to represent a group of patterns.

Let's start by understanding some meta characters.

For example

str= "Ram$"

The pattern "Ram$" is known as a regular expression. The expression has the meta character '$'. Meta character '$' is used to match the given regular expression at the end of the string. So the regular expression would match the string 'SitaRam' or 'HeyRam' but will not match the string 'Raman'.

Consider the following codes:

```
def find():
    import re
    string1='SitaRam'
    if re.search('Ram$',string1):
        print "String Found"
    else :
        print" No Match"
```

Output:

String Found

```
def find():
    import re
    string1='SitaRam'
    if re.search('Sita$',string1):
        print "String Found"
    else :
        print" No Match"
```

Output

No Match

As shown in the above examples, Regular expressions can be used in python for matching a particular pattern by importing the re module.

Note: re module includes functions for working on regular expression.

Now let's learn how the meta characters are used to form regular expressions.

S.No	Meta character	Usage	Example
1	[ ]	Used to match a set of characters.	[ram] The regular expression would match any of the characters r, a, or m. [a-z] The regular expression would match only lowercase characters.
2	^	Used to complementing a set of characters	[^ram] The regular expression would match any other characters than r, a or m.
3	$	Used to match the end of string only	Ram$ The regular expression would match Ram in SitaRam but will not match Ram in Raman
4	*	Used to specify that the previous character can be matched zero or more times.	wate*r The regular expression would match strings like watr, wateer, wateeer and so on.
5	+	Used to specify that the previous character can be matched one or more times.	wate+r The regular expression would match strings like water, wateer, wateeer and so on.

6	?	Used to specify that the previous character can be matched either once or zero times	wate?r The regular expression would only match strings like watr or water
7	{ }	The curly brackets accept two integer value s. The first value specifies the minimum no of occurrences and second value specifies the maximum of occurrences	wate{1,4}r The regular expression would match only strings water, wateer, wateeer or wateeeer

Let's learn about few functions from re module

re.compile()

The re.compile( ) function will compile the pattern into pattern objects. After the compilation the pattern objects will be able to access methods for various operations like searching and subsitutions

Example

```
import re
p=re.compile('hell*o')
```

re.match()

The match function is used to determine if the regular expression (RE) matches at the beginning of the string.

re.group()

The group function is used to return the string matched the RE

Example

```
>>>P=re.compile('hell*o')
>>>m=re.match('hell*o', ' hellooooo world')
>>>m.group()
'hello'
```

re.start()

The start function returns the starting position of the match.

re.end()

The end function returns the end position of the match.

re.span()

The span function returns the tuple containing the (start, end) positions of the match

Example

```
>>> import re
>>> P=re.compile('hell*o')
>>> m=re.match('hell*o', 'hellooooo world')
>>> m.start()
0
>>> m.end()
5
>>> m.span()
(0, 5)
```

re.search()

The search function traverses through the string and determines the position where the RE matches the string

Example

```
>>> m=re.search('hell*o', 'favorite words hellooooo world')
>>> m.start()
15
>>> m.end()
```

```
20
>>> m.group()
'hello'
>>> m.span()
(15, 20)
```

Re.findall()

The function determines all substrings where the RE matches, and returns them as a list.

Example

```
>>> m=re.findall('hell*o', 'hello my favorite words hellooooo world')
>>> m
['hello', 'hello']
```

re.finditer()

The function determines all substrings where the RE matches, and returns them as an iterator.

Example

```
>>> m=re.finditer('hell*o', 'hello my favorite words hellooooo world')
>>> m
<callable-iterator object at 0x0000000002E4ACF8>
>>> for match in m:
print match.span()
(0, 5)
(24, 29)
```

As shown in the above example, m is a iterator. So m is used in the for loop.

Script 1: Write a script to determine if the given substring is present in the string.

```
def search_string():
    import re
    substring='water'
    search1=re.search(substring,'Water water everywhere but not a drop to drink')
    if search1:
        position=search1.start()
        print "matched", substring, "at position", position
    else:
        print "No match found"
```

Script 2: Write a script to determine if the given substring (defined using meta characters) is present in the given string

```
def metasearch():
  import re
  p=re.compile('sing+')
  search1=re.search(p,'Some singers sing well')
  if search1:
    match=search1.group()
    index=search1.start()
    lindex=search1.end()
    print "matched", match, "at index", index ,"ending at" ,lindex
  else:
    print "No match found"
```

EXERCISE

1. Input a string "Green Revolution". Write a script to print the string in reverse.
2. Input the string "Success". Write a script of check if the string is a palindrome or not
3. Input the string "Successor". Write a script to split the string at every occurrence of the letter s.
4. Input the string "Successor". Write a script to partition the string at the occurrence of the letter s. Also Explain the difference between the function split( ) and partition().
5. Write a program to print the pyramid.

   1

   2 2

   3 3 3

   4 4 4 4

   5 5 5 5 5

6. What will be the output of the following statement? Also justify for answer.

   >>> print 'I like Gita\'s pink colour dress'.

7. Give the output of the following statements

   >>> str='Honesty is the best policy'

   >>> str.replace('o','*')

8. Give the output of the following statements

   >>> str='Hello World'

   >>>str.istiltle()

9. Give the output of the following statements.

   >>> str="Group Discussion"

   >>> print str.lstrip("Gro")

10. Write a program to print alternate characters in a string. Input a string of your own choice.

11. Input a string 'Python'. Write a program to print all the letters except the letter'y'.

12. Consider the string str="Global Warming"

    Write statements in python to implement the following

    a) To display the last four characters.

    b) To display the substring starting from index 4 and ending at index 8.

    c) To check whether string has alphanumeric characters or not.

    d) To trim the last four characters from the string.

    e) To trim the first four characters from the string.

    f) To display the starting index for the substring 'Wa'.

    g) To change the case of the given string.

    h) To check if the string is in title case.

    i) To replace all the occurrences of letter 'a' in the string with '*'

13. Study the given script

```
def metasearch():
    import re
    p=re.compile('sing+')
    search1=re.search(p,'Some singers sing well')
    if search1:
        match=search1.group()
        index=search1.start()
        lindex=search1.end()
        print "matched", match, "at index", index ,"ending at", lindex
else:
```

```
        print "No match found"
```

What will be the output of the above script if search() from the re module is replaced by match () of the re module. Justify your answer

14. What will be the output of the script mentioned below? Justify your answer.

```
def find():
        import re
        p=re.compile('sing+')
        search1=p.findall('Some singer sing well')
        print search1
```

15. Rectify the error (if any) in the given statements.

```
>>> str="Hello World"
>>> str[5]='p'
```