CHAPTER 6 STRING

6.1 STRINGS a string is a sequence of characters enclosed within quotes

a string is a sequence of characters. A string may be specified by placing the member characters of the sequence within quotes (single, double or triple). Triple quotes are typically used for strings that span multiple lines. The following assignment statement assigns the string 'Hello Gita' to the variable message. We may also say that the string 'Hello Gita' has been bound to the name message.

```
>>> message = 'Hello Gita'
```

Python function len finds the length of a string. Thus,

```
>>> len(message)
10
```

Individual characters within a string are accessed using a technique known as indexing. In Fig. 6.1, we illustrate the notion of indexing with the help of the string 'Hello Gita'.

0	1	2	3	4	5	6	7	8	9
H	E	l	l	o		G	i	t	a
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Note that the indices start with 0 (index for accessing the first character from the left) and end with one less than the length of the string (index for accessing the last character). The following instructions illustrate how to access individual characters in a string via indices. An index is specified within the sqaure brackets, for example:

```
>>> message[0]
'H'
>>> message[6]
'G'
>>> index = len(message)-1
>>> message[index]
'a'
```

The expression message[0] yields character 'H' since 'H' is stored at index 0.

the value of len(message) being 10, when index is set equal to len(message)-1, message[index] yields the last character of the string message at index 9, i.e. 'a'.

Sometimes it is more convenient to access the elements of a string from the right end. For this purpose, Python provides negative indices.

The negative indices range from - (length of the string) to -1. For the string message, the negative indices would range from -10 to -1. Thus, the entire range of valid indices would be {-10, -9, …, -1, 0, 1, …, 9}. Consequently, message[-1] would yield 'a' and message[-index] would yield 'e':

```
>>> message[-1]
'a'
```

```
>>> message[-index]
'e'
```

If we try to access an index which is not in the valid index range of a string, IndexError will be generated:

```
>>> message[15]
Traceback (most recent call last):
File "<pyshell#1>", line 1, in <module> message[15]
IndexError: string index out of range
```

Strings in Python are immutable, i.e., a component of a string cannot be altered, and an attempt to do so would lead to an error:

```
>>> message[6] = 'S'
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module> message[6] = 'S'
TypeError: 'str' object does not support item
assignment
```

As mentioned earlier, **strings can be concatenated using concatenation operator +** and can be **repeated a multiple number of times using the operator *.**

```
>>> 'Computer' + ' Science'
 'Computer Science'
>>> 'Hi' * 3
'HiHiHi'
```

Strings may also be compared using relational operators . Also, recall that the **functions max and min may be used to find maximum and minimum respectively of several values**,

max(): to find the largest value
min(): to find the smallest value
for example:

```
>>> max('AZ', 'C', 'BD', 'BT')
 'C'
>>> min('BD', 'AZ', 'C')
 'AZ'
>>> max('hello', 'How', 'Are', 'You', 'sir')
'sir'
```

6.1.1 Slicing slicing for retrieving a substring

Sometimes, we need to retrieve a substring, also called a slice, from a string. This can be done by specifying an index range. For example, to extract the substring comprising the character sequence having indices from start to end-1, we specify the range in the form start:end as illustrated below:

```
>>> message = 'Hello Sita'
>>> message[0:5]
'Hello'
>>> message[-10:-5]
 'Hello'
```
If the value of start or end is not specified, Python assumes 0 as the default value of start, and length of the string as the default value of end, for example:
for a slice of string s, **default start index is: 0 or -(len(s)) default end index is: len(s)-1 or -1**
```
>>> message[:5]
 'Hello'
>>> message[5:]
 ' Sita'
>>> message[:]
'Hello Sita'
>>> message[:5] + message[5:]
'Hello Sita'
>>> message[:15]
 'Hello Sita'
>>> message[15:]
''
>>> message[:15] + message[15:]
 'Hello Sita'
```
Note that message[:n] + message[n:] always yields message, irrespective of the value of n being negative, positive, or even out of range. Indeed, values used for defining a slice may be arbitrary integers or even None, for example:
```
>>> message[8:20]
 'ta'
>>> message[6:None]
'Sita'
```
Apart from extracting a consecutive subsequence of characters from a string, Python also allows us to extract a subsequence of the form
start:end:inc.
This subsequence will include every inc th element of the sequence in the range start to end-1, for example:
```
>>> message[0:len(message):2]
'HloSt'
>>> message[0:len(message):3]
'HlSa'
```

We have already mentioned that a string is a sequence of characters. The slicing operation discussed above can also be applied to other sequences such as lists and tuples, to be discussed later.

6.1.2 Membership determining whether a string is a substring of another string

Python also allows us to check for membership of the individual characters or substrings in strings using in operator. Thus, the expression s in str1 yields True or False depending on whether s is a substring of str1, for example:

```
>>> 'h' in 'hello'
True
>>> 'ell' in 'hello'
True
>>> 'h' in 'Hello'
False
```

In Python, **we use a for loop to iterate over each element in a sequence.** In the following example, we construct the string 'h e l l o ' from the string 'hello' by iterating over each element of the string: iterating over a string

```
>>> helloSpaced = ''

>>> for ch in 'hello':

        helloSpaced = helloSpaced + ch + ' '
        >>> helloSpaced
'h e l l o '
```

6.1.3 Built-in Functions on Strings

Next, we examine some built-in functions that can be applied on strings.

Function count count(): to count occurrences of a string in another string

Suppose, we wish to find the number of occurrences of character 'c' in the string 'Encyclopedia'. To achieve this, we apply the function count with argument 'c' to the string 'Encyclopedia':

```
>>> 'Encyclopedia'.count('c')
 2
```

As an application of the function count, examine the following code intended to find the count of all the vowels in the string
'Encyclopedia:':

```
>>> vowels = 'aeiou'
>>> vowelCount = 0
>>> for ch in vowels:
        vowelCount += 'Encyclopedia'.count(ch)
>>> vowelCount
 4
```

The system responds with 4 as the value of the vowelCount, even though the number of vowels in the search string 'Encyclopedia' is 5. As the character 'E' was not included in the string vowels, it was not included in counting too. To include the count of uppercase vowels also, we just need to include vowels in uppercase also in the string vowels:

Vowel=’AEIOUaeiou’

Functions find and rfind finding index of first occurrence of a string in another string

Examine the string colors. Suppose we wish to find out whether 'red' is present as a substring in the string colors. We can do so by using the **function find that returns the index of the first occurrence of string argument 'red' in colors:**

```
>>> colors = 'green, red, blue, red, red, green'
>>> colors.find('red')
 7
```

To find the last occurrence of a string, we use the function rfind that scans the string in reversed order from the right end of the string to the beginning:

finding index of the last occurrence of a string in another string

```
>>> colors.rfind('red')
23
```

If the function find fails to find the desired substring, it returns -1:

```
>>> colors.find('orange')
-1
```

Functions capitalize, title, lower, upper, and swapcase

Python provides several functions that enable us to manipulate the case of strings. For example, the **function capitalize can be used for converting the first letter of a string to uppercase character and converting the remaining letters in the string to lowercase** (if not already so).

transforming a string to sentence case

```
>>> 'python IS a Language'.capitalize()
 'Python is a language'
```

Python function title can be used to capitalize the first letter of each word in a string and change the remaining letters to lowercase (if not already so):

```
>>> 'python IS a PROGRAMMING Language'.title()
 'Python Is A Programming Language'
```

Python functions lower and upper are used to convert all letters in a string to lowercase and uppercase, respectively. Suppose we want to check for the equivalence of a pair of email ids. Since email ids are not case sensitive, we convert both email ids to either uppercase or lowercase before testing for equality:

```
>>> emailId1 = 'geek@gmail.com'
>>> emailId2 = 'Geek@gmail.com'
>>> emailId1 == emailId2
False
>>> emailId1.lower() == emailId2.lower()
True
>>> emailId1.upper() == emailId2.upper()
True
```

Python function swapcase may be used to convert lowercase letters in a string to uppercase letters and vice versa, for example:

```
>>> 'pYTHON IS PROGRAMMING LANGUAGE'.swapcase()
'Python is programming language'
```

Functions islower, isupper, and istitle

The functions islower and isupper can be used to check if all letters in a string are in lowercase or uppercase, respectively, for example:

checking case (lower/ upper) of a string

```
>>> 'python'.islower()
True
>>> 'Python'.isupper()
False
```

The function istitle returns True if a string S (comprising atleast one alphabet) is in title case, for example:

checking whether the string is in title case

```
>>> 'Basic Python Programming'.istitle()
True
>>> 'Basic PYTHON Programming'.istitle()
False
>>> '123'.istitle()
False
>>>’Book 123’.istitle()
True
```

Function replace replacing a substring with another string

The function replace allows to replace part of a string by another string. It takes two arguments as inputs. The first argument is used to specify the substring that is to be replaced. The second argument is used to specify the string that replaces the first string. For example:

```
>>> message = 'Amey my friend, Amey my guide'
>>> message.replace('Amey', 'Vihan')
 'Vihan my friend, Vihan my guide'
```

Functions strip, lstrip, and rstrip **removing whitespace from the beginning/ end of a string**

The functions lstrip and rstrip remove whitespaces from the beginning and end, respectively. The function strip removes whitespaces from the beginning as well as the end of a string. We may choose to remove any other character(s) from the beginning or end by explicitly passing the character(s) as an argument to the function. The following examples illustrate the use of the functions lstrip, rstrip, and strip:

```
>>> '        Hello How are you!         '.lstrip()
 'Hello How are you!         '
>>> '          Hello How are you!       '.rstrip()
'            Hello How are you!'

>>> '                Hello How are you!          '.strip()
'Hello How are you!'
```

Functions split and partition splitting a string into substrings

The function split enables us to **split a string into a list of strings based on a delimiter**. For example:

```
>>> colors = 'Red,  Green,  Blue,  Orange,  Yellow,  Cyan'
>>> colors.split(',')
['Red', ' Green', ' Blue', ' Orange', ' Yellow', ' Cyan']
```

Note that the function split outputs a sequence of strings enclosed in square brackets. **A sequence of objects enclosed in square brackets defines a list.**

If no delimiter is specified, Python uses **whitespace as the default delimiter**:

```
>>> colors.split()
['Red,', 'Green,', 'Blue,', 'Orange,', 'Yellow,', 'Cyan']
```

The function partition divides a string S into two parts based on a delimiter and **returns a tuple** (discussed in the next chapter) comprising string before the delimiter, the delimiter itself, and the string after the delimiter, for example:

```
 >>> 'Hello. How are you?'.partition('.')
('Hello', '.', ' How are you?')
```

Function join joining a sequence of strings

Python function join **returns a string comprising elements of a sequence separated** by the specified delimiter. For example,

```
>>> ' > '.join(['I', 'am', 'ok'])
'I > am > ok'
>>> ' '.join(('I', 'am', 'ok'))
'I am ok'
>>> ' > '.join("'I', 'am', 'ok'")
"' > I > ' > , > > ' > a > m > ' > , > > ' > o > k > '"
```

In the first example, the sequence comprises three elements, namely, 'I', 'am', and 'ok', which are combined to form the string 'I > am > ok'.
In the second example, we use space as a delimiter instead of >.
In the third example, each character in the string "'I > am > ok'" is an element of the sequence of characters in "'I > am > ok'".

Functions isspace, isalpha, isdigit, and isalnum

The functions isspace, isalpha, isdigit, and isalnum enable us to check whether a value is of the desire type. For example, we can check whether the name entered by a user contains only alphabets as follows:

```
>>> name = input('Enter your name : ')
Enter your name : Nikhil
>>> name.isalpha()
True
>>> name = input('Enter your name : ')
Enter your name : Nikhil Kumar
>>> name.isalpha()
False
```

Note that the blank character is not an alphabet. Similarly, to check the validity of a mobile number, we may want it to be a string of length 10 comprising of digits only. This can be achieved using functions **isdigit and len:**

```
>>> mobileN = input('Enter mobile no : ')
Enter mobile no : 1234567890
>>> mobileN.isdigit() and len(mobileN)== 10
True
```

Python function isspace is used to check if the string comprises of all whitespaces:

check for a white space only string

```
>>> '     \n\t '.isspace()
True
```

The function isalnum checks whether a string comprises of alphabets and digits only.

For example, if the password is allowed to comprise of only alphabets and digits, we may use the function isalnum to ensure this constraint for a user specified password:

```
>>> password = input('Enter password : ')
Enter password : Kailash107Ganga
>>> password.isalnum()
True
>>> password = input('Enter password : ')
Enter password : Kailash 107 Ganga
>>> password.isalnum()
False
```

Function startswith and endswith checking whether a string starts or ends with a particular string
Suppose we want to check if the last name of a person is Talwar. We can do this using Python function endswith as follows:

```
>>> name = 'Ankita Narain Talwar'
>>> name.endswith('Talwar')
True
```

Similarly, to find whether a person's name begins with Dr. we can use the function startswith as follows:

```
>>> name = 'Dr. Vihan Garg'
>>> name.startswith('Dr. ')
True
>>> name = ' Dr.Amey Gupta'
>>> name.startswith('Dr. ')
 False
```

Functions encode and decode
Sometimes, we need to transform data from one format to another for the sake of compatibility. Thus, we use, Python function encode that returns the encoded version of a string, based on the given encoding scheme. Another function decode (reverse of function encode) returns the decoded string, for example:

```
>>> str1 = 'message'.encode('utf32')
>>> str1
b'\xff\xfe\x00\x00m\x00\x00\x00e\x00\x00\x00s\x00\x00\x00s\x00\x00\x00a\x00\x00\x00g\x00\x00\x
00e\x00\x00\x00'
>>> str1.decode('utf32')
'message'
```

One may also use alternative coding schemes such as utf8 and utf16.

List of Functions
The functions described above are listed in Table 6.1. Note that the

Table 6.1 String functions

Table 6.1 String functions

Functions	Explanation
`S.count(str1)`	Counts number of times string `str1` occurs in the string `S`.
`S.find(str1)`	Returns index of the first occurrence of the string `str1` in string `S`, and returns -1 if `str1` is not present in string `S`.
`S.rfind(str1)`	Returns index of the last occurrence of string `str1` in string `S`, and returns -1 if `str` is not present in string `S`.
`S.capitalize()`	Returns a string that has first letter of the string `S` in uppercase and rest of the letters in lowercase.
`S.title()`	Returns a string that has first letter of every word in the string `S` in uppercase and rest of the letters in lowercase.
`S.lower()`	Returns a string that has all uppercase letters in string `S` converted into corresponding lowercase letters.
`S.upper()`	Returns a string that has all lowercase letters in string `S` converted into corresponding uppercase letters.
`S.swapcase()`	Returns a string that has all lowercase letters in string `S` converted into uppercase letters and vice versa.
`S.islower()`	Returns `True` if all alphabets in string `S` (comprising atleast one alphabet) are in lowercase, else returns `False`.
`S.isupper()`	Returns `True` if all alphabets in string `S` (comprising atleast one alphabet) are in uppercase, else returns `False`.

Functions	Explanation
`S.istitle()`	Returns `True` if string `S` is in title case, i.e. only first letter of each word is capitalized and the string `S` contains at least one alphabet, and returns `False` otherwise..
`S.replace(str1,str2)`	Returns a string that has every occurrence of string `str1` in `S` replaced by string `str2`.
`S.strip()`	Returns a string that has whitespaces in `S` removed from the beginning and the end.
`S.lstrip()`	Returns a string that has whitespaces in `S` removed from the beginning.
`S.rstrip()`	Returns a string that has whitespaces in `S` removed from the end.
`S.split(delimiter)`	Returns a list formed by splitting the string `S` into substrings. The `delimiter` is used to mark the split points.
`S.partition(delimiter)`	Partitions the string `S` into three parts based on `delimiter` and returns a tuple comprising the string before `delimiter`, `delimiter` itself and the string after `delimiter`.
`S.join(sequence)`	Returns a string comprising elements of the `sequence` separated by delimiter `S`.
`S.isspace()`	Returns `True` if all characters in string `S` comprise whitespace characters only, i.e. `' '`, `'\n'`, and `'\t'` else `False`.
`S.isalpha()`	Returns `True` if all characters in string `S` comprise alphabets only, and `False` otherwise.
`S.isdigit()`	Returns `True` if all characters in string `S` comprise digits only, and `False` otherwise.
`S.isalnum()`	Returns `True` if all characters in string `S` comprise alphabets and digits only, and `False` otherwise.
`S.startswith(str1)`	Returns `True` if string `S` starts with string `str1`, and `False` otherwise.
`S.endswith(str1)`	Returns `True` if string `S` ends with string `str1`, and `False` otherwise.
`S.encode(encoding)`	Returns `S` in an encoded form, based on the given `encoding` scheme.
`S.decode(encoding)`	Returns the decoded string `S`, based on the given `encoding` scheme.

6.2 STRING PROCESSING EXAMPLES

In this section, we will study several examples that illustrate the use of string processing functions described in the previous section.