

10

String Manipulation

In This Chapter

- 10.1 Introduction
- 10.2 Traversing a String
- 10.3 String Operators
- 10.4 String Slices
- 10.5 String Functions and Methods

10.1 INTRODUCTION

You all have basic knowledge about Python strings. You know that Python strings are characters enclosed in quotes of any type – *single quotation marks*, *double quotation marks* and *triple quotation marks*. You have also learnt things like – an empty string is a string that has 0 characters (*i.e.*, it is just a pair of quotation marks) and that Python strings are immutable. You have used strings in earlier chapters to store text type of data.

You know by now that strings are sequence of characters, where each character has a unique position-id/index. The indexes of a string begin from 0 to $(\text{length} - 1)$ in forward direction and $-1, -2, -3, \dots, -\text{length}$ in backward direction.

In this chapter, you are going to learn about many more string manipulation techniques offered by Python like operators, methods etc.

10.2 TRAVERSING A STRING

You know that individual characters of a string are accessible through the unique index of each character. Using the indexes, you can traverse a string character by character. Traversing refers to iterating through the elements of a string, one character at a time. You have already traversed through strings, though unknowingly, when we talked about sequences along with *for loops*.

To traverse through a string, you can write a loop like :

To see
String Traversal
in action



Scan
QR Code

```
name = "superb"
for ch in name :
    print(ch, end = ' ')
```

This loop will traverse
through string **name**
character by character.

The above code will print :

s-u-p-e-r-b-

The information that you have learnt till now is sufficient to create wonderful programs to manipulate strings. Consider the following programs that use the Python string indexing to display strings in multiple ways.

P rogram 10.1

*Program to read a string and display it in reverse order – display one character per line.
Do not create a reverse string, just display in reverse order.*

```
string1 = input("Enter a string : ")
print("The", string1, "in reverse order is:")
length = len(string1)
for a in range(-1, (-length - 1), -1) :
    print(string1[a])
```

Since the **range()** excludes the number
mentioned as upper limit, we have
taken care of that by increasing the
limit accordingly.

Sample run of above program is :

Enter a string : python
The python in reverse order is:

n
o
h
t
y
p

P rogram 10.2

Program to read a string and display it in the form :

first character	last character
second character	second last character
⋮	⋮

For example, string "try" should print as :

t y
r r
y t

```
string1 = input("Enter a string : ")
length = len(string1)
i = 0
for a in range(-1, (-length - 1), -1) :
    print(string1[i], "\t", string1[a])
    i += 1
```

NOTE

When you give a negative index,
Python adds **length** of string to
get its forward index e.g., for a
5-lettered string S, S[-1] will
give S[-1 + 5] i.e., S[4] letter;
for S[-5], it will give you
S[-5 + 5] i.e., S[0]

Sample run of above program is :

Enter a string : python
p n
y o
t h
h t
o y
n p

10.3 STRING OPERATORS

In this section, you'll be learning to work with various operators that can be used to manipulate strings in multiple ways. We'll be talking about basic operators + and *, membership operators in and not in and comparison operators (all relational operators) for strings.

10.3.1 Basic Operators

The two basic operators of strings are : + and *. You have used these operators as *arithmetic operators* before for *addition* and *multiplication* respectively. But when used with strings, + operator performs **concatenation** rather than addition and * operator performs **replication**.

Also, before we proceed, recall that strings are immutable i.e., un-modifiable. Thus every time you perform something on a string that changes it, Python will internally create a new string rather than modifying the old string in place.

String Concatenation Operator +

The + operator creates a new string by joining the two operand strings, e.g.,

"tea" + "pot"

will result into

'teapot'

Two input strings joined (concatenated) to form a new string

Consider some more examples :

Expression

will result into

'1' + '1'

'11'

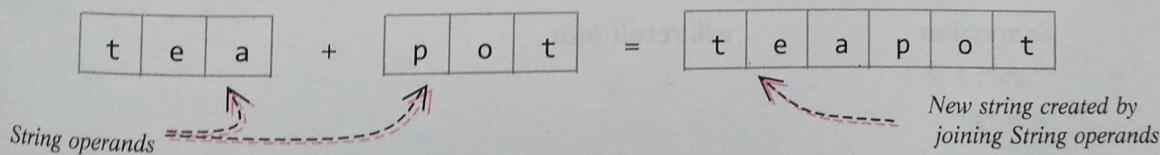
"a" + "0"

'a0'

'123' + 'abc'

'123abc'

Let us see how concatenation takes place internally. Python creates a new string in the memory by storing the individual characters of first string operand followed by the individual characters of second string operand. (see below)



Original strings are not modified as strings are immutable ; new strings can be created but existing strings cannot be modified.

Caution !

Another important thing that you need to know about + operator is that this operator can work with *numbers* and *strings* separately for *addition* and *concatenation* respectively, but in the same expression, **you cannot combine numbers and strings as operands with a + operator**.

For example,

2 + 3 = 5

'2' + '3' = '23'

addition - VALID

concatenation - VALID

But the expression

'2' + 3

is invalid. It will produce an error like :

>>> '2' + 3

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

'2' + 3

TypeError: cannot concatenate 'str' and 'int' objects

Thus we can summarize + operator as follows :

Table 10.1 Working of Python + operator

Operands' data type	Operation performed by +	Example
numbers	addition	9 + 9 = 18
string	concatenation	"9" + "9" = "99"

String Replication Operator *

The * operator when used with numbers (i.e., when both operands are numbers), it performs *multiplication* and returns the *product* of the two number operands.

To use a * operator with strings, you need two types of operands – *a string* and *a number*, i.e., as *number * string* or *string * number*.

Where *string operand* tells the string to be replicated and *number operand* tells the number of times, it is to be repeated; Python will create a new string that is a number of repetitions of the string operand. For example,

3 * "go!"

will return

'go!go!go!'



Input strings repeated specified number of times to form a new string.

Consider some more examples :

Expression

"abc" * 2

5 * "@"

":-:" * 4

"1" * 2

will result into

"abcabc"

"@#####"

":-:-:-:-"

"11"

NOTE

For replication operator *, Python creates a new string that is a number of repetitions of the input string.

Caution!

Another important thing that you need to know about * operator is that this operator can work with numbers as both operands for *multiplication* and with a string and a number for *replication* respectively, but in the same expression, you cannot have *strings* as both the operands with a * operator.

For example,

2 * 3 = 6

"2" * 3 = "222"

multiplication - VALID
replication - VALID

But the expression
"2" * "3"

is invalid. It will produce an error like :

```
>>> "2" * "3"
```

Traceback (most recent call last):

```
  File "<pyshell#0>", line 1, in <module>
    "2" * "3"
```

TypeError: can't multiply sequence by non-int of type 'str'

NOTE

The * operator has to either have both *operands of the number types* (for multiplication) or one *string type and one number type* (for replication). It cannot work with both operands of string types.

Thus we can summarize + operator as follows :

Table 10.2 Working of Python * operator

Operands' data type	Operation performed by *	Example
numbers	multiplication	9 * 9 = 18
string, number	replication	"#" * 3 = "###"
number , string	replication	3 * "#" = "###"

10.3.2 Membership Operators

There are two membership operators for strings (in fact for all sequence types). These are **in** and **not in**. We have talked about them in previous chapter, briefly. Let us learn about these operators in context of strings. Recall that :

- in** Returns *True* if a character or a substring exists in the given string ; *False* otherwise
- not in** Returns *True* if a character or a substring does not exist in the given string; *False* otherwise

Both membership operators (when used with strings), require that both operands used with them are of string type, i.e.,

```
<string> in <string>
<string> not in <string>
```

e.g.,

```
"12" in "xyz"
"12" not in "xyz"
```

Now, let's have a look at some examples :

"a" in "heya"	will give	True
"jap" in "heya"	will give	False
"jap" in "japan"	will give	True
"Jap" in "Japan"	will give	False because j letter's cases are different; hence "jap" is not contained in "Japan"
"jap" not in "Japan"	will give	True because string "jap" is not contained in string "Japan"
"123" not in "hello"	will give	True because string "123" is not contained in string "hello"
"123" not in "12345"	will give	False because "123" is contained in string "12345"

The **in** and **not in** operators can also work with string variables. Consider this :

```
>>> sub = "help"
>>> string = 'helping hand'
>>> sub2 = 'HELP'
>>> sub in string
True
>>> sub2 in string
False
>>> sub not in string
False
>>> sub2 not in string
True
```

P 10.3 Write a program to input an integer and check if it contains any 0 in it.

Program

```
n = int(input("Enter a number :"))
s = str(n)      # convert n to string
if '0' in s:
    print("There's a 0 in", n)
else:
    print("No 0 in", n)
```

Enter a number : 6708

There's a 0 in 6708

=====

Enter a number : 7281

No 0 in 7281

10.3.3 Comparison Operators

Python's standard comparison operators *i.e.*, all relational operators (**<**, **<=**, **>**, **>=**, **==**, **!=**) apply to strings also. The comparisons using these operators are based on the standard character-by-character comparison rules for Unicode (*i.e.*, dictionary order).

Thus, you can make out that

"a" == "a"	will give	True
"abc" == "abc"	will give	True
"a" != "abc"	will give	True
"A" != "a"	will give	True
"ABC" == "abc"	will give	False (letters' case is different)
"abc" != "Abc"	will give	True (letters' case is different)

Equality and non-equality in strings are easier to determine because it goes for exact character matching for individual letters including the case (upper-case or lower-case) of the letter. But for other comparisons like *less than* (**<**) or *greater than* (**>**), you should know the following piece of useful information.

As internally Python compares using Unicode values (called ordinal value), let us know about some most common characters and their ordinal values. For most common characters, the ASCII values and Unicode values are the same.

The most common characters and their ordinal values are shown in Table 10.3.

Table 10.3 Common Characters and their Ordinal Values

Characters	Ordinal Values
'0' to '9'	48 to 57
'A' to 'Z'	65 to 90
'a' to 'z'	97 to 122

'a' < 'A' will give False

because the Unicode value of lower-case letters is higher than upper case letters ; hence 'a' is greater than 'A', not lesser.

'ABC' > 'AB' will give True for obvious reasons.

'abc' <= 'ABCD' will give False

because letters of 'abc' have higher ASCII values compared to 'ABCD'.

'abcd' > 'abcD' will give True

because strings 'abcd' and 'abcD' are same till first three letters but the last letter of 'abcD' has lower ASCII value than last letter of string 'abcd'.

Thus, you can say that Python compares two strings through relational operators using character-by-character comparison of their Unicode values.

Determining Ordinal/Unicode Value of a Single Character

Python offers a built-in function **ord()** that takes a single character and returns the corresponding ordinal Unicode value. It is used as per following format :

ord(<single-character>)

Let us see how, with the help of some examples :

- To know the ordinal value of letter 'A', you'll write `ord('A')` and Python will return the corresponding ordinal value (see below) :

```
>>> ord('A')
```

65

But you need to keep in mind that `ord()` function requires single character string only. You may even write an escape sequence enclosed in quotes for `ord()` function.

The opposite of `ord()` function is `chr()`, i.e., while `ord()` returns the ordinal value of a character, the `chr()` takes the ordinal value in integer form and returns the character corresponding to that ordinal value. The general syntax of `chr()` function is :

```
chr(<int>) # the ordinal value is given in integer
```

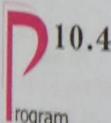
Have a look at some examples (compare with the Table 10.3 given above) :

```
>>> chr(65)
```

'A'

```
>>> chr(97)
```

'a'



10.4 An easy way to print a blank line is `print()`. However, to print ten blank lines, would you write `print()` 10 times ? Or is there a smart alternative ? If yes, write code to print a line with 30 '=' characters followed by 10 blank line and another line with 30 '=' characters.

```
print('=' * 30)
print('\n'* 9)
print('=' * 30)
```



10.5 Write a program that asks a user for a `username` and a `code`. Ensure that the user doesn't use their `username` as part of their `code`.

```
uname = input("Enter user name :")
code = input("Enter code :")
```

```
if uname in code :
    print("Your code should not \
contain your user name.")
    print("Thank you")
```

Enter user name : kobe

Enter code : mykobeworld

Your code should not contain your user name.

Thank you



10.6 Write a program that asks a user for a `username` and a `pcode`. Make sure that `pcode` does not contain `username` and matches '`Trident111`'.

```
uname = input ("Enter user name :")
pcode = input ("Enter code :")
if uname not in pcode and pcode == 'Trident111' :
    print("Your code is Valid to proceed.")
else :
    print("Your code is Not Valid.")
```

Enter user name : Raina

Enter code : Xerxes111

Your code is Not Valid.

Thank you

Enter user name : Raina

Enter code : Raina123

Your code is Not Valid.

Enter user name : Raina

Enter code : Trident111

Your code is Valid to proceed.



10.7 Program that prints the following pattern without using any nested loop.

```
# 
## 
### 
#### 
#####
```

```
string = '#'
pattern = ""      # empty string
for a in range(5) :
    pattern += string
    print(pattern)
```

```
# 
## 
### 
#### 
#####
```

10.4 STRING SLICES

As an English term, you know the meaning of word 'slice', which means - 'a part of'. In the same way, in Python, the term 'string slice' refers to a part of the string, where strings are sliced using a range of indices.

That is, for a string say `name`, if we give `name[n:m]` where `n` and `m` are integers and legal indices, Python will return a slice of the string by returning the characters falling between indices `n` and `m` - starting at `n`, `n+1`, `n+2` ... till `m-1`. Let us understand this with the help of examples. Say we have a string namely `word` storing a string 'amazing' i.e.,

	0	1	2	3	4	5	6
word	a	m	a	z	i	n	g
	-7	-6	-5	-4	-3	-2	-1

Then,

<code>word[0 : 7]</code>	will give	'amazing'	(the letters starting from index 0 going up till 7-1 i.e., 6 : from indices 0 to 6, both inclusive)
<code>word[0 : 3]</code>	will give	'am'	(letters from index 0 to 3-1 i.e., 0 to 2)
<code>word[2 : 5]</code>	will give	'azi'	(letters from index 2 to 4 (i.e., 5-1))
<code>word[-7 : -3]</code>	will give	'amaz'	(letters from indices -7, -6, -5, -4 excluding index -3)
<code>word[-5 : -1]</code>	will give	'azin'	(letters from indices -5, -4, -3, -2 excluding -1)

From above examples, one thing must be clear to you :

- In a string slice, the character at last index (the one following colon (:)) is not included in the result.

To see
String Slices
in action



Scan
QR Code

In a string slice, you give the slicing range in the form [`<begin-index> : <last>`]. If, however, you skip either of the `begin-index` or `last`, Python will consider the limits of the string i.e., for missing `begin-index`, it will consider 0 (the first index) and for missing `last` value, it will consider length of the string.

Consider following examples to understand this :

<code>word[:7]</code>	will give	'amazing'	(missing index before colon is taken as 0 (zero))
<code>word[:5]</code>	will give	'amazi'	(-do-)
<code>word[3:]</code>	will give	'zing'	(missing index after colon is taken as 7 (the length of the string))
<code>word[5:]</code>	will give	'ng'	(-do-)

The string slice refers to a part of the string `s[start:end]` that is the elements beginning at `start` and extending up to but not including `end`.

Following figure (Fig. 10.1) shows some string slices :

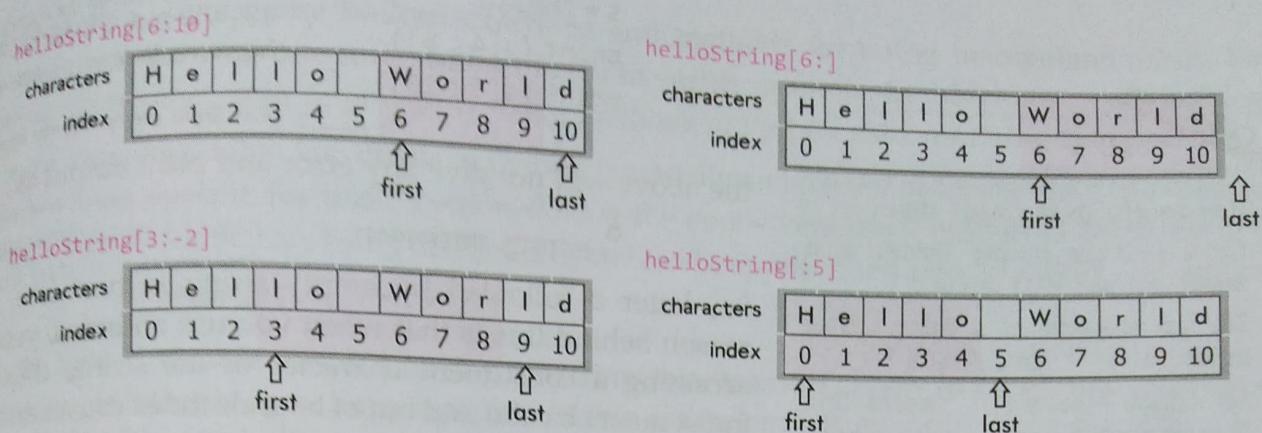


Figure 10.1 String Slicing in Python.

Interesting Inference

Using the same string slicing technique, you will find that

- for any index n , $s[:n] + s[n:]$ will give you original string s .

This works even for n negative or out of bounds.

Let us prove this with an example. Consider the same string namely **word** storing '*amazing*'.

```
>>> word[3:], word [:3]
'zing' 'ama'
>>> word[ :3] + word[3:]
'amazing'
>>> word[ :-7], word [-7: ]
''' 'amazing'
>>> word[ :-7] + word[-7: ]
'amazing'
```

TIP

String $[:: -1]$ is an easy way to reverse a string.

You can give a third (optional) index (say n) in string slice too. With that every n th element will be taken as part of slice e.g., for $word = \text{'amazing'}$, look at following examples.

>>> word [1:6:2]	<i>It will take every 2nd character starting from index = 1 till index < 6.</i>
'mzn'	
>>> word [-7:-3:3]	<i>It will take every 3rd character starting from index = -7 to index < -3.</i>
'az'	
>>> word [:: -2]	<i>Every 2nd character taken backwards.</i>
'giaa'	
>>> word [:: -1]	<i>Every character taken backwards.</i>
'gnizama'	

Another interesting inference is :

- Index out of bounds causes error with strings but slicing a string outside the bounds does not cause error.

```
s = "Hello"
print (s[5])
```

Will cause error because 5 is invalid index-out of bounds, for string "Hello"

Check Point

10.1

1. How are strings internally stored ?
2. For a string s storing 'Goldy', what would $s[0]$ and $s[-1]$ return ?
3. The last character of a string s is at index $\text{len}(s) - 1$. True / False ?
4. For strings, + means (1); * means (2). Suggest words for positions (1) and (2).
5. Given that


```
s1 = "spam"
s2 = "ni!"
```

 What is the output produced by following expressions ?
 - (a) "The Knights who say," + s2
 - (b) 3 * s1 + 2 * s2
 - (c) s1[1]
6. What are membership operators? What do they basically do ?
7. On what principles, the strings are compared in Python ?
8. What will be the result of following expressions ?
 - (a) "Wow Python" [1]
 - (b) "Strings are fun." [5]
 - (c) len("awesome")
 - (d) "Mystery" [:4]
 - (e) "apple" > "pineapple"
 - (f) "pineapple" < "Peach"
 - (g) "cad" in "abracadabra"
 - (h) "apple" in "Pineapple"
 - (i) "pine" in "Pineapple"
9. What do you understand by string slices ?
10. Considering the same strings $s1$ and $s2$ of question 5 above, evaluate the following expressions :
 - (a) $s1[1:3]$
 - (b) $s1[2] + s2[:2]$
 - (c) $s1 + s2[-1]$
 - (d) $s1[:3] + s1[3:]$
 - (e) $s2[:-1] + s2[-1:]$

But if you give

```
s = "Hello"
print(s[4 : 8])
print(s[5 : 10])
```

One limit is outside the bounds
(length of Hello is 5 and thus valid
indexes are 0-4)

Both limits are outside the bounds

the above will not give any error and print output as :

\emptyset empty string

i.e., letter \emptyset followed by empty string in next line. The reason behind this is that when you use an index, you are accessing a constituent character of the string, thus the index must be valid and out of bounds index causes error as there is no character to return from the given index. But slicing always returns a subsequence and empty sequence is a valid sequence. Thus when you slice a string outside the bounds, it still can return empty subsequence and hence Python gives no error and returns empty subsequence.

Truly amazing ;). Isn't it ?



- 10.8** Write a program to input two strings. If string1 is contained in string2 , then create a third string with first four characters of string2 added with word 'Restore'.

```
s1 = input("Enter string 1 :")
s2 = input("Enter string 2 :")
print("Original strings :", s1, s2)
if s1 in s2 :
    s3 = s2[0:4] + "Restore"
    print("Final strings :", s1, s3)
```

```
Enter string 1 : rin
Enter string 2 : shagrin
Original strings : rin shagrin
Final strings : rin shagRestore
```



- 10.9** Write a program to input a string and check if it is a palindrome string using a string slice.

```
s = input("Enter a string :")
if (s == s[::-1]):
    print(s, "is a Palindrome.")
else:
    print(s, "is NOT a Palindrome.")
```

```
Enter a string : Mira
Mira is NOT a Palindrome.
=====
Enter a string : mirarim
mirarim is a Palindrome.
```

10.5 STRING FUNCTIONS AND METHODS

Python also offers many built-in functions and methods for string manipulation. You have already worked with one such method `len()` in earlier chapters. In this chapter, you will learn about many other built-in powerful string methods of Python used for string manipulation.

Every string object that you create in Python is actually an instance of `String` class (you need not do anything specific for this ; Python does it for you – you know built-in). The string manipulation methods that are being discussed below can be applied to string as per following syntax :

`<stringObject>. <method name> ()`

In the following table we are referring to `<stringObject>` as *string* only (no angle brackets) but the meaning is intact i.e., you have to replace *string* with a legal string (i.e., either a string literal or a string variable that holds a string value).

Let us now have a look at some useful built-in string manipulation methods.

Python's Built-in String Manipulation Functions and Methods

Let us now discuss some most commonly used string functions and methods of Python.

1. The `len()` function

It returns the length of its argument string, i.e., it returns the count of characters in the passed string. The `len()` function is a Python standard library function.

It is used as per the syntax :

`len(<string>)`

Examples

`>>> len("hello")`

6

`>>> name = "Maria"`

`>>> len(name)`

5

2. The `capitalize()` method

It returns a copy of the string with its first character capitalized.

It is used as per the syntax :

`<string>.capitalize()`

Example

`>>> 'true'.capitalize()`

True

`>>> 'i love my India'.capitalize()`

I love my India

3. The `count()` method

It returns the number of occurrences of the substring `sub` in `string` (or `string [start:end]` if these arguments are given).

It is used as per the syntax :

`<string>.count (sub[, start[, end]])`

Examples

To count the occurrences of 'ab' in - 'abracadabra' in the whole string, 4-8 characters and 6th character onwards.

`>>> 'abracadabra'.count('ab')`

2

Count the occurrences of 'ab' in the whole string

`>>> 'abracadabra'.count('ab', 4, 8)`

0

Count the occurrences of 'ab' in string's 4th to 8th characters.

`>>> 'abracadabra'.count('ab', 6)`

1

Count the occurrences of 'ab' in string's 6th character onwards

4. The `find()` method

It returns the *lowest* index in the string where the substring `sub` is found within the slice range of `start` and `end`. Returns -1 if `sub` is not found.

It is used as per the syntax :

```
<string>.find(sub[, start[, end]])
```

Example

```
>>> string = 'it goes as - ringa ringa
roses'
>>> sub = 'ringa'
>>> string.find(sub)
13
>>> string.find(sub, 15, 22)
-1
>>> string.find(sub, 15, 25)
19
```

5. The index() method

It returns the *lowest index* where the specified substring is found. If the substring is not found then an exception, `ValueError`, is raised. It works like `find()`, but `find()` returns `-1` if the `sub` is not found, BUT `index()` raises an exception, if `sub` is not found in the string.

It is used as per the syntax :

```
<string>.index(sub[, start[, end]])
```

Examples

To find the index of the 1st occurrence of 'ab' in 'abracadabra' – in the whole string, 4-8 characters – and 6th character onwards.

```
>>> 'abracadabra'.index('ab')
0
    Returned the index of 1st occurrence
    of 'ab' in the whole string

>>> 'abracadabra'.index('ab', 6)
7
    Returned the index of 1st occurrence of
    'ab' in the string's 6th character onwards

>>> 'abracadabra'.index('ab', 4, 8)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'abracadabra'.index('ab', 4, 8)
ValueError: substring not found
    Raised ValueError exception, when 'ab' could
    not be found in given range of the string
```

6. The isalnum() method

It returns `True` if the characters in the `string` are alphanumeric (alphabets or numbers) and there is at least one character, `False` otherwise. Please note that the space (' ') is not treated as alphanumeric.

It is used as per the syntax :

```
<string>.isalnum()
```

Example

```
>>> string = "abc123"
>>> string2 = 'hello'
>>> string3 = '12345'
>>> string4 = ' '
>>> string.isalnum()
True
>>> string2.isalnum()
True
>>> string3.isalnum()
True
>>> string4.isalnum()
False
```

NOTE

To check if `c` is a space or a special character, you can use `c.isalnum() != True`

7. The isalpha() method

It returns `True` if all characters in the `string` are alphabetic and there is at least one character, `False` otherwise.

It is used as per the syntax :

```
<string>.isalpha()
```

Example

(considering the same string values as used in example of previous function - `isalnum`)

```
>>> string.isalpha()
False
>>> string2.isalpha()
True
>>> string3.isalpha()
False
>>> string4.isalpha()
False
```

NOTE

Use the `index()` and `find()` functions only when you want to know the index position of the substring. For checking the presence of a substring, use the `in` operator instead.

Example (considering the same string values as used in the example of previous function - `isupper()`)

```
>>> string.upper()
'HELLO'
>>> string2.upper()
'THERE'
>>> string3.upper()
'GOLDY'
>>> string4.upper()
'U123'
>>> string5.upper()
'123F'
```

14. The `lstrip()`, `rstrip()`, `strip()` methods

lstrip() Returns a copy of the string with leading whitespaces removed, i.e., whitespaces from the leftmost end are removed.

It is used as per the syntax :

`<string>.lstrip()`

rstrip() Returns a copy of the string with trailing whitespaces removed, i.e., whitespaces from the rightmost end are removed.

It is used as per the syntax :

`<string>.rstrip()`

strip() Returns a copy of the string with leading and trailing whitespaces removed, i.e., whitespaces from the leftmost and the rightmost ends are removed.

It is used as per the syntax :

`<string>.strip()`

Example

```
>>> " Sipo ".lstrip()
'Sipo'           ↑
The lstrip() removed only the
leading whitespaces

>>> " Sipo ".rstrip()
' Sipo'          ↑
The rstrip() removed only the trailing
whitespaces

>>> " Sipo ".strip()
'Sipo'           ↑
The strip() removed both the leading and the
trailing whitespaces
```

Path Wala

15. The `startswith()`, `endswith()` method
startswith() Returns *True* if the string starts with the substring *sub*, otherwise returns *False*.

It is used as per the syntax :

`<string>.startswith()`

endswith() Returns *True* if the string ends with the substring *sub*, otherwise returns *False*. It is used as per the syntax :

`<string>.endswith()`

Example

```
>>> "abcd".startswith("cd")
False
>>> "abcd".startswith("ab")
True
>>> "abcd".endswith("b")
False
>>> "abcd".endswith("cd")
True
```

16. The `title()` method

It returns a title-cased version of the string where all words start with uppercase characters and all remaining letters are in lowercase.

It is used as per the syntax :

`<string>.title()`

Example

```
>>> "the sipo app".title()
'The Sipo App'
>>> "COMPUTER SCIENCE".title()
'Computer Science'
```

17. The `istitle()` method

It returns *True* if the string has the title case, (i.e., the first letter of each word in uppercase, while rest of the letters in lowercase), *False* otherwise. It is used as per the syntax :

`<string>.istitle()`

Example

```
>>> 'Computer Science'.istitle()
True
>>> "COMPUTER SCIENCE".istitle()
False
```

18. The `replace()` method

It returns a copy of the string with all occurrences of substring *old* replaced by *new string*. It is used as per the syntax :

`<string>.replace(old, new)`

Example

```
>>> 'abracadabra'.replace('ab', 'sp')
'spracadspra'
>>> 'I work for you'.replace('work', 'care')
'I care for you'
>>> 'you and I work for you'.replace('you', 'U')
'U and I work for U'
```

19. The `join()` method

It joins a string or character (*i.e.*, `<str>`) after each member of the *string iterator* *i.e.*, a *string based sequence*. It is used as per the syntax :

`<string>.join(<string iterable>)`

- (i) If the string based iterator is a string then the `<str>` is inserted after every character of the string, *e.g.*,

```
>>> "*".join("Hello") ←
'H*e*l*l*o' ← See, a character is joined with each member of the given string to form the new string
```

```
>>> "***".join("TRIAL") ←
'T***R***I***A***L' ← See, a string("****" here) is joined with each member of the given string to form the new string
```

- (ii) If the string based iterator is a *list* or *tuple* of strings then, the given string/character is joined with each member of the list or tuple, BUT the tuple or list must have all member as strings otherwise Python will raise an error.

```
>>> "$$".join(["trial", "hello"])
'trial$$hello' ← Given string ("$$") joined between the individual items of a string based list
```

```
>>> "###".join(("trial", "hello", "new"))
'trial##hello##new' ← Given string ("##") joined between the individual items of a string based tuple
```

```
>>> "###".join((123, "hello", "new"))
Traceback (most recent call last):
"##".join((123, "hello", "new")) ← The sequence must contain all strings, else Python will raise an error.
```

TypeError: sequence item 0: expected str instance, int found

20. The `split()` method

It splits a string(*i.e.*, `<str>`) based on given string or character (*i.e.*, `<string/char>`) and returns a list containing split strings as members. It is used as per the syntax :

`<string>.split(<string/char>)`

- (i) If you do not provide any argument to split then by default it will split the given string considering whitespace as a separator, e.g.,

```
>>>"I Love Python".split()  
['I', 'Love', 'Python']
```

```
>>>"I Love Python".split(" ")  
['I', 'Love', 'Python']
```

With or without whitespace, the output is just the same i.e., the list containing individual words

- (ii) If you provide a string or a character as an argument to split(), then the given string is divided into parts considering the given string/character as separator and separator character is not included in the split strings e.g.,

```
>>>"I Love Python".split("o")  
['I L', 've Pyth', 'n']
```

The given string is divided from positions containing "o"

NOTE

You can use **split()** with a line of text to split it in words. It will be quite handy for you in some file based programs.

21. The **partition()** method

The **partition()** method splits the string at the first occurrence of separator, and returns a tuple containing three items.

- ⇒ The part before the separator
- ⇒ The separator itself
- ⇒ The part after the separator

It is used as per the syntax :

```
<string>.partition(<separator/string>)
```

Example

```
>>> txt = "I enjoy working in Python"  
>>> x = txt.partition("working") ← 'working' is the separator string  
>>> print(x)  
('I enjoy', 'working', 'in Python') ←
```

See, it returned a tuple with 3 elements : the substring until the separator, the separator, and the substring after the separator

Table 10.1 Difference between **partition()** vs. **split()**

The split()'s Functioning	The partition()'s Functioning
<p>split() will split the string at any occurrence of the given argument.</p> <p>It will return a list type containing the split substrings.</p> <p>The length of the list is equal to the number of words, if split on whitespaces.</p>	<p>partition() will only split the string at the first occurrence of the given argument.</p> <p>It will return a tuple type containing the split substrings.</p> <p>It will always return a tuple of length 3, with the given separator as the middle value of the tuple.</p>

Consider following program that applies some of the string manipulation functions that you have learnt so far.



10.10 Program that reads a line and prints its statistics like :

Number of uppercase letters :	Number of lowercase letters :
Number of alphabets :	Number of digits :
Number of symbols :	

```

line = input( "Enter a line :")
lowercount = uppercount = 0
digitcount = alphacount = symcount = 0
for a in line :
    if a.islower() :
        lowercount += 1
    elif a.isupper() :
        uppercount += 1
    elif a.isdigit() :
        digitcount += 1
    elif a.isalpha() :
        alphacount += 1
    elif a.isalnum() != True and a != ' ' :
        symcount += 1
print ("Number of uppercase letters : ", uppercount)
print ("Number of lowercase letters : ", lowercount)
print ("Number of alphabets : ", alphacount)
print ("Number of digits : ", digitcount)
print("Number of symbols : ", symcount)

```

Sample run of the program is :

```

Enter a line : Hello 123, ZIPPY zippy Zap
Number of uppercase letters : 7
Number of lowercase letters : 11
Number of alphabets : 18
Number of digits : 3
Number of symbols : 1

```



10.11 Program that reads a line and a substring. It should then display the number of occurrences of the given substring in the line.

```

line = input( "Enter line :")
sub = input( "Enter substring :")
length = len(line)
lensub = len(sub)
start = count = 0
end = length
while True :
    pos = line.find(sub, start, end)
    if pos != -1 :
        count += 1
        start = pos + lensub
    else :
        break
    if start >= length :
        break
print ("No. of occurrences of", sub, ':', count)

```

Sample runs of above program is :

```
Enter line : jingle bells jingle bells jingle all the way
Enter substring : jingle
No. of occurrences of jingle : 3
===== RESTART =====
Enter line : jingle bells jingle bells jingle all the way
Enter substring : bells
No. of occurrences of bells : 2
```

P 10.12 Write a program that inputs a string that contains a decimal number and prints out the decimal part of the number. For instance, if 515.8059 is given, the program should print out 8059.

```
Program s = input('Enter a string (a decimal number) : ')
t = s.partition('.')
print("Given string:", s)
print("part after decimal", t[2])
```

Sample run of the program is :

```
Enter a string (a decimal number):515.8059
Given string: 515.8059
part after decimal 8059
```

P 10.13 Write a program that inputs individual words of your school motto and joins them to make a string. It should also input day, month and year of your school's foundation date and print the complete date.

```
Program
print("Enter words of your school motto, one by one")
w1 = input("Enter word 1:")
w2 = input("Enter word 2:")
w3 = input("Enter word 3:")
w4 = input("Enter word 4:")
motto = " ".join( [w1, w2, w3, w4])
print("Enter your school's foundation date.")
dd = input("Enter dd:")
mm = input("Enter mm:")
yyyy = input("Enter yyyy:")
dt = "/".join( [dd, mm, yyyy] )
print("School motto :", motto)
print("School foundation date : ", dt)
```

Check Point

10.2

- What is the role of these functions ?
 - isalpha()
 - isalnum()
 - isdigit()
 - isspace()
- Name the case related string manipulation functions.
- How is islower() function different from lower() function ?
- What is the utility of find() function ?
- How is capitalize() function different from upper() function ?

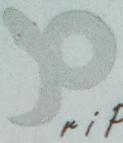
Sample run of the program is :

```
Enter words of your school motto, one by one
Enter word 1:World
Enter word 2:is
Enter word 3:a
Enter word 4:family.
Enter your school's foundation date.
Enter dd:01
Enter mm:10
Enter yyyy:1975
School motto : World is a family.
School foundation date : 01/10/1975
```

Path Wala

String Manipulation Functions and Methods

S.No.	Function / Method	Description
1.	<code>len(<string>)</code>	It returns the length of its argument string.
2.	<code><string>.capitalize()</code>	It returns a copy of the string with its first character capitalized.
3.	<code><string>.count(sub[, st[, end]])</code>	It returns the number of occurrences of the substring <code>sub</code> in string <code>(or string [start:end] if these arguments are given)</code> .
4.	<code><string>.find(sub[, st[, end]])</code> st is start	It returns the <i>lowest</i> index in the string where the substring <code>sub</code> is found within the slice range of start and end. Returns <code>-1</code> if <code>sub</code> is not found.
5.	<code><string>.index(sub[, st[, end]])</code>	It returns the <i>lowest index</i> where the specified substring is found.
6.	<code><string>.isalnum()</code>	It returns True if the characters in the <code>string</code> are alphanumeric (alphabets or numbers) and there is at least one character, False otherwise.
7.	<code><string>.isalpha()</code>	It returns True if all characters in the <code>string</code> are alphabetic and there is at least one character, False otherwise.
8.	<code><string>.isdigit()</code>	It returns True if all the characters in the <code>string</code> are digits. There must be at least one character, otherwise it returns False .
9.	<code><string>.islower()</code>	It returns True if all cased characters in the <code>string</code> are lowercase. There must be at least one cased character. It returns False otherwise.
10.	<code><string>.isspace()</code>	It returns True if there are only whitespace characters in the <code>string</code> . There must be at least one character. It returns False otherwise.
11.	<code><string>.isupper()</code>	It tests whether all cased characters in the <code>string</code> are uppercase and requires that there be at least one cased character. Returns True if so and False otherwise.
12.	<code><string>.lower()</code>	It returns a copy of the <code>string</code> converted to lowercase.
13.	<code><string>.upper()</code>	It returns a copy of the <code>string</code> converted to uppercase.
14.	<code><string>.lstrip()</code> <code><string>.rstrip()</code> <code><string>.strip()</code>	<p>It returns a copy of the string with leading white- spaces removed.</p> <p>It returns a copy of the string with trailing white- spaces removed.</p> <p>It returns a copy of the string with leading and trailing whitespaces removed.</p>
15.	<code><string>.startswith()</code> <code><string>.endswith()</code>	<p>It returns True if the string starts with the substring <code>sub</code>, otherwise returns False.</p> <p>It returns True if the string ends with the substring <code>sub</code>, otherwise returns False.</p>
16.	<code><string>.title()</code>	It returns a title-cased version of the string where all words start with uppercase characters and all remaining letters are lowercase.
17.	<code><string>.istitle()</code>	It returns True if the string has the title case.
18.	<code><string>.replace(old, new)</code>	It returns a copy of the string with all occurrences of substring <code>old</code> replaced by <code>new</code> string.
19.	<code><string>.join(<string iterable>)</code>	It <i>joins</i> a string or character (<i>i.e.</i> , <code><str></code>) after each member of the <code>string iterator</code> <i>i.e.</i> , a <code>string based sequence</code> .
20.	<code><string>.split(<string/char>)</code>	It <i>splits</i> a string(<i>i.e.</i> , <code><str></code>) based on the given string or character (<i>i.e.</i> , <code><string/char></code>) and returns a list containing split strings as members.
21.	<code><string>.partition(<sep/string>)</code> sep is seperator	It splits the string at the first occurrence of separator, and returns a tuple containing <i>three</i> items as string till separation, separator and the string after separator.



This 'Progress in Python' session works on the objective of practicing String manipulation operators and functions.

Please check the practical component-book – Progress in Computer Science with Python and fill it there in PriP 10.1 under Chapter 10 after practically doing it on the computer.

>>> * <<<

LET US REVISE

- ☞ Python strings are stored in memory by storing individual characters in contiguous memory locations.
- ☞ The memory locations of string characters are given indexes or indices.
- ☞ The index (also called subscript sometimes) is the numbered position of a letter in the string.
- ☞ In Python, indices begin 0 onwards in the forward direction up to **length-1** and -1, -2, ... up to -**length** in the backward direction. This is called two-way indexing.
- ☞ For strings, + means concatenation, * means replication.
- ☞ The **in** and **not in** are membership operators that test for a substring's presence in the string.
- ☞ Comparison in Python strings takes place in dictionary order by applying character-by-character comparison rules for ASCII or Unicode.
- ☞ The **ord()** function returns the ASCII value of given character.
- ☞ The string slice refers to a part of the string **s[start:end]** is the element beginning at **start** and extending up to but not including **end**.
- ☞ The string slice with syntax **s[start : end : n]** is the element beginning at **start** and extending up to but not including **end**, taking every **n**th character.
- ☞ Python also provides some built-in string manipulation methods like : **len()**, **capitalize()**, **find()**, **isalnum()**, **isalpha()**, **isdigit()**, **islower()**, **isupper()**, **lower()**, **upper()**, **index()**, **join()**, **split()**, **partition()** etc.

Objective Type Questions

OTQs

Multiple Choice Questions

1. Negative index -1 belongs to _____ of string.

(a) first character	(b) last character
(c) second last character	(d) second character
2. Which of the following is/are not legal string operators ?

(a) in	(b) +	(c) *	(d) /
--------	-------	-------	-------
3. Which of the following functions will return the total number of characters in a string ?

(a) count()	(b) index()	(c) len()	(d) all of these
--------------	--------------	------------	------------------

4. Which of the following functions will return the last three characters of a string s ?
 (a) `s[3:]` (b) `s[:3]` (c) `s[-3:]` (d) `s[:-3]`
5. Which of the following functions will return the first three characters of a string s ?
 (a) `s[3:]` (b) `s[:3]` (c) `s[-3:]` (d) `s[:-3]`
6. Which of the following functions will return the string in all caps ?
 (a) `upper()` (b) `toupper()` (c) `isupper()` (d) `to-upper()`
7. Which of the following functions will return the string with every 'P' replaced with a 'z' ?
 (a) `find()` (b) `index()` (c) `replace()` (d) `split()`
8. Which of the following functions will return a list containing all words of the string ?
 (a) `find()` (b) `index()` (c) `partition()` (d) `split()`
9. Which of the following functions will return always a tuple of 3 elements ?
 (a) `find()` (b) `index()` (c) `partition()` (d) `split()`
10. What is the output of the following code ?
- ```
str1 = "Mission 999";
str2 = "999"
print(str1.isdigit(), str2.isdigit())
```
- (a) False True      (b) False False      (c) True False      (d) True True
11. Choose the correct function to get the ASCII code of a character.  
 (a) `char('char')`      (b) `ord('char')`      (c) `ascii('char')`      (d) All of these
12. Which method should I use to convert String "Python programming is fun" to "Python Programming Is Fun" ?  
 (a) `capitalize()`      (b) `title()`      (c) `istitle()`      (d) `upper()`
13. Guess the correct output of the following String operations.
- ```
str1 = 'Wah'
print(str1*2)
```
- (a) WahWah (b) TypeError: unsupported operand type(s) for * : 'str' and 'int'
 (c) WWaahh (d) Wah2
14. What is the output of the following string operation ?
- ```
str = "My roll no. is 12"
print(str.isalnum())
```
- (a) True      (b) False      (c) Error      (d) No output
15. Select the correct output of the following String operations.
- ```
str1 = 'Waha'
print(str1[:3] + 'Bhyi' + str1[-3:])
```
- (a) Wah Bhyi Wah (b) Wah Bhyi aha (c) WahBhyiWah (d) WahBhyiWaha
16. Select the correct output of the following String operations.
- ```
str = "my name is Anu John"""
print(str.capitalize())
```
- (a) 'My name is anu john'  
 (b) TypeError: unsupported operand type(s) for \* : 'str' and 'int'  
 (c) 'My name is Anu John'      (d) 'My Name Is Anu John'

17. Choose the correct function to get the character from ASCII number.  
 (a) ascii('number')    (b) char(number)    (c) chr(number)    (d) all of these
18.  $s = ' '$  (single space). Then  $s.isalnum()$  will return.  
 (a) True    (b) False    (c) Error    (d) nothing
19. Which of the following functions removes all leading and trailing spaces from a string?  
 (a) lstrip()    (b) rstrip()    (c) strip()    (d) all of these
20. Which of the following functions will raise an error if the given substring is not found in the string?  
 (a) find()    (b) index()    (c) replace()    (d) all of these

### Fill in the Blanks

- The string indexes begin \_\_\_\_\_ onwards.
- For strings, \_\_\_\_\_ operator performs concatenation.
- For strings, \_\_\_\_\_ operators performs replication.
- The \_\_\_\_\_ and \_\_\_\_\_ are membership operators for strings (in, not in).
- The \_\_\_\_\_ returns the ASCII value of a given character.
- If a string contains letters and digits, function \_\_\_\_\_ will return true.
- 'ab'.isalpha() will return value as \_\_\_\_\_.
- To get each word's first letter capitalized, \_\_\_\_\_ function is used.
- Function \_\_\_\_\_ raises an exception if the substring is not found.
- Function \_\_\_\_\_ divides a line of text into individual words.

### True/False Questions

- Strings have both positive and negative indexes.
- Python does not support a character type; a single character is treated as strings of length one.
- Strings are immutable in Python, which means a string cannot be modified.
- Like '+', all other arithmetic operators are also supported by strings.
- Functions capitalize() and title() return the same result.
- Functions partition() and split() work identically.
- The find() and index() are similar functions.
- The find() does not raise an exception if the substring is not found.
- The partition() function's result is always a 3-element tuple.
- The split() returns always a 3-element list.

**NOTE :** Answers for OTQs are given at the end of the book.

### Solved Problems

- What is a string slice? How is it useful?

**Solution.** A sub-part or a slice of a string, say  $s$ , can be obtained using  $s[n:m]$  where  $n$  and  $m$  are integers. Python returns all the characters at indices  $n, n+1, n+2 \dots m-1$  e.g., 'Well done' [1 : 4] will give 'ell'.

2. Figure out the problem with following code fragment. Correct the code and then print the output.

```

1. s1 = 'must'
2. s2 = 'try'
3. n1 = 10
4. n2 = 3
5. print (s1 + s2)
6. print (s2 * n2)
7. print (s1 + n1)
8. print (s2 * s1)

```

**Solution.** The problem is with lines 7 and 8.

- ❖ Line 7 – **print s1 + n1** will cause error because **s1** being a string cannot be concatenated with a number **n1**.

This problem can be solved either by changing the operator or operand e.g., all the following statements will work :

- print (s1 \* n1)**
- print (s1 + str(n1))**
- print (s1 + s2)**

- ❖ Line 8 – **print (s2 \* s1)** will cause error because two strings cannot be used for replication.

The corrected statement will be :

```
print (s2 + s1)
```

If we replace the **Line 7** with its suggested solution (b), the output will be :

```

must try
try try try
must 10
try must

```

3. Consider the following code :

```

string = input("Enter a string :")
count = 3
while True :
 if string[0] == 'a' :
 string = string[2 :]
 elif string[-1] == 'b' :
 string = string [: 2]
 else :
 count += 1
 break
 print (string)
 print (count)

```

What will be the output produced, if the input is :

- aabbcc
- aaccbb
- abcc

**Solution.**

- |          |        |        |
|----------|--------|--------|
| (a) bbcc | (b) cc | (c) cc |
| 4        | 4      | 4      |

4. Consider the following code :

```

Inp = input("Please enter a string :")
while len(Inp) <= 4 :
 if Inp[-1] == 'z' : #condition 1
 Inp = Inp [0 : 3] + 'c'
 elif 'a' in Inp : #condition 2
 Inp = Inp[0] + 'bb'
 elif not int(Inp[0]) : #condition 3
 Inp = '1' + Inp[1 :] + 'z'
 else :
 Inp = Inp + '*'
print (Inp)

```

What will be the output produced if the input is

- 1bzz
- '1a'
- 'abc'
- 'Oxy'
- 'xyz'

**Solution.**

- 1bzc\*
- 1bb\*\*
- endless loop because 'a' will always remain at index 0 and condition 3 will be repeated endlessly.
- 1×yc\*
- Raises an error as *Inp[0]* cannot be converted to int.

5. What will be the output produced by the following ?

- >>> "One\n\tTwo Three Four\n\nFive\n".split()
- >>> "-".join( ["a","b","c","d"] )

**Solution.**

- [ 'One', 'Two', 'Tree', 'Four', 'Five' ]
- 'a—b—c—d'

6. What is the error in the following code ?

```

s = "Anata won"
s[3] = "y"

```

**Solution.** The problem with the above code is that it is trying to change the 4th character of the given string through item assignment, but it is not allowed for strings as strings are immutable in Python. Hence the error.

7. Why is the following code raising an exception `ValueError`? Correct the code so that it does not raise an exception.

```
>>> "whenever".index("what")
Traceback(most recent call last):
 File "<pyshell#25>", line 1, in <module>
 "whenever".index("what")
ValueError: substring not found
```

**Solution.** The search string "what" is not found in the given string "whenever" and when the search is performed through `index()`, it raises an exception if the search string is not found in the main string. To avoid the exception, we can use the `find()`, which will return -1, if the search string is not found and will not raise any exception, i.e., :

```
>>> "whenever".find("what")
-1
```

8. What is the similarity and difference between `partition()` and `split()` functions?

**Solution.** Both the `partition()` and `split()` functions divide the given strings into sub-strings but their working is different from one another. Listed below are the differences between the `partition()` and `split()` functions.

| The <code>split()</code> 's Functioning                                                                                                                                                                                                               | The <code>partition()</code> 's Functioning                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><code>split()</code> splits the string at any occurrence of the given argument</p> <p>It returns a list type containing the split substrings.</p> <p>The length of the returned list is equal to the number of words, if split on whitespaces.</p> | <p><code>partition()</code> splits the string only at the first occurrence of the given argument</p> <p>It returns a tuple type containing the split substrings.</p> <p>It always returns a tuple of length 3, with the given separator as the middle value of the tuple.</p> |

9. Write a program that takes a string with multiple words and then capitalizes the first letter of each word and forms a new string out of it.

**Solution.**

```
string = input("Enter a string :")
length = len(string)
a = 0
end = length
string2 = '' #empty string
while a < length :
 if a == 0 :
 string2 += string[0].upper()
 a += 1
 elif (string[a] == ' ' and string[a+1] != ' ') :
 string2 += string[a]
 string2 += string[a+1].upper()
 a += 2
 else :
 string2 += string[a]
 a += 1
print ("Original String :", string)
print ("Capitalized words String", string2)
```

10. Write a program that reads a string and checks whether it is a palindrome string or not without using string slice.

**Solution.**

```
string = input("Enter a string :")
length = len(string)
mid = length/2
rev = -1
for a in range(mid) :
 if string[a] == string[rev] :
 a += 1
 rev -= 1
 else :
 print (string, "is not a palindrome")
 break
else :
 print (string, "is a palindrome")
```

11. Write a program that reads a string and displays the longest substring of the given string having just the consonants.

**Solution.**

```
string = input("Enter a string :")
length = len(string)
maxlength = 0
maxsub = '' # empty string
sub = '' # empty string
lensub = 0
for a in range(length):
 if string[a] in 'aeiou' or string[a] in 'AEIOU':
 if lensub > maxlength:
 maxsub = sub
 maxlength = lensub
 sub = ''
 lensub = 0
 else:
 sub += string[a]
 lensub = len(sub)
 a += 1
print ("Maximum length consonant substring is :" , maxsub, end = ' ')
print ("with" , maxlength, "characters")
```

12. Write a program that reads a string and then prints a string that capitalizes every other letter in the string e.g., passion becomes pAsSiOn.

**Solution.**

```
string = input("Enter a string :")
length = len(string)
print ("Original string :" , string)
string2 = "" # empty string
for a in range(0, length, 2):
 string2 += string[a]
 if a < (length-1):
 string2 += string[a + 1].upper()
print ("Alternatively capitalized string :" , string2)
```

13. Write a program that reads email-id of a person in the form of a string and ensures that it belongs to domain @edupillar.com. (Assumption : No invalid characters are there in email-id).

**Solution.**

```
email = input("Enter your email id :")
domain = '@edupillar.com'
ledo = len(domain) # ledo - length of domain
lema = len(email) # lema - length of email
sub = email[lema-ledo :]
if sub == domain:
 if ledo != lema:
 print ("It is valid email id")
 else:
 print ("This is invalid email id - contains just the domain name.")
else:
 print ("This email-id is either not valid or belongs to some other domain.")
```

14. Write a program that asks the user for a string *s* and a character *c*; and then it prints out the location of character *c* in the string *s*.

**Solution.**

```
s = input('Enter some text: ')
c = input('Enter a character: ')
print("In", s, "character", c, "found at these locations : ", end = "")
for i in range(len(s)):
 if s[i]==c: print(i, end = ",")
```

Enter some text: "Hello there"

Enter a character: e

In "Hello there" character e found at these locations : 2,9,11,

15. Write a program that asks the user for a string and creates a new string that doubles each character of the original string. For instance, if the user enters *Sipo*, the output should be *SSiioppoo*

**Solution.**    s = input('Enter a string :')

```
ds = ""
for c in s:
 ds = ds + c*2
print("Original string:", s)
print("Doubled string:", ds)
```

Sample run of the program is:

Enter a string : Sipo  
Original string: Sipo  
Doubled string: SSiioppoo

16. Write a program that removes all capitalization and common punctuation from a string *s*.

**Solution.**    s1 = input ("Enter string 1 : ")
s2 = input ("Enter string 2 : ")
print("Original strings :", s1, s2)
if s1 in s2 :
 s3 = s2[:4] + "Restore"
print("Final strings :", s1, s3)

Enter a string : "Hello! Ria, Siya & I work together."
Final string: hello ria siya & i work together

17. A DNA string is string storing multiple codons, where each codon is 3-lettered string, e.g., if the DNA string is MDCHPIWRFVRYCEYPLHVEMACKR, then first 3 codons will be MDC, HPI, and WRF.  
Write a program to input the DNA string and print first 3 codons from it.

**Solution.**    dna\_seq = input('Enter DNA string :')
Dseq = dna\_seq.upper() # convert to upper case
print("Codon 1", Dseq[0:3] ) # print first 3 characters
print("Codon 2", Dseq[3:6] ) # print next 3
print("Codon 3", Dseq[6:9] ) # print next 3

Enter DNA string : MDCHPIWRFVRYCEYPLHVEMACKR
Codon 1 MDC
Codon 2 HPI
Codon 3 WRF

18. Write a program that inputs a line of text and prints its each word in a separate line. Also, print the count of words in the line.

Solution.

```
s = input("Enter a line of text :")
count = 0
for word in s.split():
 print(word)
 count += 1
print("Total words :", count)
```

Enter a line of text : Python is fun.  
Python  
is  
fun.

Total words : 3

19. Write a program to input a string and print number of upper- and lower-case letters in it.

Solution.

```
str1 = input("Enter a string: ")
ucase, lcase = 0, 0
for ch in str1:
 if ch >= 'A' and ch <= 'Z':
 ucase += 1
 if ch >= 'a' and ch <= 'z':
 lcase += 1
print("No.of uppercase letters: ", ucase)
print("No.of lowercase letters: ", lcase)
```

Enter a string: AbrAcAbaRA  
No.of uppercase letters: 5  
No.of lowercase letters: 6

20. Write a program to input a string and print each individual word of it along with its length.

Solution.

```
strg = input("Enter a string :")
stlst = strg.split(' ')
iterate words in string
for word in stlst:
 print(word, " (", len(word), ")")
```

Enter a string :nema the fish  
nema ( 4 )  
the ( 3 )  
fish ( 4 )

21. Write a program to input a string and check if contains a digit or not.

Solution.

```
strg = input("Enter a string :")
test = False
dig = "0123456789"
for ch in strg:
 if ch in dig:
 print("The string contains a digit.a")
 test = True
 break
if test == False:
 print("The string doesn't contain a digit.")
```

Enter a string :rf67  
The string contains a digit.

Enter a string :yeqi  
The string doesn't contain a digit.

22. Write a program to input an address containing a pincode and extract and print the pincode from the input address.

Solution.

```
add = input("Enter text:")
i = 0
ln = len(add)
for ch in add:
while i < ln:
 ch = add[i]
 if ch.isdigit():
 sub = add[i:(i+6)]
 if sub.isdigit():
 print(sub)
 break
 i += 1
```

Enter text:New Delhi, 110001  
110001

## Guidelines to NCERT Questions [NCERT Chapter 8]

1. Consider the following string mySubject :

```
mySubject = "Computer Science"
```

What will be the output of the following string operations :

- |                                            |                                         |
|--------------------------------------------|-----------------------------------------|
| (i) print(mySubject[0:len(mySubject)])     | (ii) print(mySubject[-7 : -1])          |
| (iii) print(mySubject[::-2])               | (iv) print(mySubject[len(mySubject)-1]) |
| (v) print(2*mysubject)                     | (vi) print(mySubject[::-2])             |
| (vii) print(mySubject[:3] + mySubject[3:]) | (viii) print(mySubject.swapcase( ))     |
| (ix) print(mySubject.startswith('Comp'))   | (x) print(mySubject.isalpha( ))         |

Ans.

- |                                      |             |                |                        |
|--------------------------------------|-------------|----------------|------------------------|
| (i) Computer Science                 | (ii) Scienc | (iii) Cmue cec | (iv) e                 |
| (v) Computer ScienceComputer Science |             | (vi) eniSrtpo  | (vii) Computer Science |
| (viii) cOMPUTER sCIENCE              | (ix) True   | (x) False      |                        |

2. Consider the following string myAddress :

```
myAddress = "WAZ-1, New Ganga Nagar, New Delhi"
```

What will be the output of following string operations :

- |                                        |                                               |
|----------------------------------------|-----------------------------------------------|
| (i) print(myAddress.lower( ))          | (ii) print(myAddress.upper( ))                |
| (iii) print(myAddress.count('New'))    | (iv) print(myAddress.find('New'))             |
| (v) print(myAddress.rfind('New'))      | (vi) print(myAddress.split(' , '))            |
| (vii) print(myAddress.split(' '))      | (viii) print(myAddress.replace('New', 'Old')) |
| (ix) print(myAddress.partition(' , ')) | (x) print(myAddress.index('Agra'))            |

Ans.

- |                                                                  |                                                       |        |                                               |
|------------------------------------------------------------------|-------------------------------------------------------|--------|-----------------------------------------------|
| (i) wz-1,new ganga nagar, new delhi                              | (ii) WZ-1,NEW GANGA NAGAR, NEW DELHI                  |        |                                               |
| (iii) 2                                                          | (iv) 5                                                | (v) 21 | (vi) ['WZ-1', 'New Ganga Nagar', 'New Delhi'] |
| (vii) ['WZ-1', 'New', 'Ganga', 'Nagar', 'New', 'Delhi']          |                                                       |        |                                               |
| (viii) WZ-1,Old Ganga Nagar,Old Delhi                            | (ix) ('WZ-1', ' ', ' ', 'New Ganga Nagar, New Delhi') |        |                                               |
| (x) It will raise an exception : ValueError: substring not found |                                                       |        |                                               |

### NCERT PROGRAMMING PROBLEMS

1. Write a program to input line(s) of text from the user until enter is pressed. Count the total number of characters in the text (including white spaces), total number of alphabets, total number of digits, total number of special symbols and total number of words in the given text. (assume that each word is separated by one space).

Ans. Similar to program 10.10, page 351.

2. Write a program to convert a string with more than one word into title case string where string is passed as parameter. (Title case means that the first letter of each word is capitalised)

```
s = input("Enter a string :")
print("String With Title Case is :")
print(s.title())
```

Sample run of the program is :

```
Enter a string : i love programming.
String With Title Case is :
I Love Programming.
```

3. Write a program which takes two inputs one is a string and other is a character. The function should create a new string after deleting all occurrences of the character from the string and return the new string.

Ans.

```
s = input("Enter a string :")
c = input("Enter a character :")
print("String With deleted characters is:")
print(s.replace(c, ' '))
```

Sample run of the program is :

```
Enter a string : 'abracadabra'
Enter a character : a
String With deleted characters is:
'brcdbr'
```

4. Input a string having some digits. Write a program to calculate the sum of digits present in this string.

Ans. s = input("Enter a string :")

```
dsum = 0
for a in s :
 if a.isdigit() :
 dsum += int(a)
print("Sum of string's digits is", dsum)
```

Sample run of the program is :

```
Enter a string : hello123
Sum of string's digits is 6
```

5. Write a program that takes a sentence as an input where each word in the sentence is separated by a space. the function should replace each blank with a hyphen and then print the modified sentence.

Ans. s = input("Enter a string :")
print("Replaced String is:")
print(s.replace(' ', '-'))

Sample run of the program is :

```
Enter a string : hello123
Sum of string's digits is 6
```

## GLOSSARY

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <b>Index</b>        | A variable or value used to select a member character from a string. |
| <b>String slice</b> | A part of a string (substring) specified by a range of indices.      |
| <b>Subscript</b>    | Index.                                                               |
| <b>Traversal</b>    | Iterating through a sequence such as a string, member by member.     |

## Assignments

### Type A : Short Answer Questions/Conceptual Questions

- Write a Python script that traverses through an input string and prints its characters in different lines – two characters per line.
- Out of the following operators, which ones can be used with strings ?  
 $=, -, *, /, //, %, >, <, \in, \text{not in}, \leq$
- What is the result of following statement, if the input is 'Fun' ?  
`print ( input("...") + "trial" + "Ooty" * 3 )`
- Which of the following is not a Python legal string operation ?  
(a) 'abc' + 'abc'      (b) 'abc' \* 3      (c) 'abc' + 3      (d) 'abc'.lower()
- Can you say strings are character lists ? Why ? Why not ?

For  
Solutions for  
Selected Questions



Scan  
QR Code

6. Given a string  $S = \text{"CARPE DIEM"}$ . If  $n$  is  $\text{length}/2$  ( $\text{length}$  is the length of the given string), then what would following return ?  
 (a)  $S[:n]$    (b)  $S[n:]$    (c)  $S[n:n]$    (d)  $S[1:n]$    (e)  $S[n:\text{length} - 1]$
7. From the string  $S = \text{"CARPE DIEM"}$ , which ranges return "DIE" and "CAR" ?
8. What happens when from a string slice you skip the **start** and/or **end** values of the slice ?
9. What would the following expressions return ?
- |                                     |                                   |
|-------------------------------------|-----------------------------------|
| (a) "Hello World".upper().lower()   | (b) "Hello World".lower().upper() |
| (c) "Hello World".find("Wor", 1, 6) | (d) "Hello World".find("Wor")     |
| (e) "Hello World".find("wor")       | (f) "Hello World".isalpha()       |
| (g) "Hello World".isalnum()         | (h) "1234".isdigit()              |
| (i) "123FGH".isdigit()              |                                   |
10. Which functions would you choose to use to remove leading and trailing white spaces from a given string?
11. Try to find out if for any case, the string functions **isalnum()** and **isalpha()** return the same result.
12. Suggest appropriate functions for the following tasks :
- (i) To check whether the string contains digits
  - (ii) To find for the occurrence a string within another string
  - (iii) To convert the first letter of a string to upper case      (iv) to capitalize all the letters of the string
  - (v) to check whether all letters of the string are in capital letters
  - (vi) to remove from right of a string all string-combinations from a given set of letters
  - (vii) to remove all white spaces from the beginning of a string
13. In a string slice, the **start** and **end** values can be beyond limits. Why ?
14. Can you specify an out of bound index when accessing a single character from a string ? Why ?
15. Can you add two strings ? What effect does '+' have on strings ?

### Type B : Application Based Questions

1. What is the result of the following expressions ?
- |                                                                                                     |                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (a) <code>print(" " "</code><br>1<br>2<br>3<br>" " ")                                               | (d) <code>s = '0123456789'<br/>print(s[3], ", ", s[0:3], " - ", s[2:5])<br/>print(s[:3], " - ", s[3:], ", ", s[3:100])<br/>print(s[20:], s[2:1], s[1:1])</code> |
| (b) <code>text = "Test.\nNext line."<br/>print(text)</code>                                         | (e) <code>s = '987654321'<br/>print(s[-1], s[-3])<br/>print(s[-3:], s[:-3])<br/>print(s[-100:-3], s[-100:3])</code>                                             |
| (c) <code>print('One', ' Two' * 2)<br/>print('One' + 'Two' * 2)<br/>print(len('0123456789'))</code> |                                                                                                                                                                 |
2. What will be the output produced by following code fragments ?
- |                                                                                                                            |                                                                                                                                          |                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| (a) <code>y = str(123)<br/>x = "hello" * 3<br/>print(x, y)<br/>x = "hello" + "world"<br/>y = len(x)<br/>print(y, x)</code> | (b) <code>x = "hello" + \<br/>"to Python" + \<br/>"world"<br/>for char in x :<br/>    y = char<br/>    print(y, ' : ', end = ' ')</code> | (c) <code>x = "hello world"<br/>print(x[:2], x[:-2], x[-2:])<br/>print(x[6], x[2:4])<br/>print(x[2:-3], x[-4:-2])</code> |
|----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|

3. Carefully go through the code given below and answer the questions based on it :

```

theStr = " This is a test "
inputStr = input(" Enter integer: ")
inputInt = int(inputStr)
testStr = theStr
while inputInt >= 0 :
 testStr = testStr[1:-1]
 inputInt = inputInt - 1
testBool = 't' in testStr
print (theStr) # Line 1
print (testStr) # Line 2
print (inputInt) # Line 3
print (testBool) # Line 4

```

- (i) Given the input integer 3, what output is produced by Line 1 ?  
 (a) This is a test    (b) This is a    (c) is a test  
 (d) is a              (e) None of these

- (ii) Given the input integer 3, what output is produced by Line 2 ?  
 (a) This is a test    (b) s is a t    (c) is a test  
 (d) is a              (e) None of these

- (iii) Given the input integer 2, what output is produced by Line 3 ?  
 (a) 0                  (b) 1                  (c) 2  
 (d) 3                  (e) None of these

- (iv) Given the input integer 2, what output is produced by Line 4 ?  
 (a) False             (b) True             (c) 0  
 (d) 1                  (e) None of these

4. Carefully go through the code given below and answer the questions based on it :

```

testStr = "abcdefghijkl"
inputStr = input ("Enter integer:")
inputInt = int(inputStr)
count = 2
newStr = ''
while count <= inputInt :
 newStr = newStr + testStr[0 : count]
 testStr = testStr[2:] #Line 1
 count = count + 1
print (newStr) # Line 2
print (testStr) # Line 3
print (count) # Line 4
print (inputInt) # Line 5

```

# Path Wala

- (i) Given the input integer 4, what output is produced by Line 2 ?  
 (a) abcdefg    (b) aabbccddeeffgg  
 (c) abcdeefgh    (d) ghi    (e) None of these
- (ii) Given the input integer 4, what output is produced by Line 3 ?  
 (a) abcdefg    (b) aabbccddeeffgg  
 (c) abcdeefgh    (d) ghi    (e) None of these
- (iii) Given the input integer 3, what output is produced by Line 4 ?  
 (a) 0              (b) 1              (c) 2  
 (d) 3              (e) None of these
- (iv) Given the input integer 3, what output is produced by Line 5 ?  
 (a) 0              (b) 1              (c) 2  
 (d) 3              (e) None of these
- (v) Which statement is equivalent to the statement found in Line 1 ?  
 (a) testStr = testStr[2:0]  
 (b) testStr = testStr[2:-1]  
 (c) testStr = testStr[2:-2]  
 (d) testStr = testStr - 2  
 (e) None of these

5. Carefully go through the code given below and answer the questions based on it :

```

inputStr = input(" Give me a string:")
bigInt = 0
littleInt = 0
otherInt = 0
for ele in inputStr:
 if ele >= 'a' and ele <= 'm': # Line 1
 littleInt = littleInt + 1
 elif ele > 'm' and ele <= 'z':
 bigInt = bigInt + 1
 else:
 otherInt = otherInt + 1
print (bigInt) # Line 2
print (littleInt) # Line 3
print (otherInt) # Line 4
print (inputStr.isdigit()) # Line 5

```

- (i) Given the input abcd what output is produced by Line 2 ?  
 (a) 0    (b) 1    (c) 2    (d) 3    (e) 4

- (ii) Given the input *Hi Mom* what output is produced by Line 3 ?  
 (a) 0      (b) 1      (c) 2      (d) 3  
 (e) None of these
- (iii) Given the input *Hi Mom* what output is produced by Line 4 ?  
 (a) 0      (b) 1      (c) 2      (d) 3  
 (e) None of these
- (iv) Given the input  $1+2=3$  what output is produced by Line 5 ?  
 (a) 0      (b) 1      (c) True      (d) False  
 (e) None of these
- (v) Give the input *Hi Mom*, what changes result from modifying Line 1 from  
 $\text{if ele} \geqslant 'a' \text{ and ele} \leqslant 'm'$  to the expression  
 $\text{if ele} \geqslant 'a' \text{ and ele} < 'm'$ ?  
 (a) No change      (b) otherInt would be larger  
 (c) littleInt would be larger  
 (d) bigInt would be larger  
 (e) None of these
6. Carefully go through the code given below and answer the questions based on it :
- ```

in1Str = input(" Enter string of digits: ")
in2Str = input(" Enter string of digits: ")

if len(in1Str)>len(in2Str):
    small = in2Str
    large = in1Str
else :
    small = in1Str
    large = in2Str
newStr = ''
for element in small:
    result = int(element) + int(large[0])
    newStr = newStr + str(result)
    large = large[1:]
print (len(newStr))      # Line 1
print (newStr)            # Line 2
print (large)             # Line 3
print (small)             # Line 4
    
```
- (i) Given a first input of 12345 and a second input of 246, what result is produced by Line 1 ?
 (a) 1 (b) 3 (c) 5 (d) 0
 (e) None of these
- (ii) Given a first input of 12345 and a second input of 246, what result is produced by Line 2 ?
 (a) 369 (b) 246 (c) 234 (d) 345
 (e) None of these
- (iii) Given a first input of 123 and a second input of 4567, what result is produced by Line 3 ?
 (a) 3 (b) 7 (c) 12 (d) 45
 (e) None of these
- (iv) Given a first input of 123 and a second input of 4567, what result is produced by Line 4 ?
 (a) 123 (b) 4567 (c) 7 (d) 3
 (e) None of these
7. Find the output if the input string is 'Test'.
 (a) S = input("Enter String :")
 RS = " "
 for ch in S :
 RS = ch + RS
 print(S + RS)
- (b) S = input("Enter string :")
 RS = " "
 for ch in S :
 RS = ch + 2 + RS
 print(RS + S)
8. Find the errors. Find the line numbers causing errors.
- (a) 1. S = "PURA VIDA"
 2. print(S[9] + S[9 : 15])
- (b) 1. S = "PURA VIDA"
 2. S1 = S[: 10] + S[10 :]
 3. S2 = S[10] + S[-10]
- (c) 1. S = "PURA VIDA"
 2. S1 = S * 2
 3. S2 = S1[-19] + S1[-20]
 4. S3 = S1[-19 :]
- (d) 1. S = "PURA VIDA"
 2. S1 = S[: 5]
 3. S2 = S[5 :]
 4. S3 = S1 * S2
 5. S4 = S2 + '3'
 6. S5 = S1 + 3
9. What is the output produced ?
 (i) >>> "whenever".find("never")
 (ii) >>> "whenever".find("what")

10. What is the output produced ?
 (i) `>>> "-".join(['123', '365', '1319'])`
11. Given a string S, write expressions to print
 (i) first five characters of S
 (ii) Ninth character of S
 (iii) reversed S
 (iv) alternate characters from reversed S

Type C : Programming Practice/Knowledge based Questions

- Write a program to count the number of times a character occurs in the given string.
- Write a program which replaces all vowels in the string with '*'.
- Write a program which reverses a string and stores the reversed string in a new string.
- Write a program that prompts for a phone number of 10 digits and two dashes, with dashes after the area code and the next three numbers. *For example*, 017-555-1212 is a legal input.
Display if the phone number entered is valid format or not and display if the phone number is valid or not (*i.e.*, contains just the digits and dash at specific places).
- Write a program that should do the following :
 - ▲ prompt the user for a string ▲ extract all the digits from the string.
 - ▲ If there are digits :
 - ▲ sum the collected digits together
 - ▲ print out :
 - the original string
 - the digits
 - the sum of the digits
 - ▲ If there are no digits:
 - ▲ print the original string and a message "has no digits"

Sample

▲ given the input : abc123

prints abc123 has the digits 123 which sum to 6

▲ given the input : abcd

prints abcd has no digits

- Write a program that should prompt the user to type some sentence(s) followed by "enter". It should then print the original sentence(s) and the following statistics relating to the sentence(s) :
 - ▲ Number of words
 - ▲ Number of characters (including white-space and punctuation)
 - ▲ Percentage of characters that are alphanumeric

Hints

▲ Assume any consecutive sequence of non-blank characters is a word.

- Write a Python program as per specifications given below :

▲ Repeatedly prompt for a sentence (string) or for 'q' to quit.

▲ Upon input of a sentence s, print the string produced from s by converting each lower case letter to upper case and each upper case letter to lower case.

▲ All other characters are left unchanged.

For example,

Please enter a sentence, or 'q' to quit : This is the Bomb!

tHIS IS THE bOMB!

Please enter a sentence, or 'q' to quit : What's up Doc ???

WHAT'S UP DOC ???

Please enter a sentence, or 'q' to quit : q

Please enter a sentence, or 'q' to quit : q

Path Wala

8. Write a program that does the following :
- ▲ takes two inputs : the first, an integer and the second, a string
 - ▲ from the input string extract all the digits, in the order they occurred, from the string.
 - ▲ if no digits occur, set the extracted digits to 0
 - ▲ add the integer input and the digits extracted from the string together as integers
 - ▲ print a string of the form :
"integer_input + string_digits = sum"

For example :

For inputs 12, ' abc123 ' → ' 12 + 123 = 135 '

For inputs 20, ' a5b6c7 ' → ' 20 + 567 = 587 '

For inputs 100, ' hi mom ' → ' 100 + 0 = 100 '

9. On what principles does Python compare two strings ? Write a program that takes two strings from the user and displays the smaller string in single line and the larger string as per this format :

1st letter	last letter
2nd letter	2nd last letter
3rd letter	3rd last letter

:

For example,

if the two strings entered are Python and PANDA then the output of the program should be :

PANDA	
P	n
y	o
t	h

10. Write a program to convert a given number into equivalent Roman number (store its value as a string). You can use following guidelines to develop solution for it :

- ▲ From the given number, pick successive digits, using %10 and /10 to gather the digits from right to left.
- ▲ The rules for Roman Numerals involve using four pairs of symbols for ones and five, tens and fifties, hundreds and five hundreds. An additional symbol for thousands covers all the relevant bases.
- ▲ When a number is followed by the same or smaller number, it means addition. "II" is two 1's = 2. "VI" is 5 + 1 = 6.
- ▲ When one number is followed by a larger number, it means subtraction. "IX" is 1 before 10 = 9. "IX" isn't allowed, this would be "VIII". For numbers from 1 to 9, the symbols are "I" and "V", and the coding works like this. "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX".
- ▲ The same rules work for numbers from 10 to 90, using "X" and "L". For numbers from 100 to 900, using the symbols "C" and "D". For numbers between 1000 and 4000, using "M".

Here are some examples. 1994 = MCMXCIV, 1956 = MCMLVI, 3888= MMMDCCCLXXXVIII

11. Write a program that asks the user for a string (only single space between words) and returns an estimate of how many words are in the string. (**Hint.** Count number of spaces)
12. Write a program to input a formula with some brackets and checks, and prints out if the formula has the same number of opening and closing parentheses.
13. Write a program that inputs a line of text and prints out the count of vowels in it.
14. Write a program to input a line of text and print the biggest word (length wise) from it.
15. Write a program to input a line of text and create a new line of text where each word of input line is reversed.