

String

String

A string literal is specified with single or double quotes.

```
>>> a = "IIT Kharagpur"
>>> a
'IIT Kharagpur'
>>> b = 'IIT Bombay'
>>> b
'IIT Bombay'
>>> e = "" # empty string
>>> e
''
```

Operations

```
>>> c = a + ' and ' + b
>>> c
'IIT Kharagpur and IIT Bombay'
>>> a + e
'IIT Kharagpur'
>>> d = 3*a
>>> d
'IIT KharagpurIIT KharagpurIIT Kharagpur'
>>> 'p' in a
True
>>> 'q' in a
False
```

Representation and Indices

a:

	0	1	2									12
I	I	T		K	h	a	r	a	g	p	u	r
-13				-9							-2	-1

Note

The content of a string cannot be modified. It is an **immutable** object.

```
>>> a = 'cat'
```

```
>>> a[1]
```

```
'a'
```

```
>>> a[1] = 'o'
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'str' object does not support  
        item assignment
```

Slicing a String

```
>>> a[0]
'I'
>>> a[5]
'h'
>>> a[0:7]
'IIT Kha'
>>> a[: ]
'IIT Kharagpur'
>>> a[2: ]
'T Kharagpur'
>>> a[-10:-2]
' Kharagp'
```

Slicing a String

```
>>> a[0:13:2]
'ITKaapr'
>>> len(a)
13
```

Note

A large number of functions are supported by Python string (`str`) library.

docs.python.org/2/library/string.html

Note

A string is a sequence of characters. It is an **iterator** in Python. Essentially there is a notion of first character (index 0), last character (index $len(s) - 1$) and next character (index $i + 1$). This can be used to **iterate** over a sequence of code.

Python `for` Statement

A `for`-statement in Python is used to `iterate` over the `elements of a sequence` e.g. `string`, `list`, `tuple` etc. The structure or syntax of `for`-statement is as follows.

```
for target in iterator-exp:
```

```
    statement1
```

```
else:
```

```
    statement2
```

Note

The *iterator-exp* is evaluated before entering the loop. The value should be an **iterable object** (a sequence). Number of iterations are determined by the number of elements of the sequence. In each iteration element of the sequence according to their order of indices is assigned to the *target* and *statement₁* is executed.

Note

The loop terminates at the end of the sequence. The `else` part is optional. It is executed after a normal termination of the loop. Both *statement₁* and *statement₂* may be block of statements.

The effect of `break` and `continue` are similar to `while`-loop.

Example

```
# numAs.py : number of A's in a string
s = input("Enter a string: ")
aCount = 0
for x in s:
    if x == 'a' or x == 'A':
        aCount = aCount + 1
print s, ':', aCount, "a's"
```

Assignment 9

Write a Python program that reads a non-negative integer n and computes $n!$, where $0! = 1$ and $n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$ if $n > 0$.

Assignment 10

Write a Python program that reads a positive integer n and reports whether it is a prime number.

First 10 prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31.

Assignment 11

Write a Python program that reads a string `s` of characters and forms a new string with the characters in the odd-indices of `s`.

Input: IIT Kharagpur

Output: I hrgu

Note: `range(len(s))` will give you the sequence of index, and you may use `for`-loop.

Assignment 12

Write a Python program that reads a string **s** of characters and forms a new string by concatenation of **s** and **s reverse**. It will form an even length palindrome.

Input: IIT Kharagpur

Output: IIT KharagpurrupgarahK TII