# Python String

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

In Python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in Python to create a string.

**Syntax:**

```
str = "Hi Python !"
```

Here, if we check the type of the variable **str** using a Python script

***print**(type(str)), then it will **print** a string (str).*

In Python, strings are treated as the sequence of characters, which means that Python doesn't support the character data-type; instead, a single character written as 'p' is treated as the string of length 1.

## Creating String in Python

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or **docstrings**.

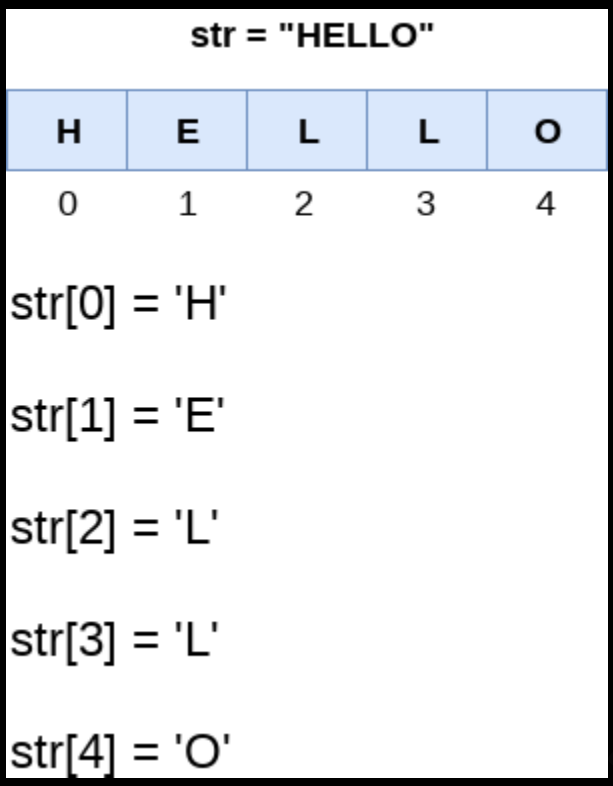
```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = '''''Triple quotes are generally used for
    represent the multiline or
    docstring'''
print(str3)
```

***Output:***

```
Hello Python
Hello Python
Triple quotes are generally used for
    represent the multiline or
    docstring
```

## Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.




Consider the following example:

```
str = "HELLO"
print(str[0])
print(str[1])
print(str[2])
print(str[3])
print(str[4])
# It returns the IndexError because 6th index doesn't exist
print(str[6])
```

***Output:***

```
H
E
L
L
O
IndexError: string index out of range
```

As shown in Python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. Consider the following example.

str = "HELLO"

| H | E | L | L | O |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

str[0] = 'H'
str[1] = 'E'
str[2] = 'L'
str[3] = 'L'
str[4] = 'O'

str[:] = 'HELLO'
str[0:] = 'HELLO'
str[:5] = 'HELLO'
str[:3] = 'HEL'
str[0:2] = 'HE'
str[1:4] = 'ELL'

Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

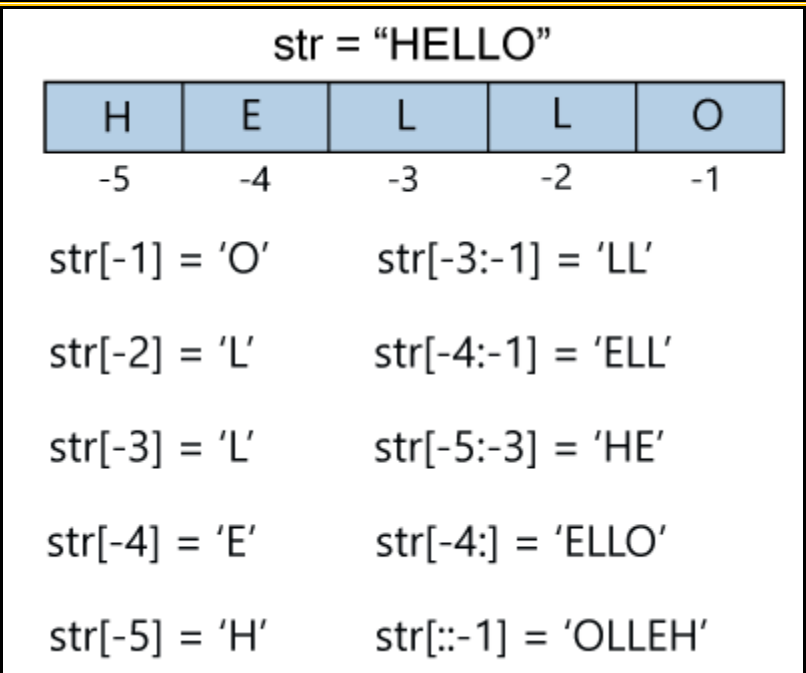
**Consider the following example:**

```
# Given String
str = "JAVATPOINT"
# Start Oth index to end
print(str[0:])
# Starts 1th index to 4th index
print(str[1:5])
# Starts 2nd index to 3rd index
print(str[2:4])
# Starts 0th to 2nd index
print(str[:3])
#Starts 4th to 6th index
print(str[4:7])
```

***Output:***

```
JAVATPOINT
AVAT
VA
JAV
TPO
```

We can do the negative slicing in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on. Consider the following image.




**Consider the following example**

```
str = 'JAVATPOINT'
print(str[-1])
print(str[-3])
print(str[-2:])
print(str[-4:-1])
print(str[-7:-2])
# Reversing the given string
print(str[::-1])
print(str[-12])
```

***Output:***

```
T
I
NT
OIN
ATPOI
TNIOPTAVAJ
IndexError: string index out of range
```

## Reassigning Strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.
Consider the following example.

### Example 1

```
str = "HELLO"
str[0] = "h"
print(str)
```

**Output:**

```
Traceback (most recent call last):
  File "12.py", line 2, in <module>
    str[0] = "h";
TypeError: 'str' object does not support item assignment
```

However, in example 1, the string **str** can be assigned completely to a new content as specified in the following example.

### Example 2

```
str = "HELLO"
print(str)
str = "hello"
print(str)
```

**Output:**

```
HELLO
hello
```

## Deleting the String

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
str = "JAVATPOINT"
del str[1]
```

***Output:***

```
TypeError: 'str' object doesn't support item deletion
```

**Now we are deleting entire string.**

```
str1 = "JAVATPOINT"
del str1
print(str1)
```

***Output:***

```
NameError: name 'str1' is not defined
```

## String Operators

| Operator | Description |
|---|---|
| + | It is known as concatenation operator used to join the strings given either side of the operator. |
| * | It is known as repetition operator. It concatenates the multiple copies of the same string. |
| [] | It is known as slice operator. It is used to access the sub-strings of a particular string. |
| [:] | It is known as range slice operator. It is used to access the characters from the specified range. |
| in | It is known as membership operator. It returns if a particular sub-string is present in the specified string. |
| not in | It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string. |
| r/R | It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string. |
| % | It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python. |

**Example**

Consider the following example to understand the real use of Python operators.

```
str = "Hello"
str1 = " world"
print(str*3) # prints HelloHelloHello
print(str+str1)# prints Hello world
print(str[4]) # prints o
print(str[2:4]); # prints ll
print('w' in str) # prints false as w is not present in str
print('wo' not in str1) # prints false as wo is present in str1.
print(r'C://python37') # prints C://python37 as it is written
print("The string str : %s"%(str)) # prints The string str : Hello
```

**Output:**

```
HelloHelloHello
Hello world
o
ll
False
False
C://python37
The string str : Hello
```

# Python String Formatting

## Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

## Example

Consider the following example to understand the real use of Python operators.

```
str = "They said, "Hello what's going on?""
print(str)
```

**Output:**

```
SyntaxError: invalid syntax
```

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.
The backslash(/) symbol denotes the escape sequence. The backslash can be followed by a special character and it interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

## Example -

```
# using triple quotes
print('''''They said, "What's there?"''''')
# escaping single quotes
print('They said, "What\'s going on?"')
# escaping double quotes
print("They said, \"What's going on?\"")
```

***Output:***

```
They said, "What's there?"
They said, "What's going on?"
They said, "What's going on?"
```

**The list of an escape sequence is given below:**

| Sr. | Escape Sequence | Description | Example |
|---|---|---|---|
| 1. | \newline | It ignores the new line. | print("Python1 \<br>Python2 \<br>Python3")<br>**Output:**<br>Python1 Python2 Python3 |
| 2. | \\ | Backslash | print("\\")<br>**Output:**<br>\ |
| 3. | \' | Single Quotes | print('\'')<br>**Output:**<br>' |

| | | | |
|---|---|---|---|
| 4. | \\'' | Double Quotes | print("\"")<br>**Output:**<br>" |
| 5. | \a | ASCII Bell | print("\a") |
| 6. | \b | ASCII Backspace(BS) | print("Hello \b World")<br>**Output:**<br>Hello World |
| 7. | \f | ASCII Formfeed | print("Hello \f World!")<br>Hello  World! |
| 8. | \n | ASCII Linefeed | print("Hello \n World!")<br>**Output:**<br>Hello<br> World! |
| 9. | \r | ASCII Carriege Return(CR) | print("Hello \r World!")<br>**Output:**<br>World! |
| 10. | \t | ASCII Horizontal Tab | print("Hello \t World!")<br>**Output:**<br>Hello  World! |
| 11. | \v | ASCII Vertical Tab | print("Hello \v World!")<br>**Output:**<br>Hello<br> World! |
| 12. | \ooo | Character with octal value | print("\110\145\154\154\157")<br>**Output:**<br>Hello |
| 13 | \xHH | Character with hex value. | print("\x48\x65\x6c\x6c\x6f")<br>**Output:**<br>Hello |

**Here is the simple example of escape sequence.**

```
print("C:\\Users\\DEVANSH SHARMA\\Python32\\Lib")
print("This is the \n multiline quotes")
print("This is \x48\x45\x58 representation")
```

***Output:***

```
C:\Users\DEVANSH SHARMA\Python32\Lib
This is the
 multiline quotes
This is HEX representation
```

We can ignore the escape sequence from the given string by using the raw string. We can do this by writing **r** or **R** in front of the string. Consider the following example.

```
print(r"C:\\Users\\DEVANSH SHARMA\\Python32")
```

***Output:***

```
C:\\Users\\DEVANSH SHARMA\\Python32
```

## The format() method

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example:

```
# Using Curly braces
print("{} and {} both are the best friend".format("Devansh","Abhishek"))

#Positional Argument
print("{1} and {0} best players ".format("Virat","Rohit"))

#Keyword Argument
print("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))
```

***Output:***

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky
```

## Python String Formatting Using % Operator

Python allows us to use the format specifiers used in C's printf statement. The format specifiers in Python are treated in the same way as they are treated in C. However, Python provides an additional operator %, which is used as an interface between the format specifiers and their values. In other words, we can say that it binds the format specifiers to the values.

Consider the following example.

```
Integer = 10;
Float = 1.290
String = "Devansh"
print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\
nHi I am string ... My value is %s"%(Integer,Float,String))
```

***Output:***

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Devansh
```