Python Strings:
In this tutorial you will learn to create, format, modify and delete strings in Python. Also, you will be introduced to various string operations and functions.

In Python, **Strings** are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

What is String in Python?
A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0s and 1s.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encodings used.

In Python, a string is a sequence of Unicode characters. Unicode was introduced to include every character in all languages and bring uniformity in encoding. You can learn about Unicode from Python Unicode.

How to create a string in Python?
Strings can be created by enclosing characters inside a single quote or double-quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
# defining strings in Python
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

When you run the program, the output will be:

```
Hello
Hello
Hello
Hello, welcome to
           the world of Python
```

How to access characters in a string?

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use floats or other types, this will result in TypeError.

Python allows negative indexing for its sequences.

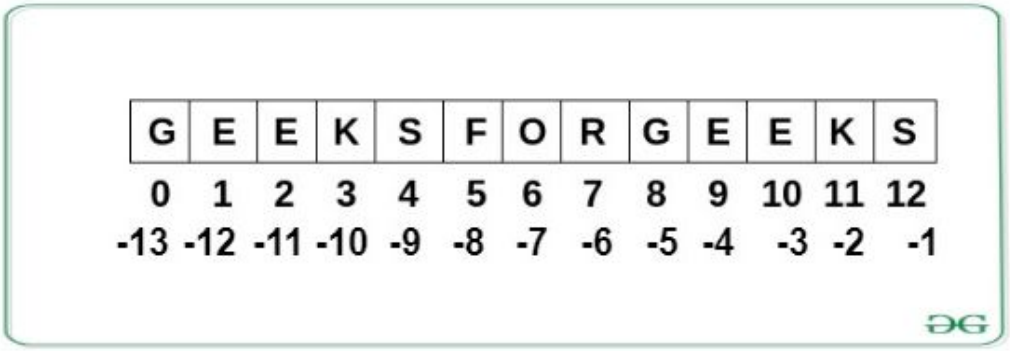
The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator :(colon).

```python
#Accessing string characters in Python
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

Accessing characters in Python

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

While accessing an index out of the range will cause an **IndexError**. Only Integers are allowed to be passed as an index, float or other types will cause a **TypeError**.



When we run the above program, we get the following output:

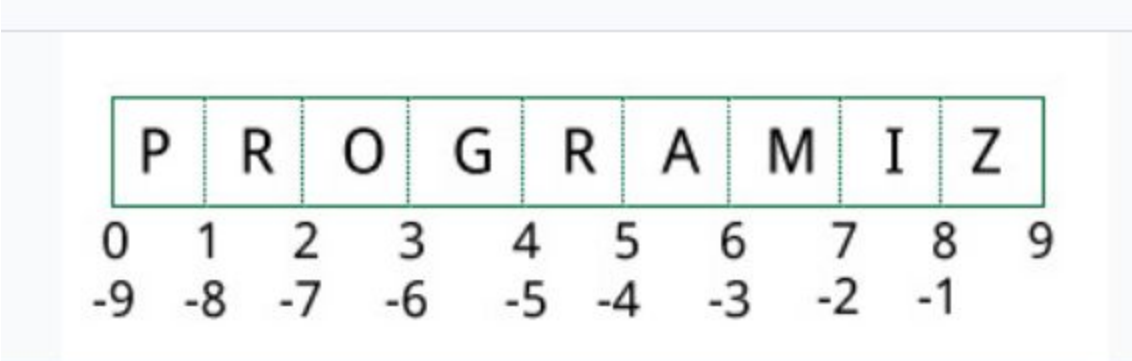
```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
```

If we try to access an index out of the range or use numbers other than an integer, we will get errors.

```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range

# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

Slicing can be best visualized by considering the index to be between the elements as shown below.

If we want to access a range, we need the index that will slice the portion from the string.



String Slicing in Python

How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once they have been assigned. We can simply reassign different strings to the same name.

```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the del keyword.

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

Python String Operations

There are many operations that can be performed with strings which makes it one of the most used data types in Python.

To learn more about the data types available in Python visit: Python Data Types
Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The + operator does this in Python. Simply writing two string literals together also concatenates them.

The * operator can be used to repeat the string for a given number of times.

```
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

When we run the above program, we get the following output:

```
str1 + str2 =  HelloWorld!
str1 * 3 = HelloHelloHello
```

Writing two string literals together also concatenates them like + operator.

If we want to concatenate strings in different lines, we can use parentheses.

```
>>> # two string literals together
>>> 'Hello ''World!'
'Hello World!'

>>> # using parentheses
>>> s = ('Hello '
...      'World')
>>> s
'Hello World'
```

String Membership Test

We can test if a substring exists within a string or not, using the keyword in.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

Built-in functions to Work with Python

Various built-in functions that work with sequence work with strings as well.

Some of the commonly used ones are enumerate() and len(). The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

Similarly, len() returns the length (number of characters) of the string.

```
str = 'cold'

# enumerate()
list_enumerate = list(enumerate(str))
print('list(enumerate(str) = ', list_enumerate)

#character count
print('len(str) = ', len(str))
```

When we run the above program, we get the following output:

```
list(enumerate(str) =  [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
len(str) =  4
```