Python

Playing With Strings
in Python
e d u r e k a !

Python String

A string is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

Example:

```
"Geeksforgeeks" or 'Geeksforgeeks'
```

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.

```
print("A Computer Science portal for geeks")
```

Output:

```
A Computer Science portal for geeks
```

Creating a String in Python

Strings in Python can be created using single quotes or double quotes or even triple quotes.

Python3

```python
# Python Program for
# Creation of String

# Creating a String
# with single Quotes
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a Geek"
print("\nString with the use of Double Quotes: ")
print(String1)

# Creating a String
# with triple Quotes
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Geeks
            For
            Life'''
print("\nCreating a multiline String: ")
print(String1)
```

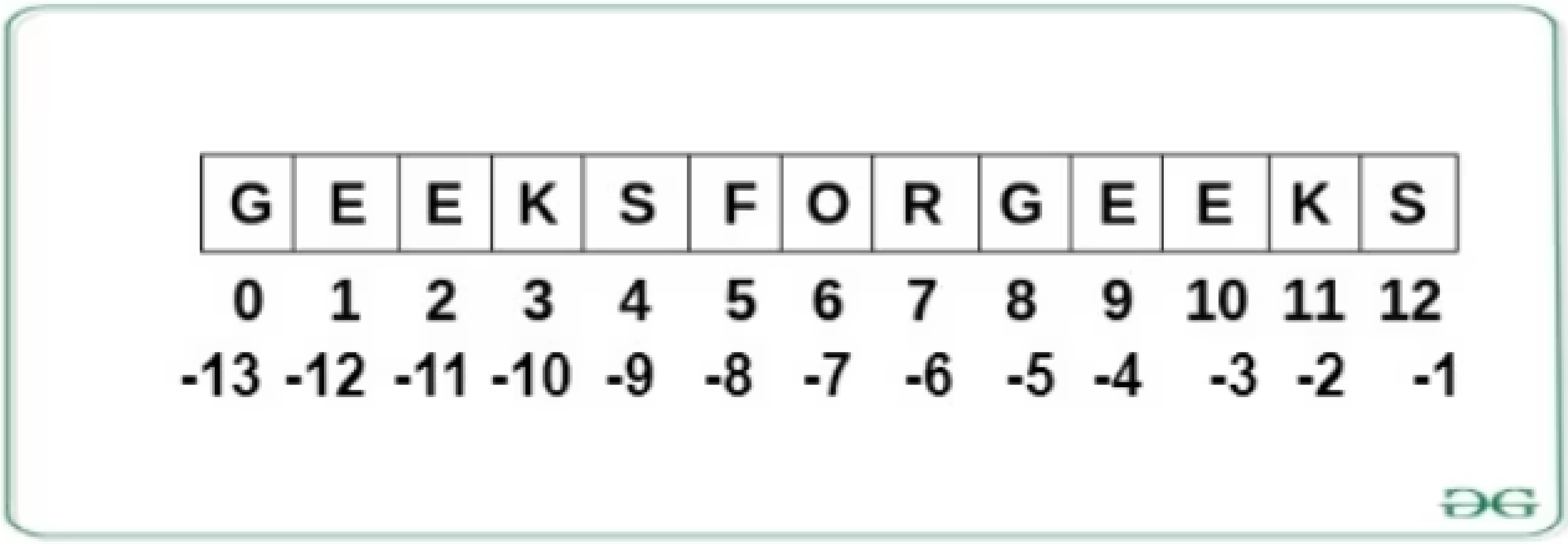
Output :

```
String with the use of Single Quotes:

Welcome to the Geeks World


String with the use of Double Quotes:

I'm a Geek


String with the use of Triple Quotes:

I'm a Geek and I live in a world of "Geeks"


Creating a multiline String:

Geeks

            For

            Life
```

Accessing characters in Python String

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 that will cause a **TypeError**.


G E E K S F O R G E E K S
0 1 2 3 4 5 6 7 8 9 10 11 12
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

```python
# Python Program to Access
# characters of String

String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)
# Printing First character
print("\nFirst character of String is: ")
print(String1[0])
# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

Output:

```
Initial String:

GeeksForGeeks


First character of String is:

G


Last cha racter of String is:

s
```

Reversing a Python String

With Accessing Characters from a string, we can also reverse them. We can Reverse a string by writing [::-1] and the string will be reversed.

```
#Program to reverse a string
gfg = "geeksforgeeks"
print(gfg[::-1])
```

Output:

```
skeegrofskeeg
```

We can also reverse a string by using built-in join and reversed function.

Python3

```
# Program to reverse a string

gfg = "geeksforgeeks"
# Reverse the string using reversed and join function
gfg = "".join(reversed(gfg))

print(gfg)
```

Output:

```
skeegrofskeeg
```

String Slicing

To access a range of characters in the String, the method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

Python3

```python
# Python Program to
# demonstrate String slicing

# Creating a String
String1 = "GeeksForGeeks"
print('Initial String: ')
print(String1)

# Printing 3rd to 12th character
print('\nSlicing characters from 3-12: ')
print(String1[3:12])

# Printing characters between
# 3rd and 2nd last character
print('\nSlicing characters between ' +
      '3rd and 2nd last character: ')
print(String1[3:-2])
```

Output:

```
Initial String:
GeeksForGeeks


Slicing characters from 3-12:
ksForGeek


Slicing characters between 3rd and 2nd last character:
ksForGee
```

Deleting/Updating from a String

In Python, Updation or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of the entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

Updation of a character:

Python3

```
# Python Program to Update
# character of a String

String1 = "Hello, I'm a Geek"
print('Initial String: ')
print(String1)
# Updating a character of the String
## As python strings are immutable, they don't support item updation directly
### there are following two ways
#1
list1 = list(String1)
list1[2] = 'p'
String2 = ''.join(list1)
print('\nUpdating character at 2nd Index: ')
print(String2)
#2
String3 = String1[0:2] + 'p' + String1[3:]
print(String3)
```

Error:

```
Traceback (most recent call last):
File "/home/360bb1830c83a918fc78aa8979195653.py", line 10, in
String1[2] = 'p'
TypeError: 'str' object does not support item assignment
```

Updating Entire String:

```
# Python Program to Update
# entire String

String1 = "Hello, I'm a Geek"
print("Initial String: ")
print(String1)
# Updating a String
String1 = "Welcome to the Geek World"
print("\nUpdated String: ")
print(String1)
```

Output:

```
Initial String:
Hello, I'm a Geek


Updated String:
Welcome to the Geek World
```

Deletion of a character:

Python3

```
# Python Program to Delete
# characters from a String

String1 = "Hello, I'm a Geek"
print("Initial String: ")
print(String1)

# Deleting a character
# of the String
String2 = String1[0:2] + String1[3:]
print("\nDeleting character at 2nd Index: ")
print(String2)
```

Error:

```
Initial String:
Hello, I'm a Geek

Deleting character at 2nd Index:
Helo, I'm a Geek
```

Deleting Entire String:

Deletion of the entire string is possible with the use of del keyword. Further, if we try to print the string, this will produce an error because String is deleted and is unavailable to be printed.

Python3

```
# Python Program to Delete
# entire String

String1 = 'Hello, I'm a Geek'
print("Initial String: ")
print(String1)
# Deleting a String
# with the use of del
del String1
print("\nDeleting entire String: ")
print(String1)
```

Error:

```
Traceback (most recent call last):
File "/home/e4b8f2170f140da99d2fe57d9d8c6a94.py", line 12, in
print(String1)
NameError: name 'String1' is not defined
```

Escape Sequencing in Python

While printing Strings with single and double quotes in it causes **SyntaxError** because String already contains Single and Double Quotes and hence cannot be printed with the use of either of these. Hence, to print such a String either Triple Quotes are used or Escape sequences are used to print such Strings.

Escape sequences start with a backslash and can be interpreted differently. If single quotes are used to represent a string, then all the single quotes present in the string must be escaped and same is done for Double Quotes.

```python
# Python Program for
# Escape Sequencing
# of String

# Initial String
String1 = '''I'm a "Geek"'''

print("Initial String with use of Triple Quotes: ")

print(String1)

# Escaping Single Quote

String1 = 'I\'m a "Geek"'

print("\nEscaping Single Quote: ")

print(String1)

# Escaping Double Quotes

String1 = "I'm a \"Geek\""

print("\nEscaping Double Quotes: ")

print(String1)

# Printing Paths with the
# use of Escape Sequences

String1 = "C:\\Python\\Geeks\\"

print("\nEscaping Backslashes: ")

print(String1)

# Printing Paths with the
# use of Tab

String1 = "Hi\tGeeks"

print("\nTab: ")

print(String1)

# Printing Paths with the
# use of New Line

String1 = "Python\nGeeks"

print("\nNew Line: ")

print(String1)
```

Output:

```
Initial String with use of Triple Quotes:

I'm a "Geek"

Escaping Single Quote:

I'm a "Geek"

Escaping Double Quotes:

I'm a "Geek"

Escaping Backslashes:

C:\Python\Geeks\

Tab:

Hi Geeks

New Line:

Python

Geeks
```

To ignore the escape sequences in a String, **r** or **R** is used, this implies that the string is a raw string and escape sequences inside it are to be ignored.

Python3

```
# Printing hello in octal
String1 = "\110\145\154\154\157"
print("\nPrinting in Octal with the use of Escape Sequences: ")
print(String1)
# Using raw String to
# ignore Escape Sequences
String1 = r"This is \110\145\154\154\157"
print("\nPrinting Raw String in Octal Format: ")
print(String1)
# Printing Geeks in HEX
String1 = "This is \x47\x65\x65\x6b\x73 in \x48\x45\x58"
print("\nPrinting in HEX with the use of Escape Sequences: ")
print(String1)
# Using raw String to
# ignore Escape Sequences
String1 = r"This is \x47\x65\x65\x6b\x73 in \x48\x45\x58"
print("\nPrinting Raw String in HEX Format: ")
print(String1)
```

Output:

```
Printing in Octal with the use of Escape Sequences:

Hello

Printing Raw String in Octal Format:

This is \110\145\154\154\157

Printing in HEX with the use of Escape Sequences:

This is Geeks in HEX

Printing Raw String in HEX Format:

This is \x47\x65\x65\x6b\x73 in \x48\x45\x58
```

Formatting of Strings

Strings in Python can be formatted with the use of format() method which is a very versatile and powerful tool for formatting Strings. Format method in String contains curly braces {} as placeholders which can hold arguments according to position or keyword to specify the order.

Python3

```python
# Python Program for
# Formatting of Strings
# Default order
String1 = "{} {} {}".format('Geeks', 'For', 'Life')
print("Print String in default order: ")
print(String1)
# Positional Formatting
String1 = "{1} {0} {2}".format('Geeks', 'For', 'Life')
print("\nPrint String in Positional order: ")
print(String1)
# Keyword Formatting
String1 = "{l} {f} {g}".format(g='Geeks', f='For', l='Life')
print("\nPrint String in order of Keywords: ")
print(String1)
```

Output:

```
Print String in default order:
Geeks For Life


Print String in Positional order:
For Geeks Life


Print String in order of Keywords:
Life For Geeks
```

Integers such as Binary, hexadecimal, etc., and floats can be rounded or displayed in the exponent form with the use of format specifiers.

Python3

```
# Formatting of Integers
String1 = "{0:b}".format(16)
print("\nBinary representation of 16 is ")
print(String1)
# Formatting of Floats
String1 = "{0:e}".format(165.6458)
print("\nExponent representation of 165.6458 is ")
print(String1)
# Rounding off Integers
String1 = "{0:.2f}".format(1/6)
print("\none-sixth is : ")
print(String1)
```

Output:

```
Binary representation of 16 is
10000


Exponent representation of 165.6458 is
1.656458e+02


one-sixth is :
0.17
```

A string can be left() or center(^) justified with the use of format specifiers, separated by a colon(:).

Python3

```python
# String alignment
String1 = "|{:<10}|{:^10}|{:>10}|".format('Geeks',
                                          'for',
                                          'Geeks')
print("\nLeft, center and right alignment with Formatting: ")
print(String1)
# To demonstrate aligning of spaces
String1 = "\n{0:^16} was founded in {1:<4}!".format("GeeksforGeeks",
                                                    2009)
print(String1)
```

Output:

```
Left, center and right alignment with Formatting:
|Geeks     |   for    |     Geeks|


 GeeksforGeeks   was founded in 2009 !
```

Old style formatting was done without the use of format method by using % operator

Python3

```python
# Python Program for
# Old Style Formatting
# of Integers

Integer1 = 12.3456789
print("Formatting in 3.2f format: ")
print('The value of Integer1 is %3.2f' % Integer1)
print("\nFormatting in 3.4f format: ")
print('The value of Integer1 is %3.4f' % Integer1)
```

Output:

```
Formatting in 3.2f format:

The value of Integer1 is 12.35



Formatting in 3.4f format:

The value of Integer1 is 12.3457
```

Python String constants

Built-In Function	Description
string.ascii_letters	Concatenation of the ascii_lowercase and ascii_uppercase constants.
string.ascii_lowercase	Concatenation of lowercase letters
string.ascii_uppercase	Concatenation of uppercase letters
string.digits	Digit in strings
string.hexdigits	Hexadigit in strings
string.letters	concatenation of the strings lowercase and uppercase
string.lowercase	A string must contain lowercase letters.
string.octdigits	Octadigit in a string
string.punctuation	ASCII characters having punctuation characters.
string.printable	String of characters which are printable
String.endswith()	Returns True if a string ends with the given suffix otherwise returns False
String.startswith()	Returns True if a string starts with the given prefix otherwise returns False
String.isdigit()	Returns "True" if all characters in the string are digits, Otherwise, It returns "False".
String.isalpha()	Returns "True" if all characters in the string are alphabets, Otherwise, It returns "False".
string.isdecimal()	Returns true if all characters in a string are decimal.
str.format()	one of the string formatting methods in Python3, which allows multiple substitutions and value formatting.
String.index	Returns the position of the first occurrence of substring in a string
string.uppercase	A string must contain uppercase letters.
string.whitespace	A string containing all characters that are considered whitespace.
string.swapcase()	Method converts all uppercase characters to lowercase and vice versa of the given string, and returns it
replace()	returns a copy of the string where all occurrences of a substring is replaced with another substring.

Deprecated string functions

Built-In Function	Description
string.Isdecimal	Returns true if all characters in a string are decimal
String.Isalnum	Returns true if all the characters in a given string are alphanumeric.
string.Istitle	Returns True if the string is a title cased string
String.partition	splits the string at the first occurrence of the separator and returns a tuple.
String.Isidentifier	Check whether a string is a valid identifier or not.
String.len	Returns the length of the string.
String.rindex	Returns the highest index of the substring inside the string if substring is found.
String.Max	Returns the highest alphabetical character in a string.
String.min	Returns the minimum alphabetical character in a string.
String.splitlines	Returns a list of lines in the string.
string.capitalize	Return a word with its first character capitalized.
string.expandtabs	Expand tabs in a string replacing them by one or more spaces
string.find	Return the lowest indexing a sub string.
string.rfind	find the highest index.
string.count	Return the number of (non-overlapping) occurrences of substring sub in string
string.lower	Return a copy of s, but with upper case, letters converted to lower case.
string.split	Return a list of the words of the string, If the optional second argument sep is absent or None
string.rsplit()	Return a list of the words of the string s, scanning s from the end.
rpartition()	Method splits the given string into three parts
string.splitfields	Return a list of the words of the string when only used with two arguments.
string.join	Concatenate a list or tuple of words with intervening occurrences of sep.
string.strip()	It returns a copy of the string with both leading and trailing white spaces removed
string.lstrip	Return a copy of the string with leading white spaces removed.
string.rstrip	Return a copy of the string with trailing white spaces removed.
string.swapcase	Converts lower case letters to upper case and vice versa.
string.translate	Translate the characters using table
string.upper	lower case letters converted to upper case.
string.ljust	left-justify in a field of given width.
string.rjust	Right-justify in a field of given width.
string.center()	Center-justify in a field of given width.
string-zfill	Pad a numeric string on the left with zero digits until the given width is reached.
string.replace	Return a copy of string s with all occurrences of substring old replaced by new.
string.casefold()	Returns the string in lowercase which can be used for caseless comparisons.
string.encode	Encodes the string into any encoding supported by Python. The default encoding is utf-8.
string.maketrans	Returns a translation table usable for str.translate()

Advantages of String in Python:

- Strings are used at a larger scale i.e. for a wide areas of operations such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.
- Python has a rich set of string methods that allow you to manipulate and work with strings in a variety of ways. These methods make it easy to perform common tasks such as converting strings to uppercase or lowercase, replacing substrings, and splitting strings into lists.
- Strings are immutable, meaning that once you have created a string, you cannot change it. This can be beneficial in certain situations because it means that you can be confident that the value of a string will not change unexpectedly.
- Python has built-in support for strings, which means that you do not need to import any additional libraries or modules to work with strings. This makes it easy to get started with strings and reduces the complexity of your code.
- Python has a concise syntax for creating and manipulating strings, which makes it easy to write and read code that works with strings.

Drawbacks of String in Python:

- When we are dealing with large text data, strings can be inefficient. For instance, if you need to perform a large number of operations on a string, such as replacing substrings or splitting the string into multiple substrings, it can be slow and consume a lot resources.
- Strings can be difficult to work with when you need to represent complex data structures, such as lists or dictionaries. In these cases, it may be more efficient to use a different data type, such as a list or a dictionary, to represent the data.