Python String Concatenation

String Concatenation is the technique of combining two strings. String Concatenation can be done using many ways.

We can perform string concatenation using the following ways:

1. Using + operator
2. Using join() method
3. Using % operator
4. Using format() function

Using + Operator

The process of combining strings is called string concatenation. In Python, strings can be concatenated in the following ways:

1. Using + operator
2. Using join() method
3. Using % operator
4. Using format() function

Using + Operator

The + operator functions similar to the arithmetic + operator, but here both the operands must be string. It is one of the most easy ways to concatenate strings.

Note: Strings are immutable objects, hence the result of concatenation needs to be stored in a new variable(string object).

```
# Python program to demonstrate
# string concatenation

var1 = "Coding "
var2 = "Ninjas"

# + Operator is used to combine strings
var3 = var1 + var2

print(var3)
```

Output:

```
Coding Ninjas
```

Using join() Method

The ***join()*** string method returns a string by joining all the character elements of an iterable, separated by a string separator.

The join() method provides a very convenient way to join characters of an iterable(such as list, tuple or string objects) such that the elements are joined by a string separator, and at the end return the concatenated string.

Syntax:

```
string.join(iterable)
```

Parameters: The join() method takes iterable – objects capable of returning their members one at a time. Some examples are **List, Tuple, String, Dictionary and Set.**

Return Value: The join() method returns a string concatenated with the elements of iterable.

Type Error: If the iterable contains any non-string values, it raises a TypeError exception.

Example:

```
# Python program to demonstrate string concatenation

var1 = "Coding"
var2 = "Ninjas"
```

```
# join() method is used to join strings with an empty character as a separator("")
var3 = "".join([var1, var2])
print(var3)


# This time we use space(" ") as a separator for join() method.
var3 = " ".join([var1, var2])
print(var3)


li = ['One', 'Two', 'Three']
sep = '-Separator-'
print(sep.join(li))


var = "TESTING"
sep = "*"
print(sep.join(var)) # String is also an iterable as it is a collection of characters.
```

Output:

```
CodingNinjas
Coding Ninjas
One-Separator-Two-Separator-Three
T*E*S*T*I*N*G
```

In the above examples initially var1 stores "Coding" and var2 stores "Ninjas". First, we joined var1 and var2 with an empty character in between them which resulted in "CodingNinjas" and later we joined them with space as a separator. The join method accepts only lists as its argument whose size can be variable.

Using % Operator

We have used % operator for string formatting, but it can also be used for string concatenation. % operator helps both in string concatenation and string formatting.

Here is an example code to demonstrate the use of % for concatenation:

Example:

```
# Python program to demonstrate
# string concatenation

var1 = "Coding"
var2 = "Ninjas"

# % Operator is used here to combine the string stored in var1 and var2
print("%s %s" % (var1, var2))
```

Output:

```
Coding Ninjas
```

Here, the % Operator combines the string that is stored in var1 and var2. The %s denotes string data type. The value in both the variables is passed to the string %s and becomes “Coding Ninjas”.

Using format() function

With Python 3.0, another function, the format() method has been introduced in the python library for handling string formatting which are more complex to be solved by % operator, in an elegant and efficient way.. format() method of the built-in string class provides a solution for complex variable substitutions and value formatting.

The syntax for format() method is:

string.format(var1, var2, var3,...)

Using a Single Formatter :

In Python, formatters work by employing replacement fields and different placeholders defined by a pair of curly braces {}, one curly brace for each string, and then calling {}.format(). The value we wish to concatenate is passed as function parameters to format().

Syntax : { } .format(value)
Parameters :
(value) : Can be an integer, floating-point, string, characters or even variables containing other values.

Return type : Returns a formatted string with the value passed as a parameter in the placeholder position.

Example:

```
# Python3 program to demonstarte
# the str.format() method

# using format option in a simple string
print ("{}, is best for DSA content." .format("CodingNinjas"))

# using format option for a
# value stored in a variable
str = "This article is provided at {}"
print (str.format("CodeStudio"))

# formatting a string using a numeric constant
print ("Hello, I am {} years old".format(20))
```

output:

```
CodingNinjas is best for DSA content.
This article is provided at CodeStudio
Hello, I am 20 years old
```

Using Multiple Formatters :

Similar to what we did in the previous example, we can add more than one placeholder to insert in a string, Python automatically assigns each value to the placeholders in the order they appear. The first value is inserted in the first placeholder's position, the second value is inserted in the second placeholder's position and so on.

Syntax : { } { } .format(value1, value2)

Parameters :
(value1, value2) : Similar to what we saw in single formatters, the values can be integers, floating-point, numeric constants, strings, characters and even variables. The difference lies that instead of one value we can pass multiple values this time.

Errors and Exceptions :
IndexError: Occurs when the number of placeholders and the number of values passed do not match. Python usually assigns each placeholder with the default index in order like 0, 1, 2... Therefore when it encounters a placeholder but cannot find a value for it, it throws IndexError.

Example:

```
# Python program to demonstrate
# string concatenation

var1 = "Coding"
var2 = "Ninjas"

# format function is used here to
# combine the string
print("{} {}".format(var1, var2))
```

Output:

```
Coding Ninjas
```

Here, the format() function combines the strings stored in the var1 and var2. The curly braces {} depict the positions of the string in the result. The first curly brace is the position of the first string(var1), the second brace is the position of the second variable and so on. Finally, it prints the value “Coding Ninjas”.