Python F-strings

Summary: in this tutorial, you'll learn about Python F-strings and how to use them to format strings and make your code more readable.

Introduction to the Python F-strings

Python 3.6 introduced the f-strings that allow you to format text strings faster and more elegant. The f-strings provide a way to embed expressions inside a string literal using a clearer syntax than the `format()` method.

For example:

```
name = 'John'
s = f'Hello, {name}!'
print(s)
```

Output:

```
Hello, John!
```

How it works.

- First, define a variable (https://www.pythontutorial.net/python-basics/python-variables/) with the value `'John'` .
- Then, place the `name` variable inside the curly braces `{}` in the literal string. Note that you need to prefix the string with the letter `f` to indicate that it is an f-string. It's also valid if you use the letter in uppercase (`F`).
- Finally, print out the string s.

It's important to note that Python evaluates the expressions in f-string at runtime. It replaces the expressions inside an f-string by their values.

Python f-string examples

The following example calls the upper() (https://www.pythontutorial.net/python-string-methods/python-string-upper/) method to convert the name to uppercase inside the curly braces of an f-string:

```
name = 'John'
s = F'Hello, {name.upper()}!'
print(s)
```

Output:

```
Hello, JOHN!
```

The following example uses multiple curly braces inside an f-string:

```
first_name = 'John'
last_name = 'Doe'
s = F'Hello, {first_name} {last_name}!'
print(s)
```

Output:

```
Hello, John Doe!
```

This example is equivalent to the above example but use the `join()` (https://www.pythontutorial.net/python-string-methods/python-string-join/) method:

```
first_name = 'John'
last_name = 'Doe'
s = F'Hello, {" ".join((first_name, last_name))}!'

print(s)
```

Output:

```
Hello, John Doe!
```

Multiline f-strings

Python allows you to have multiline f-strings. To create a multiline f-string, you place the letter `f` in each line. For example:

```
name = 'John'
website = 'PythonTutorial.net'

message = (
    f'Hello {name}. '
    f"You're learning Python at {website}."
)

print(message)
```

Output:

```
Hello John. You're learning Python on PythonTutorial.net.
```

If you want to spread an f-string over multiple lines, you can use a backslash (https://www.pythontutorial.net/python-basics/python-backslash/) (\) to escape the return character like this:

```
name = 'John'
website = 'PythonTutorial.net'

message = f'Hello {name}. ' \
          f"You're learning Python at {website}."

print(message)
```

The following example shows how to use triple quotes (`"""`) with an f-string:

```
name = 'John'
website = 'PythonTutorial.net'

message = f"""Hello {name}.
You're learning Python at {website}."""

print(message)
```

Output:

```
Hello John.
You're learning Python at PythonTutorial.net.
```

Curly braces

When evaluating an f-string, Python replaces double curly braces with a single curly brace. However, the doubled curly braces do not signify the start of an expression.

It means that Python will not evaluate the expression inside the double curly brace and replace the double curly braces with a single one. For example:

```
s = f'{{1+2}}'
print(s)
```

Output:

```
{1+2}
```

The following shows an f-string with triple curly braces:

```
s = f'{{{1+2}}}'
print(s)
```

Output:

```
{3}
```

In this example, Python evaluates the {1+2} as an expression, which returns 3. Also, it replaces the remaining doubled curly braces with a single one.

To add more curly braces to the result string, you use more than triple curly braces:

```
s = f'{{{{1+2}}}}'
print(s)
```

Output:

```
{{1+2}}
```

In this example, Python replaces each pair of doubled curly braces with a single curly brace.

Evaluation order of expressions in Python f-strings

Python evaluates the expressions in an f-string in the left-to-right order. This is obvious if the expressions have side effects like the following example:

```
def inc(numbers, value):
    numbers[0] += value
    return numbers[0]

numbers = [0]

s = f'{inc(numbers,1)},{inc(numbers,2)}'
print(s)
```

Output:

```
1,3
```

In this example, the following function call increases the first number in the numbers list by one:

```
inc(numbers,1)
```

After this call, the `numbers[0]` is one. And the second call increases the first number in the numbers list by 2, which results in 3.

Format numbers in f-strings

To format a number in an f-string, you use this simplified syntax:

```
{[number]:[.precision][type]}
```

For example:

```
previous = 99.2
current = 110.3
vs_previous = (current - previous) / previous

print(f'Current vs. previous year: {vs_previous:.2%}')
```

Output:

```
Current vs. previous year: 11.19%
```

In the syntax:

```
{vs_previous:.2%}
```

The vs_previous is the number to format. The `.2%` means to get two digits after decimal and format the result as a percentage.

Python has more sophisticated format rules that you can reference via the following link (https://docs.python.org/3/library/string.html#format-specification-mini-language) .

Summary

- Python f-strings provide a elegant way to format text strings.
- Python replaces the result of an expression embeded inside the curly braces {} in an f-string at runtime.