Updated
2022

A Guide to Formatting with f-strings in Python

Jacqueline J. Masloff, PhD
Thomas M. Connors, MS
Bentley University

Introduction

The release of Python version 3.6 introduced formatted string literals, simply called “f-strings.”

They are called **`f-strings`** because you need to prefix a string with the letter 'f' to create an f-string. The letter 'f' also indicates that these strings are used for formatting. Although there are other ways for formatting strings, the Zen of Python states that simple is better than complex and practicality beats purity--and f-strings are really the most simple and practical way for formatting strings. They are also faster any of the previous more commonly used string formatting mechanisms.

To use formatted string literals, begin a string with f or F before the opening quotation mark or triple quotation mark in a **`print()`** statement. Inside this string, you can write a Python expression between **`{ }`** characters that can refer to variables or literal values. f-strings use the same rules as normal strings, raw strings, and triple quoted strings. The parts of the f-string outside of the curly braces are literal strings. f-strings support extensive modifiers that control the final appearance of the output string. Expressions in f-strings can be modified by a format specification.

Format specifications are used within replacement fields contained within a format string to define how individual values are presented. Each formattable type may define how the format specification is to be interpreted.

This document will explain how to use the many options available with f-strings.

Alignment

There are several ways to align variables in f-strings. The various alignment options are as follows:

Option	Meaning
<	Forces the expression within the curly braces to be left-aligned. This is the default for strings.
>	Forces the expression within the curly braces to be right-aligned. This is the default for numbers.
^	Forces the expression within the curly braces to be centered.

The following is an example using alignment for both a number and a string. The "|" is used in the f-string to help delineate the spacing. That number after the ":"will cause that field to be that number of characters wide. The first line in the **`print()`** statement using **`f`**-strings is an example of using f-strings for debugging purposes which will be covered later.

```
import math
variable = math.pi
```

```
print(f"Using Numeric {variable = }")
print(f"|{variable:25}|")
print(f"|{variable:<25}|")
print(f"|{variable:>25}|")
print(f"|{variable:^25}|\n")

variable = "Python 3.9"
print(f"Using String {variable = }")
print(f"|{variable:25}|")
print(f"|{variable:<25}|")
print(f"|{variable:>25}|")
print(f"|{variable:^25}|")
```

The output of this code snippet is:

```
Using Numeric variable = 3.141592653589793
|        3.141592653589793|
|3.141592653589793         |
|        3.141592653589793|
|    3.141592653589793     |

Using String variable = 'Python 3.9'
|Python 3.9                |
|Python 3.9                |
|                Python 3.9|
|       Python 3.9         |
```

A fill character can also be used in the alignment of f-strings. This is shown in the following example:

```
import math
variable = math.pi

print(f"Using String {variable = }")
print(f"|{variable:=<25}|")
print(f"|{variable:=>25}|")
print(f"|{variable:=^25}|\n")

variable = "Python 3.9"
print(f"Using String {variable = }")
print(f"|{variable:=<25}|")
print(f"|{variable:=>25}|")
print(f"|{variable:=^25}|")
```

The output of this code snippet is:

```
Using String variable = 3.141592653589793
|3.141592653589793========|
|========3.141592653589793|
|====3.141592653589793====|

Using String variable = 'Python 3.9'
|Python 3.9===============|
|===============Python 3.9|
|=======Python 3.9========|
```

Data Types

There are many ways to represent strings and numbers when using f-strings. The most common ones that you will need for this course are shown below:

Type	Meaning
s	String format—this is the default type for strings
d	Decimal Integer. This uses a comma as the number separator character.
n	Number. This is the same as **d** except that it uses the current locale setting to insert the appropriate number separator characters.
e	Exponent notation. Prints the number in scientific notation using the letter 'e' to indicate the exponent. The default precision is 6.
f	Fixed-point notation. Displays the number as a fixed-point number. The default precision is 6.
%	Percentage. Multiplies the number by 100 and displays in fixed (**'f'**) format, followed by a percent sign.

The following is a basic example of the use of f-strings with numbers:

```
import math
variable = 10

print(f"Using Numeric {variable = }")
print(f"This prints without formatting {variable}")
print(f"This prints with formatting {variable:d}")
print(f"This prints also with formatting {variable:n}")
print(f"This prints with spacing {variable:10d}\n")

variable = math.pi

print(f"Using Numeric {variable = }")
print(f"This prints without formatting {variable}")
```

```
print(f"This prints with formatting {variable:f}")
print(f"This prints with spacing {variable:20f}")
```

The output of this code snippet is the following:

```
Using Numeric variable = 10
This prints without formatting 10
This prints with formatting 10
This prints also with formatting 10
This prints with spacing          10

Using Numeric variable = 3.141592653589793
This prints without formatting 3.141592653589793
This prints with formatting 3.141593
This prints with spacing              3.141593
```

The variable, **variable**, is enclosed in curly braces **{ }**. When **variable = 10**, the f-string understands that **variable** is an integer and displays it as such. You can also use specify the type as **n** or **d** and use spacing in the output. When **variable = math.pi**, the f-string understands that **variable** is a floating-point number and displays it as such. You can also use specify the type as **f** and use spacing in the output.

The following are examples of the exponent notation and the percentage notation:

```
variable = 4

print(f"Using Numeric {variable = }")
print(f"This prints without formatting {variable}")
print(f"This prints with percent formatting {variable:%}\n")

variable = 403267890
print(f"Using Numeric {variable = }")
print(f"This prints with exponential formatting {variable:e}")
```

The output of this code snippet is the following:

```
Using Numeric variable = 4
This prints without formatting 4
This prints with percent formatting 400.000000%

Using Numeric variable = 403267890
This prints with exponential formatting 4.032679e+08
```

Formatting Floating Point Numbers

There are several options when formatting floating point numbers. The number of decimal places, the use of the number separator character, and forcing a plus (+) or minus (-) sign can also be specified. These are shown in the following example:

```
variable = 1200356.8796

print(f"Using Numeric {variable = }")
print(f"With two decimal places: {variable:.2f}")
print(f"With four decimal places: {variable:.3f}")
print(f"With two decimal places and a comma: {variable:,.2f}")
print(f"With a forced plus sign: {variable:+.2f}\n")

variable *= -1
print(f"Using Numeric {variable = }")
print(f"With two decimal places and a comma: {variable:,.2f}")
```

The output of this code snippet is the following:

```
Using Numeric variable = 1200356.8796
With two decimal places: 1200356.88
With three decimal places: 1200356.880
With two decimal places and a comma: 1,200,356.88
With a forced plus sign: +1200356.88

Using Numeric variable = -1200356.8796
With two decimal places and a comma: -1,200,356.88
```

Use of Tabs and Spacing

Program output is often required to be in tabular form. f-strings are very useful in formatting this kind of output. The following lines produce a tidily aligned set of columns with integers and their squares and cubes, using spaces to align the columns. Notice the use of the field width to ensure that all the numbers are right-aligned:

```
print(f'Number   Square       Cube')
for x in range(1, 11):
    print(f'{x:2d}       {x*x:3d}          {x*x*x:4d}')
```

The output of this code snippet is the following:

```
Number   Square       Cube
 1          1             1
 2          4             8
 3          9            27
 4         16            64
 5         25           125
 6         36           216
 7         49           343
 8         64           512
 9         81           729
10        100          1000
```

The tab character (**\t**) can also be used in an f-string to line up columns, particularly when column headings are used:

```
print(f'Number\t\tSquare\t\tCube')
for x in range(1, 11):
    print(f'{x:2d}\t\t\t{x*x:3d}\t\t\t{x*x*x:4d}')
```

The output of this code snippet is the following and very similar to that of the previous code snippet:

```
Number          Square          Cube
 1                1               1
 2                4               8
 3                9              27
 4               16              64
 5               25             125
 6               36             216
 7               49             343
 8               64             512
 9               81             729
10              100            1000
```

Using either tabs or spacing is acceptable and may depend on which one you are more comfortable with. Either way is acceptable in Python.

The following takes the previous program and converts **x** to a **float()** so that formatting of floating-point numbers can be demonstrated:

```
print(f'Number\t\tSquare\t\t\tCube')
for x in range(1, 11):
    x = float(x)
    print(f'{x:5.2f}\t\t{x*x:6.2f}\t\t{x*x*x:8.2f}')
```

The output of this code snippet is this:

```
Number          Square              Cube
 1.00            1.00              1.00
 2.00            4.00              8.00
 3.00            9.00             27.00
 4.00           16.00             64.00
 5.00           25.00            125.00
 6.00           36.00            216.00
 7.00           49.00            343.00
 8.00           64.00            512.00
 9.00           81.00            729.00
10.00          100.00           1000.00
```

This also demonstrates how the use of a value for width will enable the columns to line up.

The following program demonstrates the use of strings, decimals, and floats, as well as tabs for a type of report that is often produced in a typical Python program. Notice the use of the dollar sign (**$**) just before the variables that are to be displayed as prices.

```
APPLES = .50
BREAD = 1.50
CHEESE = 2.25

numApples = 3
numBread = 10
numCheese = 6

prcApples = numApples * APPLES
prcBread = numBread* BREAD
prcCheese = numCheese * CHEESE
strApples = 'Apples'
strBread = 'Rye Bread'
strCheese = 'Cheese'
total = prcBread + prcCheese + prcApples

print(f'{"My Grocery List":^30s}')
print(f'{"="*30}')
print(f'{strApples:10s}{numApples:10d}\t${prcApples:>5.2f}')
print(f'{strBread:10s}{numBread:10d}\t${prcBread:>5.2f}')
print(f'{strCheese:10s}{numCheese:10d}\t${prcCheese:>5.2f}')
print(f'{"Total:":>20s}\t${total:>5.2f}')
```

The output of this code snippet is:

```
       My Grocery List
==============================
Apples             3     $ 1.50
Rye Bread         10     $15.00
Cheese             6     $13.50
               Total:     $30.00
```

Using f-strings for Debugging

Updated in version 3.9 of Python is the use of f-strings for debugging purposes. The following shows how you can use f-strings to display the value of a variable in the form: **variable name=variable value**

```
import math
goldenRatio = (1+math.sqrt(5))/2
print(f"{goldenRatio=}")
```

This programming snippet calculates the mathematical and architectural pleasing ratio known since the time of Euclid. The f-string, **{goldenRatio =}**, will display the following:

```
goldenRatio=1.618033988749895
```

which facilitates the use of basic **print()** statements for debugging which can get complex and quickly show the variable name and its value. Any expression can be used as well as f-string format specifiers.

Spacing can also be preserved around the equal sign so that the following:

```
print(f"{goldenRatio = }")
print(f"{goldenRatio = :.6f}")
print(f"{(1+math.sqrt(5))/2 = :.6f}")
```

will be displayed as:

```
goldenRatio = 1.618033988749895
goldenRatio = 1.618034
(1+math.sqrt(5))/2 = 1.618034
```

Conclusion

Printing output in Python is facilitated with f-strings as it can be considered *What You See is What You Get* (WYSIWYG). The procedure is as follows:

- Placing between the quotation marks after the **'f'** the text that you want displayed
- Enclosing the variables to be displayed within the text in curly braces
- Within those curly braces, placing a colon (**:**) after the variable
- Formatting the variable using a format specification (width, alignment, data type) after the colon

Happy formatting!