Cheatography

Python F-Strings Basics Cheat Sheet

by Brian Allan (BrianAllan) via cheatography.com/133708/cs/29498/

What are F-Strings?

What are F-Strings?

> Strings prefixed with 'f' or 'F' and containing Python expressions inside curly braces for evaluation at run time. They are more formally known as "formatted string literals" and were introduced with Python 3.6.

How are F-Strings Useful?

> They provide a concise, readable way to include the value of a Python expression with formatting control inside strings.

What Are Other Ways to Format Strings in Python?

> - `str.format()`: the string `format()` method
> - %-formatting: old string formatting using the string modulo/percent operator %
> - `string.Template`: template class of the string module

F-String Template

```
f" text {replacement_field} text ... "
```

- Inside the quotes the f-string consists of two kinds of parts: (1) regular string literals, i.e., **text**, and (2) **replacement fields** containing Python expressions for evaluation along with formatting control.
- Double quotes are used in this representative pattern but single or triple quotes could also be used.
- F-strings may consist of just a replacement field:
  `f"{replacement_field}"`

Replacement Field

```
{f-expression = !conversion:format_specifier}
```

- A replacement field is signaled by a pair of curly braces: `{ }`
- A replacement field consists of an expression with optional debugging mode (`=`), type conversion (`!`), and format specification (`:`).
- Substituting into the f-string template:

f" *text* {f-expression = !conversion:format_specifier} *text* ... "

Conversion Field Options

`!s`	calls `str()` on the value of the f-expression
`!r`	calls `repr()` on the value of the f-expression
`!a`	calls `ascii()` on the value of the f-expression

- `str()` returns string value representations that are human readable for the end user.
- `repr()` returns string value representations for the interpreter and developers.
- `!s`, `!r`, `!a` conversions are redundant since arbitrary expressions are allowed in the replacement fields; so, one could just as easily replace `!r` by using `repr()` on the f-expression and similarly for the others.

Examples: Conversion Field

```
x, name = 'cat', 'Sophia'
```

explicitly using the string conversion `!s` (the default)

`f"The black {x!s}"`	'The black cat'

using the string conversion `!r` adds quotes

`f"The black {x!r}"`	"The black 'cat'"
`f"Her name is {name! r}."`	"Her name is 'Sophia'."

`!r` is equivalent to using `repr()` on the expression

`f"The black {repr( x)} "`	"The black 'cat'"

Format Specifier

```
:fill align sign # 0 width sep .precision type
```

Brief Summary of the Format Specification Mini-Language

`fill` : the padding character
`align` : alignment of text within the space
`sign` : how + and - are used preceding numbers
`#` : alternate presentation format for some number types
`0` : sign-aware, zero-padding on numbers
`width` : the minimum total field width
`sep` : the separator character for numbers (',' or '_')
`.precision` : determines how many digits displayed for floats; maximum field width for strings
`type` : the type of presentation to use based on data type

Note: `sign`, `#`, `0`, `sep`, `precision`, and `type` are of particular interest for **number formatting**. For information about number formatting, see my cheatsheet **Python F-Strings Number Formatting**.

By **Brian Allan** (BrianAllan)
cheatography.com/brianallan/

Published 16th March, 2022.
Last updated 24th February, 2023.

Cheatography

Python F-Strings Basics Cheat Sheet

by Brian Allan (BrianAllan) via cheatography.com/133708/cs/29498/

Format Specifier: Options

fill	align	sign	# 0	width	sep	.prec	type
char	<	+		digit(s)	_	digit(s)	*string*: s
	>	-			,		*number*: n
	^	''					*integer*: d, b, o, x, X, c
	=						*float*: e, E, f, F, g, G, %

String Presentation Type

simplified form: **`f" text {f-exp res sio n:type} text ... "`**

`s`	String format. This is the default type for strings and may be omitted.
None	Same as `s`

- Where the value of the f-expression is a string, the replacement field could make explicit the string presentation type **`{f-exp res sion:s}`** but **`:s`** can be omitted since this is the default for strings.
- `.precision` can be used with strings to enforce a maximum field width: `{f-exp res sio n:.p re cision}`

Examples: Simple F-Strings

```
x = 'cat'
```

f-expression can be a string

```
f"The black {'cat' }"                    'The black cat'
```

f-expression can be a variable with a string value

```
f"The black {x}"                         'The black cat'
```

error when neither a string nor a defined variable

```
f"The black {cat}"                       ☹
```

including f-string in string concatenation

```
'The ' 'black ' f"{x }"                  'The black cat'
```

including f-string in string concatenation

```
'The ' + 'black ' + f"{x }"              'The black cat'
```

f-strings can use single, double, or triple quotes

```
f'The ' f"black " f'''cat'''             'The black cat'
```

text inside the f-string must contain a different kind of quotes than the outer quotes

```
f'The 'black' cat'                       ☹
```

Examples: Simple F-Strings (cont)

f-expressions need different quotes than the outer quotes

```
f'The black {'cat'}'                     ☹
```

Or, f-expressions can use a variable to represent the string

```
f'The black {x}'                         'The black cat'
```

for debugging, an equal sign can be used after an f-expression to display the expression text and its value

```
f"The black {x=}"                        "The black x='cat'"
```

backslash escapes can be used in text

```
f"The black \'cat \'"                    "The black 'cat'"
```

doubled curly braces for a single curly brace in text

```
f"The {{black}} {x}"                     'The {black} cat'
```

using .precision to enforce a maximum field width of 7

```
f"{'The black cat':.7 }"                 'The bla'
```

Multi-line f-strings

```
f"""
The black                                '\nThe black\ncat'
cat" " "
```

Examples: Complex F-Expressions

```
colors = ['blue', 'green', 'yellow', 'red']
pets = {'cats': 2, 'dogs': 1}
```

f-expression with indexing and a method call

```
f"{c olo rs[ 2].t it le()} is my favorite color."
                                         'Yellow is my favorite color.'
```

f-expression with slicing

```
f"{c olo rs[ :3] }"
                                         "['blue', 'green', 'yellow']"
```

f-expression with a function call

```
f"There are {len(c olors)} option s."
                                         'There are 4 options.'
```

using dictionary keys and an arithmetical operation

```
f"She has {pets[ 'cats'] + pets[' dogs']} pets."
                                         'She has 3 pets.'
```

for debugging, an equal sign can be used to display the f-expression and its value

```
f"She has {pets[ 'cats'] + pets[' dogs'] = } pets.
"
                                         "She has pets['cats'] + pets['dogs'] = 3 pets."
```

using a conditional expression

Python F-Strings Basics Cheat Sheet

by Brian Allan (BrianAllan) via cheatography.com/133708/cs/29498/

Examples: Complex F-Expressions (cont)

```
f"She {'has' if (pets[ 'cats'] > pets[' dogs']) else 'does not have'} more cats than dogs.
"
```

```
# dictionary keys used by f-string must have different kind of quotes
f'She has {pet[' cat s']}.'
```

Formatting: Fill, Align, Width

f"{f -ex pre ssi on:fill align width} "

width	a decimal integer defining the minimum total field width. If not specified, the field width is determined by the content.
align	determines the alignment of text within the available space -- left-aligned (<), right-aligned (>), or centered (^)
fill	determines the character to use for padding to achieve the minimum total field width. The default is the space character.

Alignment Options

<	Left-alignment (the default for most objects)
>	Right-alignment (the default for numbers)
^	Centered
=	Sign-aware padding. For numeric types: Places the padding before the digits but after the sign (if any)

Examples: Fill, Align, Width

```
# fill (.), right align (>), width (12)
f"Go {'righ t':.>1 2}"                    'Go .....right'
# fill (!), left align (<), width (12)
f"Go {'left ':! <12 }"                    'Go left!!!!!!!!'
# fill (*), center align (^), width (12)
f"Go {'cent er' :*^ 12} "                 'Go ***center***'
# nested replacement fields allow for the use of variables in the format specifier
fill, align, width = '*', '^', 12
f"Go {'cent er' :{f ill }{a lig n}{ wid th} }"
                                          'Go ***center***'
# NOTE: the fill is the symbol (*), not the symbol as string ('*')
f"Go {'cent er' :'* '^1 2}"               ☹
```

Examples: Fill, Align, Width (cont)

```
# BUT when using a nested replacement field for fill, the value of the variable has to be the string of the symbol ('*'), not the symbol (*)
fill = '*'  'She has more cats than dogs.'
f"Go {'cent er' :{f ill }^1 2}"           'Go ***center***'
# Default fill when not specified is the space character
f"Go {'righ t': >10 }"                    ☹
f"Go {'righ t': >10 }"                    'Go      right'
# Default for strings when not specified: fill (space), left align (<)
f"Go {'left ':1 0}"                       'Go left      '
# Default for numbers when not specified: fill (space), right align (>)
f"Total: {5:8}"                           'Total:       5'
```

Example: For Loop and Nested Replacement Fields

```
width = 12
for text, fill in zip([' left', 'center', 'right'], '<^ >'):
        align = fill
        pri nt( f"{t ext :{f ill }{a lig n}{ wid - th} }")
```

left<<<<<<<<
^^^center^^^
>>>>>>>right

Example: Text Template Function

```
# function with f-string as message template
def messag e(name, num):
        return f"{n ame.ti tle ()}'s number is {num}."
messag e(' jenny', 8675309)
```

"Jenny's number is 8675309."

Cheatography

Python F-Strings Basics Cheat Sheet

by Brian Allan (BrianAllan) via cheatography.com/133708/cs/29498/

Example: Row Template for Table Creation

```
# data for table
presidents = [
        ['G eorge Washin gton', 1, 1789, 1797],
        ['John Adams', 2, 1797, 1801],
        ['T homas Jeffer son', 3, 1801, 1809]
]
# create row template function
def row(name, num, start, end):
        return f"| {name: <20} | {num:2} | {start}
- {end} |"
# print rows iterat ively
for p in presid ents:
        pri nt( row (p[0], p[1], p[2], p[3]))
```

```
| George Washington |  1 | 1789 - 1797 |
| John Adams        |  2 | 1797 - 1801 |
| Thomas Jefferson  |  3 | 1801 - 1809 |
```

Example: Title Bar

```
fill, align, width = '*', '^', 21
for text in ['', ' Title ', '']:
        pri nt( f"{t ext :{f ill }{a lig n}{ wid -
th} }")
```

```
*********************
******* Title *******
*********************
```

Datetime Formatting with F-Strings

Some Python objects have their own format specifiers to replace the standard ones. An example of this behavior is found in the `date`, `datetime`, and `time` objects of the `datetime` module.

using the `datetime` module to obtain today's date

```
import datetime
today = dateti me.d at e.t oday()
f"{today}"
```

'2022-03-14'

object-specific formatting directives used in place of the standard format specifiers

```
f"{today:%A, %B %d, %Y}"
```

'Monday, March 14, 2022'

the output is the same as using the `strftime()` method of the `datetime` module

```
today.strftime("%A, %B %d, %Y")
```

'Monday, March 14, 2022'

Short List of Datetime Formatting Directives

Directive	Meaning	Example
`%A`	Weekday full name	Sunday, Monday,...
`%a`	Weekday abbreviated	Sun, Mon,...
`%B`	Month full name	January, February,...
`%b`	Month abbreviated	Jan, Feb,...
`%d`	Day of Month	01, 02, 03,...
`%Y`	Year with Century	2019, 2020,...
`%y`	Year without Century	19, 20,...

References

- "A Guide to the Newer Python String Format Techniques" by John Sturtz at *Real Python*: https://realpython.com/python-formatted-output/#the-python-formatted-string-literal-f-string
- "Python 3's f-Strings: An Improved String Formatting Syntax (Guide)" by Joanna Jablonski at *Real Python*: https://realpython.com/python-f-strings/
- "Format String Syntax" including "Format Specification Mini-Language" from the page "`string` -- Common string operations": https://docs.python.org/3/library/string.html#format-string-syntax
- "2.4.3. Formatted string literals" from the page "2. Lexical analysis": https://docs.python.org/3/reference/lexical_analysis.html#formatted-string-literals
- "PEP 498 -- Literal String Interpolation": https://www.python.org/dev/peps/pep-0498/
- "Python String Format Cookbook": https://mkaz.blog/code/python-string-format-cookbook/

By **Brian Allan** (BrianAllan)
cheatography.com/brianallan/

Published 16th March, 2022.
Last updated 24th February, 2023.