3 Useful Python f-string Tricks You Probably Don't Know

Things you need to know about Python's formatted string literals (f-string)

Jerry Ng

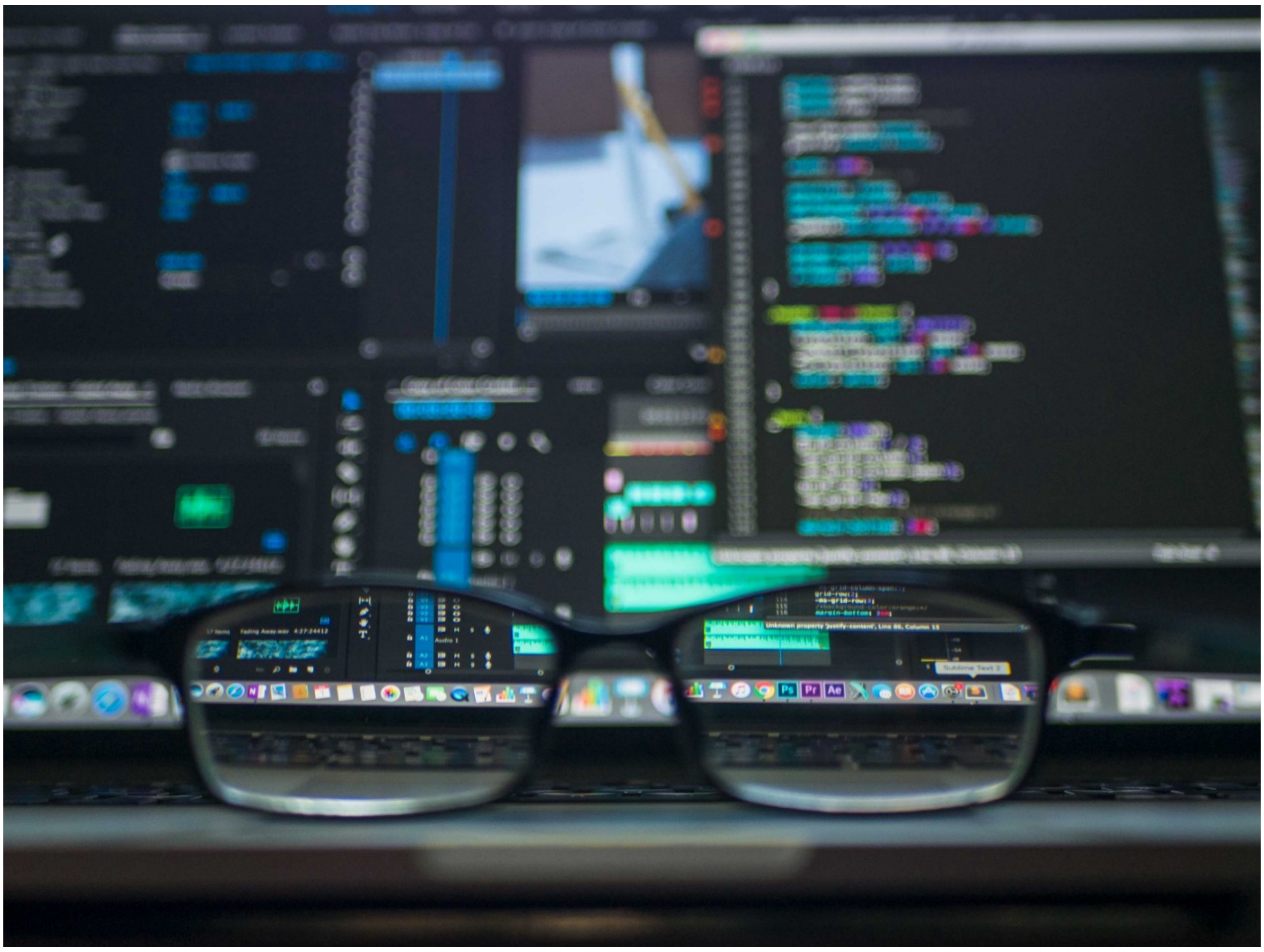
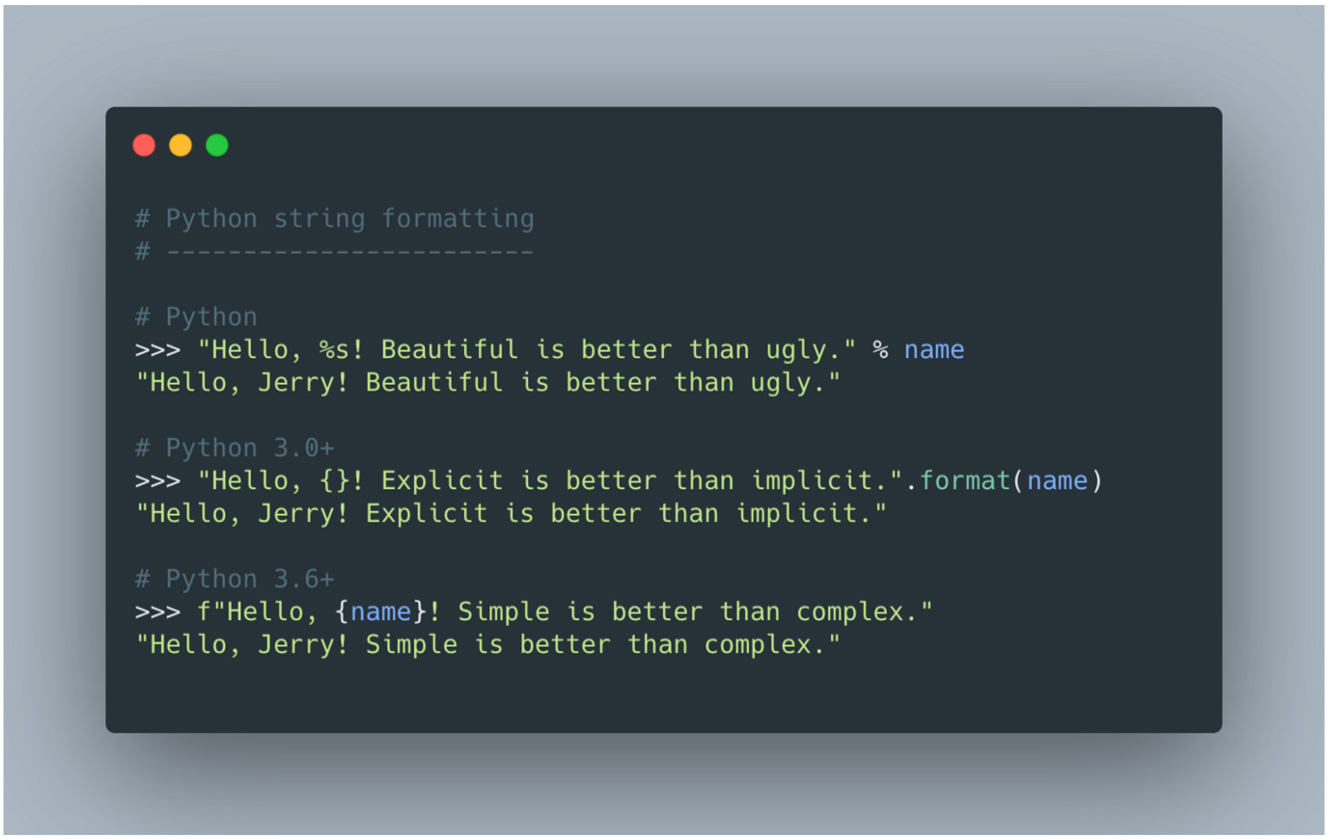


Photo by Kevin Ku on Unsplash

Long gone are the days when Python developers used the `%` operator to perform string formatting in Python.

Sometime later down the road when Python 3.0 was introduced, common use of the `%` the operator was gradually replaced with calling the `.format()` method on the String object.

Fast forward to Python 3.6 and beyond: f-string was introduced as a new way to embed Python expression inside our string constants. Today, many have already adopted this change.



```
# Python string formatting
# ------------------------

# Python
>>> "Hello, %s! Beautiful is better than ugly." % name
"Hello, Jerry! Beautiful is better than ugly."

# Python 3.0+
>>> "Hello, {}! Explicit is better than implicit.".format(name)
"Hello, Jerry! Explicit is better than implicit."

# Python 3.6+
>>> f"Hello, {name}! Simple is better than complex."
"Hello, Jerry! Simple is better than complex."
```

A brief overview of string formatting in Python. (Made using: https://carbon.now.sh/)

For those who are unfamiliar with this, using f-string in Python is super simple. Simply prefix your string with an `f` or `F` and then followed by single, double, or even triple quotes to create your string, e.g., `f"Hello, Python!"`.

While most of you might already be familiar with f-string formatting in Python, today I am going to demonstrate some of the lesser-known tricks of using f-string in

Python 3.8+.

Topics covered:

- Using f-string for better debugging
- Format `float` decimal places, currency, `datetime` objects, and padding with f-string
- Using f-string for conversions (ASCII and `repr()`)

Let's dive in, shall we?

1. F-string for Debugging

As developers, we often find ourselves using `print` statements to debug our code simply because they are easy to add.

Chances are, we'll do something like this:

A good way to lose track of which value belongs to which variable.

Here's the output printed on our terminal:

```
🐊
🐼
🐓
```

Honestly, there's nothing wrong with printing using the code above.

However, while staring at a terminal full of other debugger logs, one might easily forget the order of the variable printed out. In such a use case, debugging with `print` statements can quickly turn into finding a needle in a haystack. In our case:

```
🐊
🐼 -- Sorry, which variable does this belong to
🐓
```

A better way to print variables during debug

To cope with this, we can do `print(f’{variable_name=}’)` instead.

A clearer way to debug print using f-string!

Even though we have to wrap our variables with the f-string syntax, everything is so much clearer. Now, we will never lose track of the order of the printed variable during debugging.

Furthermore, using f-string this way preserves whitespaces, which can be helpful during our debugging process when we are in the midst of looking at confusing terminal logs.

f-string preserves whitespaces.

2. String Formatting

Besides using f-string for debugging, my absolute favorite feature about f-string is its flexibility, which allows us to format our string with ease.

With f-string, it's much easier to read, write, and maintain the output of the string with great flexibility as compared to the plain old string concatenation.

The simplest example can be demonstrated when we're dealing with `float` type variables.

How to format float to N decimal places

For instance, imagine we need to print a `float` type variable in two decimal places, e.g., making `3.1425123` into `3.14`. Not exactly an uncommon use case, right?

Typically, one might think of using the Python built-in `round` method to do so. Or perhaps, you're thinking of using the "older" `str.format` way of doing it, such as:

Ways of limiting float to two decimal places.

What if we could use f-string to do so instead? Let's try again!

Using f-string in this scenario is so much easier than `str.format()`

Turn float into a pretty currency value

When you're dealing with currency, you would need to prepare your strings to be as user-friendly as possible. For instance, it might be a better idea to format our currency value of `3142671.76` as `$3,142,671.76`.

In such a scenario, formatting our `float` value as currency using f-string is extremely handy. We can easily do this by using `,:2f`.

Format `float` type variable as a pretty-looking currency. Simple!

Pretty format `datetime` without `strftime`

Besides using it to format `float` type variables, we can even use f-string along with `datetime` type variable.

While formatting a `datetime` object into a more human-friendly manner, such as `05-July-2021`, the most common way of doing so is to use the standard built-in `strftime` method of the `datetime` object, e.g., `datetime_var.strftime('%d-%b-%y')`.

Alternatively, we could use f-string to easily format our `datetime` variable. Here's an example of what I meant:

Formatting `datetime` variable with f-string instead of `strftime`.

Padding your `int` variables with leading zeroes or whitespaces

Depending on what you're working on, there might be times where you will need to pad your integer in front while displaying or using it as a string.

Rather than having to write a custom function to zero-pad our integers, f-string allows you to zero-pad your integers without breaking a sweat. To pad with zeroes, simply do this:

Pad your integer typed variable with leading zeroes!

To pad your integers with leading whitespace:

Pad your integer typed variable with leading whitespaces!

Conversion

In the last section of this article, I will run you through some brief examples of using f-string for conversion. When you use an exclamation mark syntax, `!`, followed by your variable in your f-string, it will perform extra conversion on that variable.

Convert your string to ASCII representation

For instance, you can easily print out the ASCII representation of your string with `f'{your_variable!a}'`, just like this:

Shows the ASCII representation of your emoji.

A `repr()` alternative with f-string

Lastly, another good use of the `!` syntax in f-string is to display the string containing a printable representation of our object, just like using the `repr()` method in Python.

Here’s an example:

Using `!r` instead of `repr()`

Closing Thoughts

Since the Python 3.6 era, f-string has been an amazing way to format strings in Python. Using f-strings in Python is incredibly easy, and I have been using them ever since they were introduced.

Paired with the flexibility of letting us include expressions, conditions, and even formatting in our string object, f-string is undoubtedly simple yet powerful.

My personal favorite here is to use f-string for `datetime` object formatting as well as formatting any currency values.

Honestly, I am beyond thrilled to see how Python has evolved over these years, gradually making our developer experience better and better. I hope you learned a lot from this article and actually put these tips into use in your side projects or codebase at work.