PYTHON SNIPPETS

Formatting Numbers for Printing in Python

Phil Best
3 min read · Sep 2

Hey there, and welcome to another Python snippet post. This week we're taking a look at some formatting options for string representations of numbers in Python.

String formatting is actually a surprisingly large topic, and Python has its own internal mini language just for handling the many formatting options available to us. Here we're just going to focus on a few examples involving numbers, but there is a great deal more to explore.

Precision

First let's take a look at formatting a floating point number to a given level of precision. There are two ways we can do this: we can specify how many significant figures we want overall, or we can specify how many significant figures we want after the decimal point. Let's start with the former.

To specify a level of precision, we need to use a colon (`:`), followed by a decimal point, along with some integer representing the degree of precision. We place this inside the curly braces for an f-string, method instead, which I'll demonstrate below.

```
x = 4863.4343091          # example float to format

print(f"{x:.6}")          # f-string version
print("{:.6}".format(x))  # format method version
```

In both cases we get the same result: `4863.43`.

As we can see, our very long float got shortened down to just 6 figures. An interesting thing to note is that this formatting operation actually performs rounding, so if `x` had a value of `4863.435`, the number printed would actually be `4863.44`. The type of rounding is also very important, as this is one of the few instances where Python doesn't employ bankers' rounding (explained here).

If we specify fewer figures than we have in the integer portion of the float, we end up with an exponent representation instead:

```
x = 4863.4343091

print(f"{x:.3}")  # 4.86e+03
```

`4.86e+03` means `4.86 x 10³`, or `4.86 x 1000`, which is `4860`. Looking at this result, we see that we got three significant figures, as we requested.

So how do we specify 3 decimal places? We just need to add an `f`.

```
x = 4863.4343091
```

`f` indicates that we want our float displayed as a "fixed point number": in other words, we want a specific number of digits after the decimal point. We can use `f` on its own as well, which defaults to 6 digits of precision:

```
x = 4863.4343091

print(f"{x:f}")  # 4863.434309
```

Separators

For large numbers we often write a separator character (usually a comma, space, or period) to make it easier to read. We can specify this in Python using a comma or underscore character after the colon.

```
x = 1000000

print(f"{x:,}")  # 1,000,000
print(f"{x:_}")  # 1_000_000
```

This also works with floats, and the precision formatting, with the comma coming first:

```
x = 4863.4343091

print(f"{x:,.3f}")  # 4,863.434
print(f"{x:_.3f}")  # 4_863.434
```

Share this

Percentages

We can format a number as a percentage by simply adding the percent symbol at the end of our formatting options instead of `f`:

```
questions = 30
correct_answers = 23

print(f"You got {correct_answers / questions :.2%} correct!")
# You got 76.67% correct!
```

When formatting a number as a percentage, the level of precision always refers to the number of digits after the decimal point.

Wrapping Up

That's it for our very brief dive into formatting numbers. There's so much more to learn, and we'll be tackling this in future posts, so make sure to follow us on Twitter to keep up to date with all our content.

We're also offering our Complete Python Course for just $9.99 to readers of our blog, and if you click the link in this post, a coupon will already be applied for you. We'd love to have you, so if you're looking to upgrade your Python skills, or if you're just getting started, check it out!

Phil Best
I'm a freelance developer, mostly working in web development. I also create course content for Teclado!

Read More

Share this

Python Snippets

Assignment expressions in Python →

Python's namedtuples →

Python's divmod Function →

→ See all 26 posts

GUI DEVELOPMENT

Creating Snake Using Tkinter's Canvas Widget (Part 1)

In this post we start work on recreating the popular arcade game, Snake, using Tkinter's versatile Canvas widget.

Phil Best
11 min read · Sep 5

Share this

Like what you see? Enter your e-mail to hear when new posts come out!

E-mail*

Name*

Receive our newsletter and get notified about new posts we publish on the site, as well as occasional special offers.

Sign Up