Member-only story

15 Things I Regret Not Knowing Earlier About Python Formatted Strings

Liu Zuo Lin · Following
Published in Level Up Coding · 5 min read · Aug 21, 2024


```
name, age = 'tom', 5

print(f'Person({name=}, {age=})')
# ???

print(fr'''{3.14159:.4g}''')
# ???
```

1) {var=}

If we add a = sign after a variable, it prints both variable name and value.

```
name, age = 'tom', 1

print(f'{name=} {age=}')    # name='tom' age=1
```

Which is useful for debugging and logging purposes.

2) Rounding to N decimal places

To round a number to 2 decimal places, we can use {number:.2f}

To round a number to 3 decimal places, we can use {number:.3f}

```
pi = 3.14159265

print(f'{pi:.2f}')  # 3.14
print(f'{pi:.3f}')  # 3.142
print(f'{pi:.4f}')  # 3.1416
```

3) Rounding to N significant figures

To round a number to 2 significant figures, we can use {number:.2g}

To round a number to 3 significant figures, we can use {number:.3g}

```
x = 12345

print(f'{x:.2g}')  # 12000
print(f'{x:.3g}')  # 12300
print(f'{x:.4g}')  # 12340

y = 0.054321

print(f'{y:.2g}')  # 0.054
print(f'{y:.3g}')  # 0.0543
print(f'{y:.4g}')  # 0.05432
```

4) Formatted raw strings

A raw string is a string where \ is a literal backslash — a string where \ no longer escapes other characters.

To make a string a formatted string, we add a *f* in front of it.

To make a string a raw string, we add a *r* in front of it.

To make a string that is both a formatted string and a raw string, we add either a *fr* or *rf* in front of our string. Both works.

```
name = 'tom'
age = 1

print(fr'{name=}\n{age=}')  # name='tom'\nage=1
```

5) Alignment with whitespace

There are 3 ways to pad a variable with spaces to a width of 20 characters:

1. { var:<20 } aligns the variable var to the left
2. { var:>20 } aligns the variable var to the right
3. { var:^20 } aligns the variable var to the center

```
var = 'apple'

print(f'[{var:<20}]')   # [apple               ]
print(f'[{var:>20}]')   # [               apple]
print(f'[{var:^20}]')   # [       apple        ]
```

Note — this breaks if *var* has a length than is greater than 20

Quick Pause

I recently wrote a book — **101 Things I Never Knew About Python**

Check it out here if you wish to support me as a writer!

Link: https://payhip.com/b/vywcf

6) Alignment with other characters

We don't necessarily have to do the above with whitespace. We can specify other characters if we want to.

To do this, we simply need to add the desired character right after the colon.

```
var = 'apple'

print(f'[{var:-<20}]')   # [apple---------------]
print(f'[{var:=>20}]')   # [================apple]
print(f'[{var:@^20}]')   # [@@@@@@@apple@@@@@@@@]
```

7) Quotes within quotes in f-strings

This used to be illegal in an older version of Python, but has become legal now in recent versions — I'm using Python 3.12

If our variables inside our f-string require quotes, we can simply use them as they are.

- in the past: we'll get a syntax error
- now: this works perfectly fine

```
d = {'a': 100, 'b': 200}

print(f'{d['a']=}')     # d['a']=100
```

8) Inserting commas into large numbers

We can insert commas into very large numbers to make it easier to read. By default, every 3 digits will be grouped together.

We can do this by simply using { number:, }

```
n = 10000000000

print(f'{n:,}')     # 10,000,000,000
```

9) Displaying raw output

Adding a !r after a variable in an f-string allows us to display *raw output.*

Which is equivalent to displaying *repr(variable)*.

```
name = 'tom'

print(f'raw: {repr(name)}') # raw: 'tom'

print(f'raw: {name!r}')     # raw: 'tom'
```

Raw outputs are usually used for debugging and logging, and this shortbut might come in useful here in these use cases.

10) Formatting datetime

We can also format datetime objects directly in f-strings.

A simple example below:

- %Y represents the year in full eg. 2024
- %y represents the year but truncated eg. 24
- %m represents the month as a number
- %d represents the day as a number
- %A represents the day of the week eg. Monday, Tuesday

```
from datetime import datetime
```



```
print(f'{dt:%Y-%m-%d}')     # 2024-04-01

print(f'{dt:%y/%m/%d}')     # 24/04/01

print(f'{dt:%d/%m/%y %A}')  # 01/04/24 Monday
```

More datetime format codes can be found here:
https://docs.python.org/3/library/datetime.html#format-codes

11) Numbers to percentage

We can convert numbers directly to percentages in f-strings too.

In addition, we can specify how many decimal places there should be.

```
print(f'{0.25:.0%}')    # 25%
print(f'{0.5:.1%}')     # 50.0%
print(f'{0.75:.2%}')    # 75.00%
print(f'{1.5:.3%}')     # 150.000%
```

12) Binary, octal & hexadecimal numbers

We can represent numbers as binary, octal and hexadecimal numbers in f-strings.

- to represents numbers as binary (base 2), we add a *:b* after it
- to represents numbers as octal (base 8), we add a *:o* after it
- to represents numbers as hexadecimal (base 16), we add a *:x* after it

```
a, b, c = 2, 8, 32

print(f'binary={a:b} octal={a:o} hexadecimal={a:x}')
# binary=10 octal=2 hexadecimal=2

print(f'binary={b:b} octal={b:o} hexadecimal={b:x}')
# binary=1000 octal=10 hexadecimal=8

print(f'binary={c:b} octal={c:o} hexadecimal={c:x}')
# binary=100000 octal=40 hexadecimal=20
```

13) Triple quoted f-strings

Triple-quoted strings span multiple lines, and treat newlines as actual newline characters.

We can simply turn a triple-quoted string into a formatted triple quoted string by adding a *f* in front of it.

```
name, age = 'tom', 30

s = f'''
name: {name}
age: {age}
'''

print(s)
# name: tom
# age: 30
```

14) .format() with positional arguments

One limitation of f-strings is that we cannot use it as a template string that can be stored in a database.

To be able to store such a template string in a database for multiple uses, we might need to do something like this:

```
name = 'tom'
age =  5

s = 'my name is {} and my age is {}'

print(s.format(name, age))
# my name is tom and my age is 5
```

Note that the first argument *name* is assigned to the first empty bracket, while the second argument *age* is assigned to the second empty bracket.

15) .format() with keyword arguments

One limitation of using *.format()* with positional arguments is that our arguments must strictly follow a certain order.

To overcome this limitation, we can use keyword arguments instead. The catch is that we need to now add our variable names into our strings, like this:

```
name = 'tom'
age =  5

s = 'my name is {name} and my age is {age}'

print(s.format(name=name, age=age))
# my name is tom and my age is 5
```

We can switch the order of the keyword arguments and it'll still work

```
name = 'tom'
age =  5

s = 'my name is {name} and my age is {age}'

print(s.format(age=age, name=name))
# my name is tom and my age is 5
```

Conclusion

Hope you learnt at least a couple new things about Python formatted strings today.

Cheers

If You Wish To Support Me As A Creator

- Join my Substack newsletter at https://zlliu.substack.com/ — I send weekly emails relating to Python
- Buy my book — **101 Things I Never Knew About Python** at https://payhip.com/b/vywcf
- *Clap 50 times for this story*
- *Leave a comment telling me your thoughts*
- *Highlight your favourite part of the story*

Thank you! These tiny actions go a long way, and I really appreciate it!

YouTube: https://www.youtube.com/@zlliu246

LinkedIn: https://www.linkedin.com/in/zlliu/

Python Programming Coding Tech Technology

More from the list: "Reading list"

Curated by @reenum

Suhith Illesinghe
Agile is dead
· 6d ago

Allie Pasc... in UX Collecti...
"Winning" by design: Deceptive UX patterns...
6d ago

Oliver Bennet
Mastering Bash: Essential Commands for Everyda...
Oct 23

The I've

View list

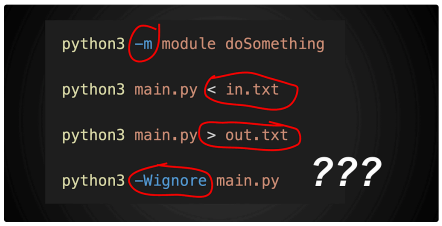
Written by Liu Zuo Lin

Following

96K Followers · Writer for Level Up Coding

My Substack Newsletter: https://zlliu.substack.com | 101 Things I Never Knew About Python: https://payhip.com/b/vywcf

More from Liu Zuo Lin and Level Up Coding



Liu Zuo Lin in Level Up Coding

Bryson Meiling in Level Up Coding

13 Python Command Line Things I Regret Not Knowing Earlier

Read free...

Oct 19 667 12

Stop making your python projects like it was 15 years ago...

I have a few things I've seen across companies and projects that I've seen...

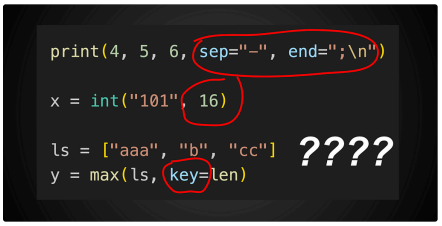
Sep 28 4.3K 45

Hayk Simonyan in Level Up Coding

STOP using Docker Desktop: Faster Alternative Nobody Uses

Ditch Docker Desktop and try this faster, lighter tool that will make your life easier!

Oct 8 2.3K 41



Liu Zuo Lin in Level Up Coding

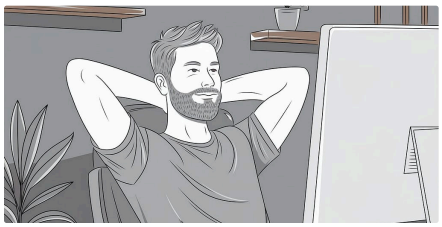
12 Python Built-in Function Things I Regret Not Knowing Earlier

Friend Link Here

Oct 12 873 5

See all from Liu Zuo Lin | See all from Level Up Coding

Recommended from Medium

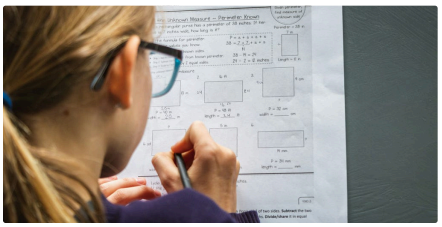


Abdur Rahman in Stackademic

20 Python Scripts To Automate Your Daily Tasks

A must-have collection for every developer

Oct 6 1.2K 9



Egor Howell in Towards Data Science

4 Years of Data Science in 8 Minutes

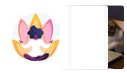
What I have learned in my 4+ year journey of studying data science

5d ago 825 13

Lists

Coding & Development
11 stories · 878 saves

General Coding Knowledge
20 stories · 1688 saves



Stories to Help You Grow as a Software Developer
19 stories · 1447 saves

ChatGPT prompts
50 stories · 2157 saves

Bryson Meiling in Level Up Coding

Stop making your python projects like it was 15 years ago...

I have a few things I've seen across companies and projects that I've seen...

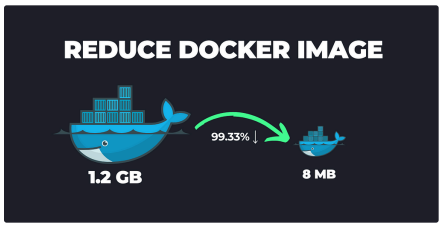
Sep 28 · 4.3K · 45

Paolo Molignini, PhD in Puzzle Sphere

Can you solve this famous interview question?

100 passengers, 100 seats—but the first one sits randomly! What's the chance the last...

Oct 5 · 1.3K · 31



Dipanshu in AWS in Plain English

Docker pros are shrinking images by 99%: The hidden techniques yo...

Unlock the secrets to lightning-fast deployments and slashed costs—before yo...

Sep 18 · 2.4K · 11

Nidhi Jain in Code Like A Girl

7 Productivity Hacks I Stole From a Principal Software Engineer

Golden tips and tricks that can make you unstoppable

Oct 15 · 3.2K · 54

See more recommendations