Beginner Python drops

Rodrigo Girão Serrão

Contents

1 – `zip`'s keyword argument `strict`

The Python built-in `zip` has a keyword argument `strict` that will raise an error if the 2 (or more) iterables that you pass to `zip` don't have the same length.

Use this whenever you are passing arguments that should have the same length: it helps catch errors early.

Beware that `zip` only raises the error when it reaches the end of the shortest iterable. In other words, it doesn't validate the lengths upfront.

That's why you are able to print the first two names, and only then `zip` raises a `ValueError` when the list `lasts` ends:

```
firsts = ["Luke", "Darth", "Obi-Wan"]
lasts = ["Skywalker", "Vader"]
for first_name, last_name in zip(firsts, lasts, strict=True):
    print(f"{first_name} {last_name}")
```

```
Luke Skywalker
Darth Vader
Traceback (most recent call last):
  File "<python-input-0>", line 3, in <module>
    for first_name, last_name in zip(firsts, lasts, strict=True):
                                 ~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: zip() argument 2 is shorter than argument 1
```

Further reading:

- Article about `zip`

13 – String prefixes and suffixes

Strings have four convenience methods to replace some slicing: `startswith`, `endswith`, `removeprefix`, and `removesuffix`.

These methods are preferred over the slicing alternatives because they are more convenient and more readable. (The methods `removeX` require Python 3.9+.)

Here are two examples operating on the start of a string:

```
string = "Hello, world!"
print(string.startswith("Hello"))
# True
print(string.removeprefix("Hello"))
# , world!
```

The methods `startswith` and `endswith` also accept a tuple of strings to check:

```
string = "abracadabra"
possible_prefixes = ("aa", "ab", "ac")
print(string.startswith(possible_prefixes))
# True
```

19 – Self-debugging f-strings

f-strings have an awesome feature: if you include an equals sign = at the end of the formatted value, the f-string will show you the code and the value that you're formatting.

Here is an example:

```
name = "RoDrIgO"
print(f"Method title: {name.title() = }")
# Method title: name.title() = 'Rodrigo'
```

Note that the spaces around the equals sign = are not necessary but the result usually looks better if you include them.

27 – Enforce keyword arguments for options

You can use a single asterisk * in a function definition to force all following arguments to be keyword-only.

This is particularly helpful for arguments that act as options or as configuration values. Here is an example with a function that can return the temperature in a room in two units, Celsius and Fahrenheit:

```
def get_temperature(room, *, unit)
```

By using *, the second argument must be passed as a keyword argument:

```
get_temperature("bedroom", unit="celsius")  # This works.
```

If you don't, you get an exception `TypeError`:

```
get_temperature("bedroom", "celsius")  # TypeError
```

44 – Format specifier `!r`

When you're using f-strings, you can use the format specifier `!r` to use a value's debugging representation instead of pretty-printing it.

Some values and types cannot be distinguished from one another if you pretty-print them, but can if you use their debugging representation. For example, if a string represents an integer, you can't distinguish it from the same integer when printing:

```
s = "3"
print(f"{s}")  # 3
# !? Was `s` the string "3" or the integer 3?
```

Using `!r` makes it clearer what's being printed:

```
print(f"{s!r}")  # '3'
```

Here's another example:

```
from fractions import Fraction

one_third = Fraction(1, 3)
print(f"{one_third}, {one_third!r}")
# 1/3, Fraction(1, 3)
```

Further reading:

- `str` and `repr`