PYTHON SNIPPETS

Using zip in Python

Phil Best
2 min read · Jul 15

Python's `zip` function is an underused and extremely powerful tool, particularly for working with multiple collections inside loops. In this snippet post, we're going to show off a couple of cool ways you can use `zip` to improve your Python code in a big way.

What is `zip`

`zip` is a function allows us to combine two or more iterables into a single iterable object. As the name would suggest, it interlaces values from the different iterables, creating a collection of tuples.

For example, the lists `[1, 2, 3]` and `["a", "b", "c"]` would yield a `zip` object containing `(1, "a")`, `(2, "b")`, and `(3, "c")`.

When to use `zip`

One thing we see a lot with our newer students over at the Complete Python Course is that when they're working with two separate collections in a loop, they tend to do something like this:

```python
names = ["John", "Anne", "Peter"]
ages = [26, 31, 29]

for i in range(len(names)):
    print(f"{names[i]} is {ages[i]} years old.")
```

I can certainly understand why somebody would write something like this. By generating a collection of indices, we can use the stable ordering of lists to access the right value from each collection. We could use this technique for combining as many collections as we liked.

This kind of situation, however, is a prime candidate for `zip`. Using `zip`, we can combine the `names` and `ages` into a shiny new `zip` object. We can then iterate over the `zip` object, and we can also make use of some destructuring, allowing us to use nice descriptive names for our loop variables.

```python
names = ["John", "Anne", "Peter"]
ages = [26, 31, 29]

for name, age in zip(names, ages):
    print(f"{name} is {age} years old.")
```

Using `zip` in reverse

As if that wasn't enough, `zip` can actually do even more. It really does just keep on giving.

Using the `*` operator, we can break up a `zip` object, or really any collection of collections. For example, we might have something like this:

```
zipped = [("John", 26), ("Anne", 31), ("Peter", 29)]
```

We can use the `*` operator in conjunction with `zip` to split this back into `names` and `ages`:

```
zipped = [("John", 26), ("Anne", 31), ("Peter", 29)]
names, ages = zip(*zipped)

print(names)  # ("John", "Anne", "Peter")
print(ages)   # (26, 31, 29)
```

Wrapping up

`zip` is a really fantastic tool for getting rid of things like indices in your loops, and it can also be used to break apart nested collections into specific fields.

If you're interested in learning more cool things like this, follow us on Twitter, and consider taking our Complete Python Course. We recently updated it with all new sections, so it's better value than ever!

We also have a form down below for signing up to our mailing list where we post regular discount codes for all our courses, ensuring you get the best deal possible.

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

Read More

Python Snippets

Assignment expressions in Python →

Python's namedtuples →

Python's divmod Function →

→ See all 26 posts

CODING INTERVIEW PROBLEMS

Coding Interview Problems: Palindromes

We're back tackling another common coding interview problem. This time we're tackling palindromes! Avoid common problems and impress at your next interview.

Phil Best

8 min read · Jul 18

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