PYTHON SNIPPETS

Python's namedtuples

Phil Best

3 min read · Oct 14

Hey there, and welcome to this short post on namedtuples! namedtuples are part of the wonderful collections module, and they let us use names to access tuple elements, rather than indices, making our code far more readable. I personally think they're absolutely amazing, so let's see them in action!

Defining a `namedtuple`

In order to use a `namedtuple`, we need to define a blueprint for it. For example, if we want a `namedtuple` called `Student`, where the elements are the student's name, their age, and their primary faculty, it might look something like this:

```
from collections import namedtuple

Student = namedtuple("Student", ["name", "age", "faculty"])
```

Let's break this down

Let's break this down:

First it's important to note that we're doing an assignment here, and the naming convention is to use an uppercase letter for the variable name, just like if we were defining a class.

On the right hand side of the assignment, we have `namedtuple` along with a set of parentheses, and inside, we have all of our configuration for the `Student` `namedtuple`.

The first item inside the brackets is the name of the variable we're assigning to, and the second is a list of keys for the elements in `Student`.

Creating instances of a `namedtuple`

Now that we have our blueprint defined, we can make use of `Student` in our code. For example, we might do something like this:

```
from collections import namedtuple

Student = namedtuple("Student", ["name", "age", "faculty"])

names = ["John", "Steve", "Mary"]
ages = [19, 20, 18]
faculties = ["Politics", "Economics", "Engineering"]

students = [
    Student(*student_details)
    for student_details in zip(names, ages, faculties)
]
```

Here we've taken some connected data from three different lists, and we've use a list comprehension in conjunction with zip to create a

we've use a list comprehension in conjunction with zip to create a single list of `Student` objects.

In the example above, we use the `*` syntax to destructure the each tuple in the `zip` object, but we have some other options. For example, we can use keyword arguments to define values for a `namedtuple`:

```
example_student = Student(name="James", age=18, faculty="Music")
```

Accessing elements within a `namedtuple`

Returning to our earlier example with the list of students, we might want to process this list in some way to determine something about our data. For example, we might want to use `max` to find the oldest student in the list.

We could do that like so:

```
oldest_student = max(students, key=lambda student: student.age)
```

Here we can see our first example of accessing values inside a named tuple. We need to use this `.` syntax to access values by key, putting the name of the key after the dot.

In the code above, we've set a custom key for `max` so that it compares our tuples based on the `age` element in each tuple. You can learn more about this in the documentation.

Using a regular tuple, or a list, this might have looked more like this:

Using a regular tuple, or a list, this might have looked more like this:

```
oldest_student = max(students, key=lambda student: student[1])
```

Now that we've had to use an index, the meaning of the lambda function is a lot less clear. We have to refer back to the original data to see what exactly is at index `1`.

If for some reason you want to use an index with a `namedtuple`, that still works, so you're free to do so.

Just to finish off our example above, let's print the `oldest_student`:

```
Student(name='Steve', age=20, faculty='Economics')
```

The output is actually very readable, but to make it look a little less like code for our users, we can use an f-string:

```
print(f"{oldest_student.name} ({oldest_student.age}) - {oldest_student.faculty}")

# Steve (20) - Economics
```

As you can see, namedtuples make it really clear which values we're accessing, making our code very readable. Everyone should be using them more often!

Wrapping up

That's it for this short post on namedtuples. I think namedtuples are awesome, and hopefully now you do too. I'd definitely recommend adding them to the types you use regularly, as they have great potential for improving code readability.

If you liked this post, you might want to check out our Complete Python Course, or join us over on Discord! We're on hand to help you take your Python to the next level!

Make sure you also sign up to the mailing list down below, as we post discount codes for our subscribers, ensuring you always get the best deals on our courses.

Happy coding!

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 divmod Function →

Python's format Function →

→ See all 26 posts

LEARN PYTHON PROGRAMMING

Writing Clean Python

In this post we talk about some tips and techniques for writing clean, readable Python. Avoid common style mistakes and write beautiful Python code!

Phil Best
12 min read · Oct 17

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