Python NamedTuple

Summary: in this tutorial, you'll learn how to use the Python `namedtuple` function to create named tuples.

Introduction to Python named tuples

The following shows a tuple (https://www.pythontutorial.net/python-basics/python-tuples/) that has two elements:

```
point = (100,200)
```

The `point` tuple represents a 2D point whose x-coordinate is `100` and y coordinate is `200` .

To get the x-coordinate and y-coordinate, you can use the following syntax:

```
x = point[0]
y = point[1]
```

This code works fine. However, it's not so obvious.

When you look at the `point[0]`, you need to know its implicitly meaning which doesn't mention in the code.

To make the code more clear, you might want to use a class (https://www.pythontutorial.net/python-oop/python-class/). For example:

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

And you can create a new instance of the `Point2D`:

```
a = Point2D(100, 200)
```

Also, you can access x-coordinate and y-coordinate attributes:

```
print(a.x)
print(a.y)
```

To compare if two points are the same, you need to implement the `__eq__` method:

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        if isinstance(other, Point2D):
            return self.x == other.x and self.y == other.y
        return False
```

The `__eq__` method checks if a point is an instance of the `Point2D` class and returns `True` if both x-coordinate and y-coordinate are equal.

The following shows how to compare two instances of the `Point2D` class:

```
a = Point2D(100, 200)
b = Point2D(100, 200)

print(a is b)  # False
print(a == b)  # true
```

The `Point2D` might work as you expected. However, you need to write a lot of code.

To combine the simplicity of a tuple and the obvious of a class, you can use a named tuple:

Named tuples allow you to create tuples and assign meaningful names to the positions of the tuple's elements.

Technically, a named tuple is a subclass of `tuple`. On top of that, it adds property names to the positional elements.

Creating named tuple classes

To create a named tuple class, you need to the `namedtuple` function of the `collections` standard library.

The `namedtuple` is a function that returns a new named tuple class. In other words, the `namedtuple()` is a class factory.

To use the `namedtuple` function, you need to import it from the `collections` module first:

```
from collections import namedtuple
```

The `namedtuple` function accepts the following arguments to generate a class:

- A class name that specifies the name of the named tuple class.
- A sequence of field names that correspond to the elements of tuples. The field names must be valid variable names except that they cannot start with an underscore (`_`).

For example, the following uses the `namedtuple` function to create the `Point2D` class:

```
Point2D = namedtuple('Point2D',['x','y'])
```

The namedtuple also can accept fields names as:

1) A tuple of string:

```
Point2D = namedtuple('Point2D',('x','y'))
```

2) Or a single string with field names separated by commas:

```
Point2D = namedtuple('Point2D',('x, y'))
```

3) Or a single string with field names separated by whitespaces

```
Point2D = namedtuple('Point2D','x y')
```

Instantiating named tuples

The `Point2D` is a class, which is a subclass (https://www.pythontutorial.net/python-oop/python-inheritance/) of the `tuple` . And you can create new instances of the `Point2D` class like you do with a regular class. For example:

```
point = Point2D(100, 200)
```

Or you can use the keyword arguments:

```
point = Point2D(x=100, y=200)
```

The `point` object is an instance of the `Point2D` class. Therefore, it's an instance of the `tuple` class:

```
print(isinstance(point, Point2D))  # True
print(isinstance(point, tuple))  # True
```

Accessing data of a named tuple

A named tuple is a regular tuple. Therefore, you can apply all tuple operations on a named tuple.

To access data in a named tuple, you can use:

- Slicing
- Unpacking
- Indexing
- and iterating

For example:

```
# unpacking
x, y = point
print(f'({x}, {y})')  # (100, 200)

# indexing
x = point[0]
y = point[1]
print(f'({x}, {y})')  # (100, 200)
```

```
# iterating

for coordinate in point:
    print(coordinate)
```

Output:

```
(100, 200)
(100, 200)
100
200
```

The rename argument of the namedtuple function

The `namedtuple` function accepts the `rename` the keyword-only argument that allows you to rename invalid field names.

The following results in an error because the field name `_radius` starts with an underscore (`_`):

```
from collections import namedtuple

Circle = namedtuple(
    'Circle',
    'center_x, center_y, _radius'
)
```

However, when you use the `rename` argument, the `namedtuple` function automatically renames the `_radius` to a valid field name. For example:

```
from collections import namedtuple

Circle = namedtuple(
```

```
    'Circle',
    'center_x, center_y, _radius',
    rename=True
)
```

To find field names of a named tuple, you can use the `_fields` class property. For example:

```
print(Circle._fields)
```

Output:

```
('center_x', 'center_y', '_2')
```

In this example, the `namedtuple` function changes the `_radius` field to `_2` automatically.

Additional Python functions of named tuples

Named tuples provide some useful functions out of the box. For example, you can use the equal operator (`==`) to compare two named tuple instances:

```
a = Point2D(100, 200)
b = Point2D(100, 200)

print(a == b)  # True
```

If you use the class, you need to implement the `__eq__` to get this function.

Also, you can get the string representation of a named tuple:

```
print(a)
```

Output:

```
Point2D(x=100, y=200)
```

Again, if you use the class, you need to implement `__rep__` method.

Since a named tuple is a tuple, you can apply any function that is relevant to a regular tuple to a named tuple. For example:

```
print(max(a))  # 200
print(min(a))  # 100
```

Summary

- Named tuples are tuples whose element positions have meaningful names.
- Use the `namedtuple` function of the `collections` standard library to create a named tuple class.
- Named tuples are immutable.