# namedtuple

The one that we'll be focusing on in this section is the **namedtuple** which you can use to replace Python's **tuple**. Of course, the namedtuple is not a drop-in replacement as you will soon see. I have seen some programmers use it like a struct. If you haven't used a language with a struct in it, then that needs a little explanation. A struct is basically a complex data type that groups a list of variables under one name. Let's look at an example of how to create a namedtuple so you can see how they work:

```
from collections import namedtuple

Parts = namedtuple('Parts', 'id_num desc cost amount')
auto_parts = Parts(id_num='1234', desc='Ford Engine',
                   cost=1200.00, amount=10)
print(auto_parts.id_num)
```

Here we import **namedtuple** from the **collections** module. Then we called namedtuple, which will return a new subclass of a tuple but with named fields. So basically we just created a new tuple class. you will note that we have a strange string as our second argument. This is a space delimited list of properties that we want to create.

Now that we have our shiny new class, let's create an instance of it! As you can see above, we do that as our very next step when we create the **auto_parts** object. Now we can access the various items in our auto_parts using dot notation because they are now properties of our Parts class.

One of the benefits of using a namedtuple over a regular tuple is that you no longer have to keep track of each item's index because now each item is named and accessed via a class property. Here's the difference in code:

```
auto_parts = ('1234', 'Ford Engine', 1200.00, 10)
print (auto_parts[2] ) # access the cost

#1200.0

id_num, desc, cost, amount = auto_parts
print (id_num )
#'1234'
```

In the code above, we create a regular tuple and access the cost of the vehicle engine by telling Python the appropriate index we want. Alternatively, we can also extract everything from the tuple using multiple assignment. Personally, I prefer the namedtuple approach just because it fits the mind easier and you can use Python's **dir()** method to inspect the tuple and find out its properties. Give that a try and see what happens!

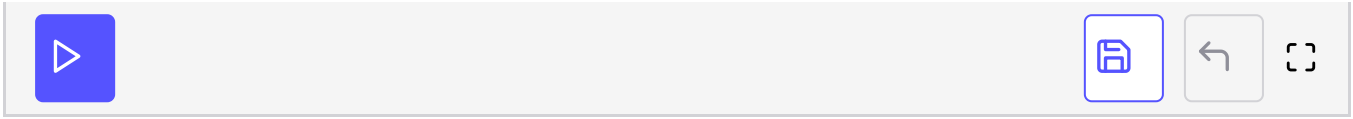
The other day I was looking for a way to convert a Python dictionary into an object and I came across some code that did something like this:

```
from collections import namedtuple

Parts = {'id_num':'1234', 'desc':'Ford Engine',
     'cost':1200.00, 'amount':10}
parts = namedtuple('Parts', Parts.keys())(**Parts)
print (parts)
#Parts(amount=10, cost=1200.0, id_num='1234', desc='Ford Engine')
```

This is some weird code, so let's take it a piece at a time. The first line we import namedtuple as before. Next we create a Parts dictionary. So far, so good. Now we're ready for the weird part. Here we create our namedtuple class and name it 'Parts'. The second argument is a list of the keys from our dictionary. The last piece is this strange piece of code: **(**Parts)**. The double asterisk means that we are calling our class using keyword arguments, which in this case is our dictionary. We could split this line into two parts to make it a little clearer:

```
parts = namedtuple('Parts', Parts.keys())
print (parts)
#<class '__main__.Parts'>
```

```
auto_parts = parts(**Parts)


print (auto_parts)
#Parts(amount=10, cost=1200.0, id_num='1234', desc='Ford Engine')
```



So here we do the same thing as before, except that we create the class first, then we call the class with our dictionary to create an object. The only other piece I want to mention is that namedtuple also accepts a **verbose** argument and a **rename** argument. The verbose argument is a flag that will print out class definition right before it's built if you set it to True. The rename argument is useful if you're creating your namedtuple from a database or some other system that your program doesn't control as it will automatically rename the properties for you.

At this point you should be familiar enough with the namedtuple to use it yourself, so let's check out the OrderedDict!