13.5. Unpacking Tuples as Arguments to Function Calls

Python even provides a way to pass a single tuple to a function and have it be unpacked for assignment to the named parameters.

Save & Run Original - 1 of 1 Show in CodeLens

```
1 def add(x, y):
2     return x + y
3
4 print(add(3, 4))
5 z = (5, 4)
6 print(add(z)) # this line causes an error
7
```

```
7
```

Activity: 1 -- ActiveCode (ac12_4_6)

Error

```
TypeError: add() takes exactly 2 arguments (1 given) on line 6
```

Description

Type errors most often occur when an expression tries to combine two objects with types that should not be combined. Like raising a string to a power

To Fix

To fix a type error you will most likely need to trace through your code and make sure the variables have the types you expect them to have. It may be helpful to print out each variable along the way to be sure its value is what you think it should be.

This won't quite work. It will cause an error, because the function add is expecting two parameters, but you're only passing one parameter (a tuple). If only there was a way to tell python to unpack that tuple and use the first element to assign to x and the second to y.

Actually, there is a way.

Save & Run Original - 1 of 1 Show in CodeLens

```
1 def add(x, y):
2     return x + y
3
4 print(add(3, 4))
5 z = (5, 4)
6 print(add(*z)) # this line will cause the values to be unpacked
7
```

```
7
9
```

Activity: 2 -- ActiveCode (ac12_4_6b)

Don't worry about mastering this idea yet. But later in the course, if you come across some code that someone else has written that uses the * notation inside a parameter list, come back and look at this again.

You have attempted 3 of 3 activities on this page

13.4. Tuples as Return Values">
ples as Return Values">
13.6. Glossary">
✔ Completed. Well Done!
13.6. Glossary">Next Section - 13.6. Glossary

| Back to top