# 13.3. Tuple Assignment with Unpacking

Python has a very powerful **tuple assignment** feature that allows a tuple of variable names on the left of an assignment statement to be assigned values from a tuple on the right of the assignment. Another way to think of this is that the tuple of values is **unpacked** into the variable names.

Save & Run Original - 1 of 1 Show in CodeLens

```
julia = "Julia", "Roberts", 1967, "Duplicity", 2009, "Actress", "Atlanta, Georgia"

name, surname, birth_year, movie, movie_year, profession, birth_place = julia

```

Activity: 1 -- ActiveCode (ac12_4_1)

This does the equivalent of seven assignment statements, all on one easy line.

Naturally, the number of variables on the left and the number of values on the right have to be the same.

Save & Run Original - 1 of 1 Show in CodeLens

```
(a, b, c, d) = (1, 2, 3) # ValueError: need more than 3 values to unpack

```

Activity: 2 -- ActiveCode (ac12_4_4)

## Error

```
ValueError: need more than 3 values to unpack on line 1
```

## Description

A ValueError most often occurs when you pass a parameter to a function and the function is expecting one type and you pass another.

## To Fix

The error message gives you a pretty good hint about the name of the function as well as the value that is incorrect. Look at the error message closely and then trace back to the variable containing the problematic value.

**Note**

Unpacking into multiple variable names also works with lists, or any other sequence type, as long as there is exactly one value for each variable. For example, you can write `x, y = [3, 4]`.

## 13.3.1. Swapping Values between Variables

This feature is used to enable swapping the values of two variables. With conventional assignment statements, we have to use a temporary variable. For example, to swap `a` and `b`:

Save & Run Original - 1 of 1 Show in CodeLens

```
a = 1
b = 2
temp = a
a = b
b = temp
print(a, b, temp)

```

```
2 1 1
```

Activity: 3 -- ActiveCode (ac12_4_2)

Tuple assignment solves this problem neatly:

Save & Run Original - 1 of 1 Show in CodeLens

```
a = 1
b = 2
(a, b) = (b, a)
print(a, b)

```

```
2 1
```

Activity: 4 -- ActiveCode (ac12_4_3)

The left side is a tuple of variables; the right side is a tuple of values. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.

## 13.3.2. Unpacking Into Iterator Variables

Multiple assignment with unpacking is particularly useful when you iterate through a list of tuples. You can unpack each tuple into several loop variables. For example:

Save & Run Original - 1 of 1 Show in CodeLens

```
authors = [('Paul', 'Resnick'), ('Brad', 'Miller'), ('Lauren', 'Murphy')]
for first_name, last_name in authors:
    print("first name:", first_name, "last name:", last_name)

```

```
first name: Paul last name: Resnick
first name: Brad last name: Miller
first name: Lauren last name: Murphy
```

Activity: 5 -- ActiveCode (ac12_4_8a)

On the first iteration the tuple `('Paul', 'Resnick')` is unpacked into the two variables `first_name` and `last_name`. One the second iteration, the next tuple is unpacked into those same loop variables.

## 13.3.3. The Pythonic Way to Enumerate Items in a Sequence

When we first introduced the for loop, we provided an example of how to iterate through the indexes of a sequence, and thus enumerate the items and their positions in the sequence.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
fruits = ['apple', 'pear', 'apricot', 'cherry', 'peach']
for n in range(len(fruits)):
    print(n, fruits[n])

```

```
0 apple
1 pear
2 apricot
3 cherry
4 peach
```

Activity: 6 -- ActiveCode (ac12_4_8b)

We are now prepared to understand a more pythonic approach to enumerating items in a sequence. Python provides a built-in function `enumerate`. It takes a sequence as input and returns a sequence of tuples. In each tuple, the first element is an integer and the second is an item from the original sequence. (It actually produces an "iterable" rather than a list, but we can use it in a for loop as the sequence to iterate over.)

Save & Run | Original - 1 of 1 | Show in CodeLens

```
fruits = ['apple', 'pear', 'apricot', 'cherry', 'peach']
for item in enumerate(fruits):
    print(item[0], item[1])

```

```
0 apple
1 pear
2 apricot
3 cherry
4 peach
```

Activity: 7 -- ActiveCode (ac12_4_8c)

The pythonic way to consume the results of enumerate, however, is to unpack the tuples while iterating through them, so that the code is easier to understand.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
fruits = ['apple', 'pear', 'apricot', 'cherry', 'peach']
for idx, fruit in enumerate(fruits):
    print(idx, fruit)

```

```
0 apple
1 pear
2 apricot
3 cherry
4 peach
```

Activity: 8 -- ActiveCode (ac12_4_8d)

**Check your Understanding**

tuples-4-1: Consider the following alternative way to swap the values of variables x and y. What's wrong with it?

```
# assume x and y already have values assigned to them
y = x
x = y
```

- A. You can't use different variable names on the left and right side of an assignment statement.
- B. At the end, x still has it's original value instead of y's original value.
- C. Actually, it works just fine!

Check me | Compare me

✔ Once you assign x's value to y, y's original value is gone.

Activity: 9 -- Multiple Choice (question12_4_2)

With only one line of code, assign the variables `water`, `fire`, `electric`, and `grass` to the values "Squirtle", "Charmander", "Pikachu", and "Bulbasaur"

Save & Run | 5/14/2021, 2:41:00 PM - 2 of 2 | Show in CodeLens

```
1 (water,fire,electric,grass)=("Squirtle","Charmander","Pikachu","Bulbasaur")
2
```

Activity: 10 -- ActiveCode (ac12_4_9)

| Result | Actual Value | Expected Value | Notes |
|---|---|---|---|
| Pass | 'Squirtle' | 'Squirtle' | Testing that water is assigned to the correct value. |
| Pass | 'Charmander' | 'Charmander' | Testing that fire is assigned to the correct value. |
| Pass | 'Pikachu' | 'Pikachu' | Testing that electric is assigned to the correct value. |
| Pass | 'Bulbasaur' | 'Bulbasaur' | Testing that grass is assigned to the correct value. |

You passed: 100.0% of the tests

With only one line of code, assign four variables, `v1`, `v2`, `v3`, and `v4`, to the following four values: 1, 2, 3, 4.

Save & Run | 5/14/2021, 2:41:57 PM - 2 of 2 | Show in CodeLens

```
1 (v1,v2,v3,v4)=(1,2,3,4)
2
```

Activity: 11 -- ActiveCode (ac12_4_10)

| Result | Actual Value | Expected Value | Notes |
|---|---|---|---|
| Pass | 1 | 1 | Testing that v1 was assigned correctly. |
| Pass | 2 | 2 | Testing that v2 was assigned correctly. |
| Pass | 3 | 3 | Testing that v3 was assigned correctly. |
| Pass | 4 | 4 | Testing that v4 was assigned correctly. |

You passed: 100.0% of the tests

If you remember, the .items() dictionary method produces a sequence of tuples. Keeping this in mind, we have provided you a dictionary called `pokemon`. For every key value pair, append the key to the list `p_names`, and append the value to the list `p_number`. Do not use the .keys() or .values() methods.

Save & Run 5/14/2021, 2:44:03 PM - 2 of 2 Show in CodeLens

```
pokemon = {'Rattata': 19, 'Machop': 66, 'Seel': 86, 'Volbeat': 86, 'Solrock': 126}
p_names=[]
p_number=[]

for i in pokemon.items():
    p_names.append(i[0])
    p_number.append(i[1])

```

Activity: 12 -- ActiveCode (ac12_4_11)

| Result | Actual Value | Expected Value | Notes | |
|---|---|---|---|---|
| Pass | ['Mac...eat'] | ['Mac...eat'] | Testing that p_name has the correct values | Expand Differences |
| Pass | [19, ... 126] | [19, ... 126] | Testing that p_number hsa the correct values | Expand Differences |
| Pass | '.keys()' | '\npoke... [1])\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |
| Pass | '.items()' | '\npoke... [1])\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |
| Pass | '.values()' | '\npoke... [1])\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |

You passed: 100.0% of the tests

The .items() method produces a sequence of key-value pair tuples. With this in mind, write code to create a list of keys from the dictionary `track_medal_counts` and assign the list to the variable name `track_events`. Do **NOT** use the .keys() method.

Save & Run 5/14/2021, 2:44:37 PM - 2 of 2 Show in CodeLens

```
track_medal_counts = {'shot put': 1, 'long jump': 3, '100 meters': 2, '400 meters':
track_keys=[]

for i in track_medal_counts.items():
    track_keys.append(i[0])

track_events=track_keys

```

Activity: 13 -- ActiveCode (ac12_4_12)

| Result | Actual Value | Expected Value | Notes | |
|---|---|---|---|---|
| Pass | ['100...ump'] | ['100...ump'] | Testing that track_events was created correctly. | Expand Differences |
| Pass | '.keys()' | '\ntrac...keys\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |
| Pass | '.items()' | '\ntrac...keys\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |
| Pass | 'in tr...unts:' | '\ntrac...keys\n' | Testing your code (Don't worry about actual and expected values). | Expand Differences |

You passed: 100.0% of the tests

You have attempted 14 of 14 activities on this page

13.2. Tuple Packing">
ple Packing
13.4. Tuples as Return Values">
✔ Completed. Well Done!
13.4. Tuples as Return Values">Next Section - 13.4. Tuples as Return Values


| Back to top