# Lesson Returning Multiple Values

## The return of many

Up until now every function we have written returned a single value. Even when we returned a list of items, that list is still considered a single value. However, there are times when we want a function to return multiple items. Lets look at the current options:

## Use a List

Although a Python list can hold mixed types (strings, numbers, functions), it's usually a good practice to think of a list as containing homogeneous items. It makes processing the list easier.

```
def get_shopping_item_list():
  return ['apple', 10]
```

The downside is the code calling get_shopping_item (a.k.a. the client) has to remember to index into the list to get the proper value.

### Use a Dictionary

Using a dictionary is also a possibility, but it also puts some coupling (a.k.a. dependence) between the client and the function. Specifically if the author of the function changes the keys of the dictionary, s/he has to inform all those that use the function of the updates.

```
def get_shopping_item_dict():
  return {
    'name': 'apple',
    'count' : 10
  }
```

## Use a Tuple

Returning a tuple is similar to returning a list; however the items returned are immutable (you can't change them).

```
def get_shopping_item_tuple():
  return ('apple', 10)
```

> ***Coder's Log:*** *Use a Class or Object We haven't seen how to make Python objects yet. This is a perfectly viable solution if the returned class seems like a reasonable*

> *container. If the function is returning very disparate items, then making a class or object to hold those items isn't necessarily a good design decision. We will visit making Python Objects in another lesson(s).*

## Unpacking To the Rescue

One of the convenient ways to 'unpack' a tuple into separate values is just to use a list of comma separated variables:

```
values = ('1', 2, 'three')
i, j, k = values
print(i, j, k)
```

When a function returns multiple values (as a tuple), you can easily access each item:

```
def get_shopping_item_tuple():
  return ('apple', 10)

tc = get_shopping_item_tuple()
print(tc[0], tc[1])
```

We can use this feature called value unpacking for the caller to use a comma separated list of variables to hold the result of calling a function that returns a tuple:

```
(fruit, count) = get_shopping_item_tuple()
print(fruit, count)
```

Note that the parenthesis around the tuple in both the function (e.g. get_shopping_item_tuple) and the list are optional:

```
fruit, count = get_shopping_item_tuple()
print(fruit, count)
```

Also since Python is unpacking the returned tuple for you, the values are being copied into the variables and can be changed:

```
t = get_shopping_item_tuple()
t[0] = 'orange' # ERROR (comment before submitting)
```

```
fruit, count = get_shopping_item_tuple()
fruit = 'orange' # OKAY
```

Note that you are changing the value in the copy and not the value in the tuple.

For functions that return a dictionary, if the caller uses a comma separated list to hold the return values, the keys are returned:

```
def get_shopping_item_dict():
  return {
    'name': 'apple',
    'count' : 10
  }

k1,k2 = get_shopping_item_dict()
print(k1, k2)
```

## Unpacking Errors

If you don't provide enough variables in which to unpack, Python will throw a ValueError (be sure to comment out the line that causes the error):

```
def get_shopping_item_dict():
  return {
    'name': 'apple',
    'type': 'granny',
    'count' : 10
  }

k1,k2 = get_shopping_item_dict()
print(k1, k2)
```

## Returning Multiple Items.

Just to reiterate the point (as we saw in the tuple example), you can write a function that returns multiple values (note that the parenthesis is not necessary) and the caller can decide whether to unpack the return values or keep them inside a tuple:

```
def get_shopping_items():
  cart = get_shopping_item_dict()
  keys = cart.keys()
  return cart, list(keys)
```

```
# both of these are valid
both = get_shopping_items()
print(type(both))

items, keys = get_shopping_items()
print(items[keys[0]])
```

**Before you go, you should know:**

- how to return multiple values from a function
- how to unpack a tuple into separate variables

## Lesson Assignment

You will use the statistics library to build some simple results.

### Part 1: tuple of stats

#### Implement the function get_stats

The function get_stats calculates statistics on a sample of values. It does the following:

- its input parameter is a csv string
- a csv (comma separated values) string contains a list of numbers (the sample)
- return a tuple that has 3 values: (n, stdev, mean)
- tuple[0] is the number of items in the sample
- tuple[1] is the standard deviation of the sample
- tuple[2] is the mean of the sample

### Hints:

- get_stats can use the statistics library
- the data should be considered a sample of items (use n-1 for degrees of freedom)
- you can convert a string to a number by using the built-in function float

Once that is done, the following should work:

```
print(get_stats("1, 2.0, 3"))
```

```
import statistics

def get_stats(csv_string):
  '''
  csv_string is a list of values (any values)
  return a tuple that holds (in this order):
  n:     number of items in the sample
  stdev: standard deviation of the n samples
  mean:  ave of the n samples
  '''

  return None
```

## Part 2: Input Cleaning (extra credit)

Implement a clean function that returns a list of numbers from a csv string.

```
def clean(csv_string):
  # return a list of numbers (floats)
  # that are inside the csv_string
  return []
```

The function clean should do the following:

- remove missing items (e.g. "1,2,,3")
- remove items marked with None or are the empty string ("None,,2")
- remove items that are not numbers ("a,b,c,2")
- However, if the item is a string, but a valid number, keep it (e.g. "'1.1', ' 3 '")

So the following example should work:

```
csv = "a, 1, '-2', 2.35, None,, 4, True"
print(clean(csv))
# output is [1.0, -2.0, 2.35, 4.0]
```

Cleaning output like this is very common when the data is generated from users (e.g. surveys, web forms, etc).

Once that is done, update get_stats to use the clean function:

```
def get_stats(csv_string):
  input = clean(csv_string)
  # now do stats on input

  return ...
```

```
# type&run the above example/exercise in this cell
```

## Submission

After implementing all the functions and testing them please download the notebook as "solution.py" and submit to gradescope under "Week13:UPY:Return_Multiple" assignment tab and Moodle.

**NOTES**

- Be sure to use the function names and parameter names as given.
- DONOT use your own function or parameter names.
- Your file MUST be named "solution.py".
- Comment out any lines of code and/or function calls to those functions that produce errors. If your solution has errors, then you have to work on them but if there were any errors in the examples/exercies then comment them before submitting to Gradescope.
- Grading cannot be performed if any of these are violated.