**Your Own Store!**

Okay—on to the core of our project.

Congratulations! You are now the proud owner of your very own Codecademy brand supermarket.

animal\_counts = { "ant": 3, "bear": 6, "crow": 2 }

In the example above, we create a new dictionary called animal\_counts with three entries. One of the entries has the key "ant" and the value 3.

Create a new dictionary called prices using {} format like the example above.

Put these values in your pricesdictionary, in between the {}:

"banana": 4, "apple": 2, "orange": 1.5, "pear": 3

Yeah, this place is really expensive. (Your supermarket subsidizes the zoo from the last course.)

# Investing in Stock

Good work! As a store manager, you’re also in charge of keeping track of your stock/inventory.

Create a stock dictionary with the values below.

"banana": 6, "apple": 0, "orange": 32, "pear": 15

# Keeping Track of the Produce

Now that you have all of your product info, you should print out all of your inventory information.

once = {'a': 1, 'b': 2} twice = {'a': 2, 'b': 4} for key in once: print "Once: %s" % once[key] print "Twice: %s" % twice[key]

1. In the above example, we create two dictionaries, once and twice, that have the same keys.
2. Because we know that they have the same keys, we can loop through one dictionary and print values from both once and twice.

Loop through each key in prices.

Like the example above, for each key, print out the key along with its price and stock information.

Print the answer in EXACTLY the following format:

apple price: 2 stock: 0

Like the example above, because you know that the prices and stockdictionary have the same keys, you can access the stock dictionary while you are looping through prices.

When you're printing, you can use the syntax from the example above.

# Something of Value

For paperwork and accounting purposes, let's record the total value of your inventory. It's nice to know what we're worth!

Let's determine how much money you would make if you sold all of your food.

* Create a variable called totaland set it to zero.
* Loop through the pricesdictionary.
* For each key in prices, multiply the number in prices by the number in stock. Print that value into the console and then add it to total.
* Finally, outside your loop, print total.

|  |
| --- |
| * prices = { * "banana" : 4, * "apple" : 2, * "orange" : 1.5, * "pear" : 3, * } * stock = { * "banana" : 6, * "apple" : 0, * "orange" : 32, * "pear" : 15, * } * for key in prices: * print key * print "price: %s" % prices[key] * print "stock: %s" % stock[key] * total = 0 * for key in prices: * x = prices[key] \* stock[key] * total = total + x * print total |

# Shopping at the Market

Great work! Now we're going to take a step back from the management side and take a look through the eyes of the shopper.

In order for customers to order online, we are going to have to make a consumer interface. Don't worry: it's easier than it sounds!

First, make a list called grocerieswith the values "banana","orange", and "apple".

|  |
| --- |
| groceries = ["banana","orange","apple"] |

# Making a Purchase

Good! Now you're going to need to know how much you’re paying for all of the items on your grocery list.

def sum(numbers): total = 0 for number in numbers: total += number return total n = [1, 2, 5, 10, 13] print sum(n)

1. In the above example, we first define a function called sum with a parameter numbers.
2. We initialize the variable total which we will use as our running sum.
3. For each number in the list, we add that number to the running sum total.
4. At the end of the function, we return the running sum.
5. After the function, we create, n, a list of numbers.
6. Finally, we call the sum(numbers)function with the variable n and print the result.

Define a function compute\_billthat takes one argument food as input.

In the function, create a variable total with an initial value of zero.

For each item in the food list, add the price of that item to total.

Finally, return the total.

Ignore whether or not the item you're billing for is in stock.Note that your function should work for **any** food list.

|  |
| --- |
| shopping\_list = ["banana", "orange", "apple"]  stock = {  "banana": 6,  "apple": 0,  "orange": 32,  "pear": 15  }    prices = {  "banana": 4,  "apple": 2,  "orange": 1.5,  "pear": 3  }  # Write your code below!  def compute\_bill(food):  total = 0  for item in food:  total += prices[item]  return total |

# Stocking Out

Now you need your compute\_bill function to take the stock/inventory of a particular item into account when computing the cost.

Ultimately, if an item isn't in stock, then it shouldn't be included in the total. You can't buy or sell what you don't have!

|  |
| --- |
| shopping\_list = ["banana", "orange", "apple"]  stock = {  "banana": 6,  "apple": 0,  "orange": 32,  "pear": 15  }    prices = {  "banana": 4,  "apple": 2,  "orange": 1.5,  "pear": 3  }  # Write your code below!  def compute\_bill(food):  total = 0  for item in food:  if stock[item] > 0:  total += prices[item]  stock[item] -= 1  return total |