HashMaps:

- In Python you have dictionaries These are similar to lists and arrays in that they let you store indexed values. However, unlike lists or arrays the index value does not have to be an integer. Instead, the index is called a *key* and can be any type.
- For example:

```
stock = {
        "name"     : "GOOG",
        "shares"   : 100,
        "price "   : 490.10
      }
```

- Creates a dictionary with three items, the keys are "**name**", "**shares**", and "**prices**" and the values are "**GOOG**", **100**, and **490.10**

```
stock = {
        "name"    : "GOOG",
        "shares"  : 100,
        "price "  : 490.10
     }
```

- To access the values you use the key, for instance:

```
stock["shares"]
```

- Has the value of 100. You may also add and modify items like:

```
stock["price"] = 510.4
stock["broker"] = "Conman Sam"
```

- In Java we have *maps*.
  - A map is a data structure that stores value/key pairs.
  - One implementation of the map data structure in Java is a *HashMap*. (A *HashMap* is actually a class that implements the *Map* interface, but we will learn about interfaces later in the course.)
  - A HashMap uses hash table to save the values. We will talk about hash tables later.
- You create a HashMap as follows:

```
Map<key type, value type> var = new HashMap<>();
```

- Where neither type can be a primitive type. For example:

```
Map<String, Integer> stockShares = new HashMap<>();
```

- Creates a HashMap named stockShares whose keys are Strings and values are integers.
- What's one big difference you can see now between Python dictionaries and Java HashMaps?

```
Map<String, Integer> stockShares = new HashMap<>();
```

- What is an Integer?
- All primitive types have class counterparts. These kind of classes are called *wrappers*.
- **Integer** is the wrapper class for **int**.
- The other wrapper classes are **Long**, **Float**, **Double**, **Byte**, **Character**, and **Boolean**.
- All these classes have methods for creating an object with the desired value and retrieving the desired value.
- For example we could write:

```
Integer a = Integer.valueof(3);
```

- Creates an Integer object with value 3.

```
int b = a.intVal();
```

- Stores the value of the Integer object **a** into the int **b**.

- Fortunately Java does not require us to do all that typing. When we write:

```
Integer a = 3;
```

  - It is automatically translated to:

```
Integer a = Integer.valueOf(3);
```

  - This is called *autoboxing*.
- Likewise when we write:

```
int b = a;  // where a is of type Integer
```

  - It is automatically translated into:

```
int b = a.intVal();
```

  - This is called *unboxing*.
- So we can treat objects of type **`Integer`** almost exactly like they were of type **`int`**.
- But one should not forget that there is overhead to wrappers that are not in primitives.

```
Map<String, Integer> stockShares = new HashMap<>();
```

- Objects are added to the HashMap or changed using the **put** method. e.g.

  ```
  stockShares.put("GOOG", 75);
  ```

  - Creates an object with key "GOOG" and value 75.

- You may retrieve values from the HashMap by using the get method. e.g.

  ```
  int numShares = stockShares.get("GOOG");
  ```

- If no information is stored in the map for the given key, **`get`** returns **`null`**.

```
Map<String, Integer> stockShares = new HashMap<>();
```

- You can use a default value instead of null by using the method **getOrDefault**.

```
if (stockShares.get("GOOG") == null)
  cnt = 0;
else
  cnt = stockShares.get("GOOG");
```

- Is functionally the same as:

```
cnt = stockShares.getOrDefault("GOOG", 0);
```

```
Map<String, Integer> stockShares = new HashMap<>();
```

- The **containsKey** method will return true if the key is in the map and false otherwise.

```
if (!stockShares.contiansKey("GOOG"))
  cnt = 0;
else
  cnt = stockShares.get("GOOG");
```

  - Is functionally the same as the code on the previous slide.
- You may iterate through all of the keys in a Map by doing the following:

```
for (String key : stockShares.keySet()) {
    // do something with each key
```