UNDERSTANDING ARRAYS

Arrays & hash maps are one of the cornerstones of modern computer programming. It's almost impossible to write a useful program without them, so it's critical that you understand them when you're getting started with programming. In this post, I'll explain arrays & hash maps, how they work, their differences, and when to use them.

Arrays

Arrays are a numerically indexed collection of items. In dynamically typed languages like PHP, Python, and Ruby, arrays can contain any value. You define an array like this:

```
my_array = ["orange", "red", "blue"]
```

Here, we are creating an array named "my_array". [and] are very common ways of creating arrays in most languages. The array defined above has 3 string values. We can access the values using their numerical index. Most programming languages use zero-indexed arrays, meaning the first item starts at 0:

```
print my_array[0] // Prints "orange"

print my_array[1] // Prints "red"
```

```
print my_array[2] // Prints "blue"
```

We can add items to an array using several methods, the simplest of which is just setting the numerical index directly:

```
my_array[3] = "purple"

print my_array[3] // Prints "purple"
```

Note: arrays in **compiled** languages usually have a fixed size, and you must create a new array if you want to add additional items. I won't really go into that here.

You can iterate over each item in an array using multiple methods, but the most common method is a `for` loop (or the `foreach` loop in some languages):

```
for color in my_array:
    print color
```

The above code will print:

```
orange

red

blue

purple
```

The for loop operates one item at a time, traditionally assigning the item to a variable (`color`) and then executing the body of the loop (`print color`) for each item.

Hash Maps

Hash maps, also called hash tables, dictionaries, or associative arrays, are a data structure very similar to arrays. They are a collection of items, indexed with a *key*. Unlike arrays, hash map keys can be strings *or* numbers instead of just numbers. Hash maps are traditionally unordered, but some implementations maintain order (JSON doesn't for example, but PHP does).

Many new programmers think hash maps are just arrays as some languages (e.g. PHP and JavaScript) only have hash maps, which they call arrays. Hash maps can function just like normal arrays, albeit less efficiently in most cases.

Some languages have a different syntax for hash maps than arrays (e.g. Python) and others don't (PHP). Here is an example of creating a hash map:

```
my_hashmap = { "name": "Brandon", "age": "22", "country": "Canada" }
```

This creates a hash map, with 3 keys, name, age, and country. I can add a new attribute just like I would with an array:

```
my_hashmap["occupation"] = "Programmer"
```

And I can iterate over a hash map similarly to an array, but not quite the same:

```
for key in my_hashmap:
    print "My " + key + " is " + my_hashmap[key]
```

Here, for every item in the hash map, I assign the *key* to the variable `key`. Then I get the value from the hash map (example, the first iteration would assign the key “name” to the variable `key`, and calling `my_hashmap[key]` is the same as calling `my_hashmap["name"]` which returns “Brandon”).

Arrays of Hash Maps

It’s very common to see an array of objects or an array of arrays, which can be confusing at first. Take the following example:

```
employees = [
    {"name": "Oliver Queen", "role": "C.E.O."},
    {"name": "Felicity Smoak", "role": "IT Support"},
    {"name": "John Diggle", "role": "Body Guard"}
]
```

Now, say you want to print out each employee's name and role. This is very easy:

```
for employee in employees:

    print employee["name"] + ", " + employee["role"]
```

Which outputs:

```
Oliver Queen, C.E.O.

Felicity Smoak, IT Support

John Diggle, Body Guard
```

Nested Hash Maps

Nested hash maps are another common occurence in programming (especially APIs). Take this example:

```
person = { "name": { "first": "Brandon", "last": "Wamboldt" } }
```

To access a nested attribute, you simply chain keys like so:

```
print person["name"]["first"] // Prints "Brandon"
```

This works with arrays too:

```
things = [["purple", "green", "orange"], ["apple", "orange", "banana"]]

print things[0][0] // Prints "purple"
```

```
print things[1][2] // Prints "banana"
```

Source: http://brandonwamboldt.ca/understanding-arrays-1637/