Reference Types

So far, all of the variables that we've talked about have been primitive types. That meant that each variable could only store one value and they pass their value by copying. I'd like to tell you now about **reference types**.

Reference variables are more complex. They can carry multiple values and they used shared memory.

Arrays for Sequential Values

Arrays are used for sequential values numbered from zero up to the size of the array. Arrays store multiple values in a single variable, and they let you access the values by specifying the index of the value that you want.

```
var words;
words = ["This", "is", "an", "array"];

console.log(words);
console.log(words[0]);
words[0] = "That";
console.log(words);
```

To assign an array, you use square brackets. Between these square brackets, we have four strings separated by commas.

By typing `words[0]` you can access individual elements of an array.

Being able to create a single variable that can contain an arbitrary number of data points each indexed by a number is very powerful. But the ability to store data that's that large comes with a bit of a cost. JavaScript needs to be able to keep track in memory of where that informationis going to be stored, and the fact that it can grow to an arbitrarily large size.

Because of this, unlike primitive variables such as strings and numbers, arrays are stored by reference. The value of an array actually refers to a shared location in memory. Where this can get tricky is when you start assigning the value of an array to another array.

When you assign the value of one array to another array, what you're actually doing is passing a pointer to that shared location in memory, from the first array to the second array. If you make changes to the second array, those changes are also going to affect the first array.

How Reference Variables Work

```
var words1 = ["This", "is", "an", "array"];
var words2 = words1;

console.log(words1);
console.log(words2);

words2[0] = "That";

console.log(words2);
console.log(words1);
```

This proves that arrays are reference variables. When you copy an array you're actually copying a pointer to the location and memory where that array is stored. Changing the second array changes the first array as well. Knowing about this characteristic of reference values can help keep

you from making mistakes that can cause difficult bugs to track down when you're writing JavaScript.

Arrays Can Store Different Types

Another nice thing about arrays is that they can store different types.You don't have to store just strings in an array. When you add values to them, you start at slot 0. But if you add a value to an array, that's more than than one index point past the end of the array, the values that you skip will be filled in, and be assigned the value of `undefined`.

```
var things = [];
var words = ["Another", "array"];

things[0] = words;
things[2] = 2;
things[3] = true;

console.log(things);
console.log(typeof(things[0]));
console.log(typeof(things[1]));
console.log(typeof(things[2]));
console.log(typeof(things[3]));
```

We're going to see that the first value in our array `things` is an another array. The second value in `things` is `undefined`. The third value, which is at index two of `things`, is the number 2. And the fourth value, which is at index three, is the value `true`, a boolean value.

The ability of an array to store different types makes arrays that much more flexible, and allows you to create nested data structures that can be very useful for representing complex data.

Some Array Properties and Methods

As we discussed before, properties contain data about the array, and methods manipulate the contents of the array. There are some methods that only access data in the arrays and there are some methods that modify it, or mutate those arrays.

So, accessors return values from an array and mutators change an array.

```
var words = ["these", "are", "some", "words"];

console.log(words.length);
console.log(words.reverse());
console.log(words.sort());
console.log(words.toString());
console.log(words.join(" # "));
console.log(words.push("augmented"));
console.log(words.pop());
console.log(words);
```

The `length` property tells you how many objects, or items are inside of a single array.

`toString` method converts the contents of an array into a string representation.

`join` method also converts the contents of your array to a string, but allows you to define the separator that joins the elements.

The next methods are mutators.

`reverse` method will reverse the order of the items in the array.

`sort` method will order the elements of the array based on the value that they have. In our example the array will be sorted alphabetically.

`push` method lets you add an item to the end of an array.

`pop` method lets you pull an item off the end of an array. It can be a little confusing keeping track of what the output of `push` and `pop` actually are. The important thing to remember is that `push` adds an item to the end of the array, and `pop` pulls an item off of the end of the array.

These are only some of the array properties and methods that are available to you, but these are some of the most useful ones.

Creating Arrays from Strings

There are also ways to convert strings to arrays. This is convenient if, for example, you want to access the elements of a string and manipulate them. There's a `split` method on strings that lets you break a string into an array of smaller strings.

```
var sentence = "This is a sentence";
var odds = "1,3,5,7,9";
var words = sentence.split(" ");
var oddArray = odds.split(",");
var chunks = sentence.split("s");
var singleItem = sentence.split("Q");

console.log(words);
console.log(oddArray);
console.log(chunks);
console.log(singleItem);
```

So `split` breaks the string in every place it finds the provided delimiter, and put anything that's not a delimiter into the array.

`split` is intelligent enough not to throw an error, if you send it a value that's not actually present in the original string.

Objects for Key:Value Pairs

JavaScript also supports **objects**. Objects let you store properties as keys and values Imagine that we wanted to keep track of the information about a toy:

```
var toy;
toy = {"color":"red", "size":5, "soft":true};

console.log(toy);
console.log(toy["size"]);
console.log(toy.color);
```

So `toy` has a bunch of properties and values. Notice that because we are creating an object, we put properties in between curly braces. So the property is called the key and that's why an object is said to consist of key-value pairs.

With `toy["size"]` and `toy.color` you can access object's individual properties. The latter option which is called dot notation is the preferred one. You can use the dot notation for any property that doesn't include spaces, but if you need to access a property that does have a space or if you need to passa variable to use as a property, you're going to want to use the square brackets.

Objects are one of the most versatile and flexible data types available in JavaScript. Objects form the foundation for JavaScript, which is one of the reasons why many people consider it an object-oriented language.

Objects Can Store Different Types

An object, just like an array, can store different data types and we've already demonstrated that. You can add properties to an empty object or to an existing object, just by assigning a value to that property. You can assign any type of variable to any one of those properties.

```
var toy = {};

toy.name = "Cardboard Box";

toy.tags = ["imagination", "cats", "recyclable"];

toy.size = {
  "units": "inches",
  "width": 10,
  "height": 5,
  "depth": 3
};
```

```
toy.quantity = 10;

console.log(toy.tags[2]);
console.log(toy.size.units);
```

So as you see a propery can contain numbers, strings, arrays and even other objects. You can create very complex data types using objects and arrays.

Objects Are Stored by Reference

Smilar to arrays, objects are stored by reference. This means that an object can be modified by any variable that is set to reference the same object.

```
var toy1 = {"color":"red", "size":5, "soft":true};
var toy2 = toy1;

console.log(toy1.color);
console.log(toy2.color);
toy2.color = "blue";

console.log(toy2.color);
console.log(toy1.color);
```

Comparing Primitive and Reference

First of all primitives, they exist independent of each other. Each primitive can only have one value, and you cannot create properties or methods on instances of a primitive. By comparison, reference objects, such as objects, arrays, and functions, which we're going to be getting to soon, can be of arbitrary size. They use a shared memory space, they can have one or more values. And you can create properties and methods on instances of reference variables.