Variables - Understanding variables in Javascript

Values

The most fundamental unit of information in a program is a value. Values are data. They're how the program maintains state. Values come in two forms in JS: primitive and object.

Primitives

- string
- number
- boolean
- null
- undefined

Objects

- object literals

- Arrays
- Functions
- Classes
- ...basically everything that is not a primitive!

Values are embedded in programs using literals. Different types of values use different **delimeters** to indicate where the value begins and ends. A common delimiter used for strings that you are probably familar with is quotations. In Javascript we have 'single quotes', "double quotes", and `back-ticks`. Single and double quotes function exactly the same...pick one and stick with it. Back-ticks are used to form template literal strings where interpolation can happen.

`typeof` can be used to see what type a stored value is given by Javascript. See if you can identify the delimeters used for the different types of values:

```
typeof 42;
typeof "abc";
typeof true;
typeof undefined;
typeof null;
typeof { a: 1 };
typeof [1, 2, 3];
typeof function hello() {};
```

Any surprises? What does a literal for a basic Object look like? An Array?

Variables

1. Variables are containers for values
2. Must be declared (created) to be used
3. var
4. let
5. const
6. Differences (Which should I use?)

```
var adult = true;

if (adult) {
  var myName = "Bob";
  let age = 24;
}

console.log(myName);
// Bob

console.log(age);
// Error!
```

   Why did we get an error on line 11?

7. `const` declared variables are not "unchangeable", they just cannot be re-assigned.

Strongly Typed? Weakly typed? What does that even mean?

Python is considered a strongly typed language. If you assign a value to a variable Python will not change the type of that variable. For example something like this would throw an error in Python:

```
'foo' + 3 --> TypeError: cannot concatenate 'str' a
```

Javascript is weakly typed...so if you tried the same thing it would work:

```
"foo" + 3; // 'foo3'
```

Javascript knows that it cannot add a string and an integer...but it does know that it can add 2 strings, so it 'coerces' the integer value into a string so the operation will work. Sometimes this is helpful...but sometimes it can cause weird bugs.

Coercion

Coercion is simply the process Javascript uses when it has to do a comparison or arithmetic operation between two values that don't match. It is important to understand that it happens...and that JS has a few guidelines it follows as it is doing it.

1. Javascript prefers numeric comparisons and will coerce to a primitive numeric value if it can
2. In the case of '+' (addition or concatination). If there is a string involved...all values will be converted to strings. That's the only way JS knows to combine mixed values.

Why do I have to declare?

Javascript will actually let you use a variable without declaring it. Just because you can do something doesn't mean you should! This has been one criticism of Javascript because it can lead to bugs. Because of the decision to never break backwards compatibility they couldn't just fix this...so instead they added an option for a new mode to run your code in: Strict mode.

One of the things that changes is if you are in strict mode the browser will throw an error if it finds you using a variable before you declare it. This is a good thing. To turn strict mode on we simply add `'use strict';` to the top of a file (or function). For a more complete description of what strict mode does visit MDN: Strict Mode

```
let myVariable = 4;
// do some stuff with myVariable

// reassign myVariable?
myVarable = 5;

// do some more stuff expecting myVariable to be ch
```

Arrays

Arrays are not primitive values like strings or numbers. This makes them much more flexible. Arrays are lists of values. Each spot of an array can hold any valid value.

In JavaScript, arrays can be declared like this.

```
let names = ["Bob", "Sue", "Jorge", "Svetlana"];
```

You access elements of an array like this

```
let aName = names[0];
```

with zero being the index of the first element in the array. Modifying values in the array is doable also.

```
names[2] = "George";
```

Now the third element of the array is George instead of Jorge.

JavaScript has a lot of Built in Functions(BIF's), or built-in object methods, that you can use to modify arrays. Imagine you needed to use an array to keep track of customers lined up to see a movie. As people show up, they need to line up in the order they arrived. You can simulate this using the push method of array.

If Bob, Sue, George, and Svetlana were already waiting in line for the show in that order, and then Grace showed up too, you would use push to add Grace at the end of the array (the back of the line).

```
names.push("Grace");
console.log(names);
//Displays ['Bob','Sue','George','Svetlana','Grace'
```

Since you can add elements to the end of the array, surely there is a way to remove elements from the end of the array. You do this with the pop method.

```
names.pop();
console.log(names);
//Displays ['Bob','Sue','George','Svetlana']
```

Can you remove elements from the end or add them to the beginning of the array? You bet. There are a large number of methods that you can apply to array. Check them out.

Strings

Strings are a type of primitive value. The share some similarities with Arrays however. If you are wondering how long a string is for example, you can look at it's `length` property.

```
let myString = "This is my String!";
myString.length; // 18
```

You can also access each character of a string with an index just like with an array.

```
let myString = "This is my String!";
myString[2]; // "i"  - remember that indexes start
```

String delimiters

Strings can be delimited in three ways:

1. double quotes '
2. single quotes "
3. Backtick `

Double quotes and single quotes are completely interchangable. Pick one and stick with it. Backticks are used to form what are know as template literal strings These strings have the added feature of allowing the embedding of variables and other bits of

code inside of a string with placeholders `${ }`. For example we can add variable values into a string with concatination like this:

```
const hello = "Hello";
const world = "World";
const complexString = hello + " " + world;
```

Or we can use a template literal string and do it this way:

```
const hello = "Hello";
const world = "World";
const complexString = `${hello} ${world}`;
```

This becomes particularly helpful when creating strings of HTML to be inserted into the DOM:

```
const myArray = ["one", "two", "three"];
const htmlString = `
<ol>
  <li>${myarray[0]}</li>
  <li>${myarray[1]}</li>
  <li>${myarray[2]}</li>
</ol>`;
```

Template literals allow for the use of formatting in these cases as well which aids readability.

Some of the detail above comes from [You Don't Know JS Yet Ch 2]. Highly recommend reading to learn more.