Data Types

We're going to spend a little time on data types. When we create a variable, the data that we define has a type. There are `primitive types` and `reference types`. The big difference under the hood is the way that they're stored and accessed from memory.

1. **Primitive Types** - Stored directly in the location that the variables is accessed
2. **Reference Types (Objects)** - Accessed by reference

Primitive Data Types

That may not make much sense right now, but in the next video I'm going to go more in depth on how this works under the hood. So first well talk about the 7 primitive types of JavaScript.

The 7 primitive data types are the following

- `String` - a sequence of characters. In JavaScript, strings can be enclosed within the single or double quotes
- `Number` - Represent both integer and floating-point numbers
- `Boolean` - Represent a logical entity and can have two values: true or false
- `Null` - Represents the intentional absence of any object value
- `Undefined` - A variable that has not been assigned a value is undefined.
- `Symbol` - It is a built-in object whose constructor returns a symbol-that is unique
- `BigInt` - New data type used for numbers that are greater than the Number type can handle.

Dynamic vs Static Types

JavaScript is a "dynamically-typed" language. This means that when we create a variable or a function, we don't explicitly assign the type. JavaScript assigns the type of that value dynamically. In other words, the type is associated with the **value**, not the **variable**. So you can have a variable value be a string and then later in the script, change it to a number. You probably won't do that very often, but you can.

There are other languages that are "statically-typed". This is where you would explicitly define the type of data. Java is an example of a statically-typed language. There's also a language called TypeScript, which is essentially JavaScript with some extra features, including types. So is TypeScript, you could do this.

```
const y:number = 100
```

You can see we defined it as a **number**. So now that **y** variable's value HAS to always be a number.

This is not something we can do with vanilla JavaScript. The advantage of static types is that it makes your code more robust and less prone to errors. The downside is you need to write a bit more code.

Assigning Variables

String

A string is a "string" of characters wrapped in either single or double quotes. Strings can include any number, letter or symbol

```
const firstName = 'Sara';
```

Number

Any number in JavaScript is the `Number` type, including floats and decimals. Some languages have separate types for floats and integers. JavaScript does not. Numbers are not wrapped in quotes.

```
const age = 30;
const temp = 98.9
```

Boolean

A boolean is a true or false value

```
const hasKids = true;
```

Null

Intentional absence

```
const aptNumber = null;
```

Undefined

Undefined represents a variable that has not been defined a value.

```
let score;
const score = undefined;
```

Symbol

A “symbol” represents a unique identifier. We will talk more about what they are used for later, but this is how we can create one

```
const id = Symbol('id');
```

BigInt

BigInt is a new primitive type and represents integers that are out of the range of the Number type

```
const n = 9007199254740991n;
```

typeof Operator

To check the type of a variable in JavaScript, you can use the typeof operator

```
console.log(typeof name)
```

Tip: If you run typeof on a variable that holds `null`, you will not get null, you will get `object`. This is generally regarded as a mistake. More info here

```
console.log(typeof aptNumber)
```

Reference Data Types (Objects)

We talked about the primitive data types that are stored directly in memory where they are accessed. Reference types work a little differently under the hood. There are basically 3 types that are passed by reference

- `Arrays`
- `Functions`
- `Objects`

Technically, these are all "objects", meaning that they have properties and methods.

Reference types or "objects" are a non-primitive value and when assigned to a variable, the variable is given a **reference** to that value. The reference points to the object's location in memory. Unlike primitives, where the variable contains the actual value. I'll talk more about this in the next video.

Arrays

I know we haven't gone over arrays yet so don't worry if you've never worked with them. They're essentially a data structure that can hold multiple values.

So if we write

```
const numbers = [1,2,3,4]
```

We have created an array in memory and a variable that points to the address or location of that array.

Object Literals

Objects are comma separated lists of `name-value pairs`. We'll get into them later, but just to give you an example here

```
{
  name: 'John',
  age: 30
}
```

Functions

Functions are also objects in JavaScript. They can have `properties` and `methods`. What distinguishes them from other objects is that they can be called.

```
const sayHello = function() {
  return 'Hello';
}

sayHello()
```

If we check the type with typeof, we'll get "object" for arrays and object literals, but we actually get "function" for a function. This is still an object, it's just identified as a function object, which can be called as you can see here