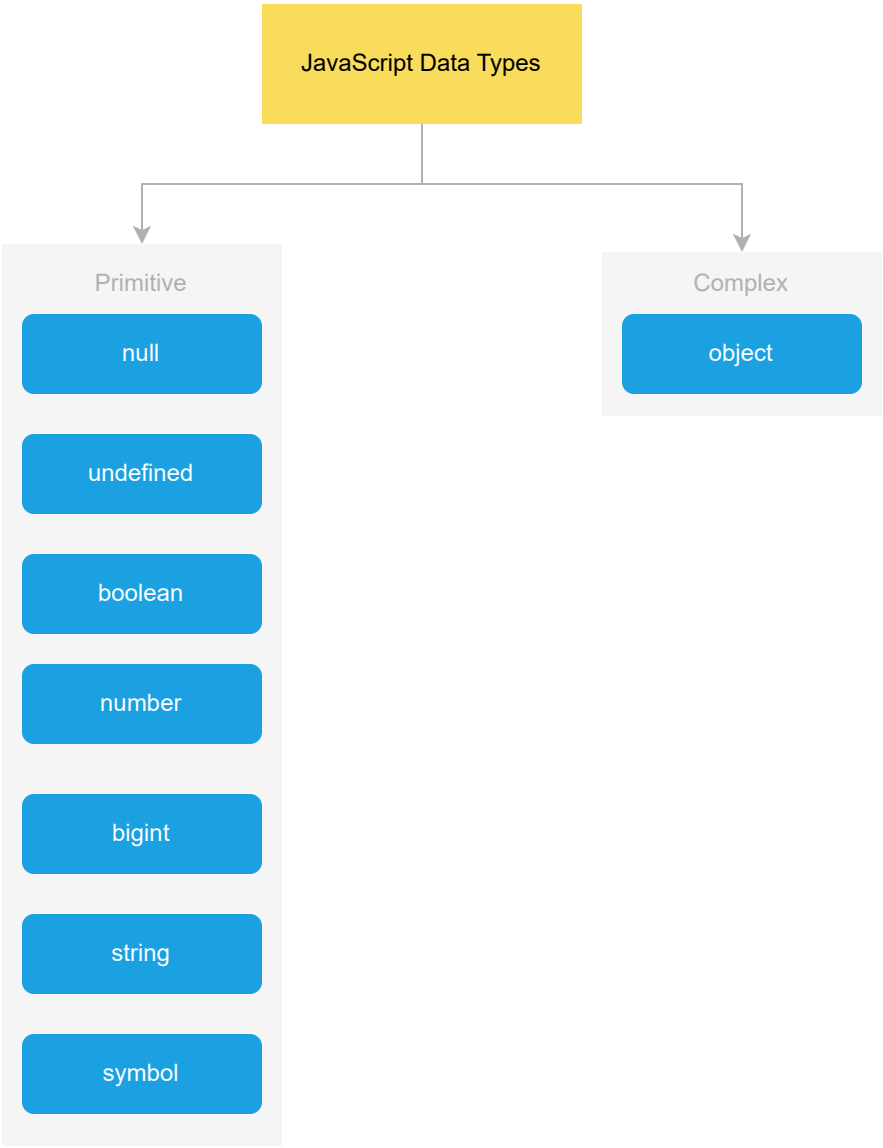
JavaScript Data Types

Summary: in this tutorial, you will learn about JavaScript data types and their unique characteristics.

JavaScript has the primitive data types:

1. `null`
2. `undefined`
3. `boolean`
4. `number`
5. `string`
6. `symbol` – available from ES2015
7. `bigint` – available from ES2020

and a complex data type `object` .



JavaScript is a dynamically typed language, meaning that a variable isn't associated with a specific type. In other words, a variable can hold a value of different types. For example:

```
let counter = 120; // counter is a number
counter = false;   // counter is now a boolean
counter = "foo";   // counter is now a string
```

To determine the current type of the value stored in a variable, you use the `typeof` operator:

```
let counter = 120;
console.log(typeof(counter)); // "number"

counter = false;
console.log(typeof(counter)); // "boolean"
```

```
counter = "Hi";
console.log(typeof(counter)); // "string"
```

Output:

```
"number"
"boolean"
"string"
```

The undefined type

The `undefined` type is a primitive type that has only one value `undefined`. By default, when a variable is declared but not initialized, it defaults to `undefined`.

Consider the following example:

```
let counter;
console.log(counter);        // undefined
console.log(typeof counter); // undefined
```

In this example, the `counter` is a variable. Since `counter` hasn't been initialized, it is assigned the value `undefined`. The type of `counter` is also `undefined`.

It's important to note that the `typeof` operator also returns `undefined` when you call it on a variable that hasn't been declared:

```
console.log(typeof undeclaredVar); // undefined
```

The null type

The `null` type is the second primitive data type that also has only one value `null`. For example:

```
let obj = null;
```

```
console.log(typeof obj); // object
```

> The `typeof null` returns `object` is a known bug in JavaScript. A proposal to fix was rejected due to the potential to break many existing sites.

JavaScript defines that `null` is equal to `undefined` as follows:

```
console.log(null == undefined); // true
```

The number type

JavaScript uses the `number` type to represent both integer and floating-point numbers.

The following statement declares a variable and initializes its value with an integer:

```
let num = 100;
```

To represent a floating-point number, you include a decimal point followed by at least one number. For example:

```
let price = 12.5;
let discount = 0.05;
```

Note that JavaScript automatically converts a floating-point number into an integer if the number appears to be a whole number.

The reason is that Javascript always wants to use less memory since a floating-point value uses twice as much memory as an integer value. For example:

```
let price = 200.00; // interpreted as an integer 200
```

To get the range of the number type, you use `Number.MIN_VALUE` and `Number.MAX_VALUE` . For example:

```
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
console.log(Number.MIN_VALUE); // 5e-324
```

Also, you can use `Infinity` and `-Infinity` to represent the infinite number. For example:

```
console.log(Number.MAX_VALUE + Number.MAX_VALUE); // Infinity
console.log(-Number.MAX_VALUE - Number.MAX_VALUE); // -Infinity
```

NaN

`NaN` stands for Not a Number. It is a special numeric value that indicates an invalid number. For example, the division of a string by a number returns `NaN` :.

```
console.log('a'/2); // NaN;
```

The `NaN` has two special characteristics:

- Any operation with `NaN` returns `NaN` .
- The `NaN` does not equal any value, including itself.

Here are some examples:

```
console.log(NaN/2); // NaN
console.log(NaN == NaN); // false
```

The string type

In JavaScript, a string is a sequence of zero or more characters. A string literal begins and ends with either a single quote(`'`) or a double quote (`"`).

A string that begins with a double quote must end with a double quote, and a string that begins with a single quote must also end with a single quote. For example:

```
let greeting = 'Hi';
```

```
let message  = "Bye";
```

If you want to use single quotes or double quotes in a literal string, you need to use the backslash to escape them. For example:

```
let message = 'I\'m also a valid string'; // use \ to escape the single quote (')
```

JavaScript strings are immutable, meaning that they cannot be modified once created. However, you can create a new string from an existing one. For example:

```
let str = 'JavaScript';
str = str + ' String';
```

In this example:

- First, declare the `str` variable and initialize it to a string of `'JavaScript'` .
- Second, use the `+` operator to combine `'JavaScript'` with `' String'` to make its value as `'Javascript String'` .

Behind the scene, the JavaScript engine creates a new string that holds the new string `'JavaScript String'` and destroys the original strings `'JavaScript'` and `' String'` .

The following example attempts to modify the first character of the string JavaScript:

```
let s = 'JavaScript';
s[0] = 'j';
console.log(s)
```

The output is:

```
'JavaScript'
```

But not:

```
'javaScript'
```

The boolean type

The `boolean` type has two literal values: `true` and `false` in lowercase. The following example declares two variables that hold the boolean values.

```
let inProgress = true;
let completed = false;

console.log(typeof completed); // boolean
```

JavaScript allows values of other types to be converted into boolean values of `true` or `false` .

To convert values of other types into boolean values, you use the `Boolean()` function.

The following table displays the conversion rules:

Type	true	false
string	non-empty string	empty string
number	non-zero number and Infinity	0, NaN
object	non-null object	null
undefined		undefined

For example:

```
console.log(Boolean('Hi'));// true
console.log(Boolean(''));  // false

console.log(Boolean(20));  // true
console.log(Boolean(Infinity));  // true
console.log(Boolean(0));  // false
```

```
console.log(Boolean({foo: 100}));  // true on non-empty object
console.log(Boolean(null));// false
```

The symbol type

JavaScript introduced a new primitive type in ES6: the `symbol` . Unlike other primitive types, the `symbol` type does not have a literal form.

To create a symbol, you call the `Symbol` function as follows:

```
let s1 = Symbol();
```

The `Symbol` function creates a new unique value every time you call it.

```
console.log(Symbol() == Symbol()); // false
```

Note that you'll learn more about symbols in the symbol tutorial.

The bigint type

The `bigint` type represents the whole numbers that are larger than $2^{53} - 1$. To form a `bigint` literal number, you append the letter `n` at the end of the number:

```
let pageView = 9007199254740991n;
console.log(typeof(pageView)); // 'bigint'
```

And you'll learn more about the `bigint` type here.

The object type

In JavaScript, an object is a collection of properties, where each property is defined as a key-value pair.

The following example defines an empty object using the object literal syntax:

```
let emptyObject = {};
```

The following example defines the `person` object with two properties: `firstName` and `lastName`.

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

A property name of an object can be any string. You can use quotes around the property name if it is not a valid identifier.

For example, if the `person` object has a property `first-name`, you must place it in the quotes such as `"first-name"`.

A property of an object can hold an object. For example:

```
let contact = {
    firstName: 'John',
    lastName: 'Doe',
    email: 'john.doe@example.com',
    phone: '(408)-555-9999',
    address: {
        building: '4000',
        street: 'North 1st street',
        city: 'San Jose',
        state: 'CA',
        country: 'USA'
    }
}
```

The `contact` object has the `firstName`, `lastName`, `email`, `phone`, and `address` properties.

The `address` property itself holds an object that has `building`, `street`, `city`, `state`, and `country` properties.

To access an object's property, you can use

- The dot notation (`.`)
- The array-like notation (`[]`).

The following example uses the dot notation (`.`) to access the `firstName` and `lastName` properties of the `contact` object.

```
console.log(contact.firstName);
console.log(contact.lastName);
```

If you reference a property that does not exist, you'll get an `undefined` value. For example:

```
console.log(contact.age); // undefined
```

The following example uses the array-like notation to access the `email` and `phone` properties of the `contact` object.

```
console.log(contact['phone']); // '(408)-555-9999'
console.log(contact['email']); // 'john.doe@example.com'
```

Quiz

Summary

- JavaScript has the primitive types: `number` , `string` , `boolean` , `null` , `undefined` , `symbol` and `bigint` and a complex type: `object` .