AkiraChix
SHE BUILDS • SHE SERVES • SHE LEADS

JS

Outline

- Values & Types
- Objects
- Built-In Type Methods
- Comparing Values
  - Coercion
  - Truthy & Falsy
  - Equality
  - Inequality

Values and Types

Primitive Types

From our previous class, we looked at the following JS primitive types:

- string
- number
- boolean
- null
- undefined
- object
- symbol (new to ES6)

Native Types

Commonly used natives include:

- String()
- Number()
- Boolean()
- Array()
- Object()
- Function()
- RegExp()
- Date()
- Error()
- Symbol() -- added in ES6!

As you can see, these natives are actually built-in functions.

JS provides a typeof operator that can introspect a value and tell you what type it is:

```
var a;
typeof a;                // "undefined"

a = "hello world";
typeof a;                // "string"

a = 42;
typeof a;                // "number"
```

```
a = true;
typeof a;                 // "boolean"

a = null;
typeof a;                 // "object" <-- weird, bug

a = undefined;
typeof a;                 // "undefined"

a = { b: "c" };
typeof a;                 // "object"
```

null Bug

typeof null is an interesting case, because it errantly returns "object", when you'd expect it to return "null".

Warning: This is a long-standing bug in JS, but one that is likely never going to be fixed. Too much code on the Web relies on the bug and thus fixing it would cause a lot more bugs!

Objects

An object's type refers to a key value data structure where you can set properties properties (named locations) that each hold their own values of any type.

```
var person = {
    name: "Jane Doe",
    age: 21,
    loggedIn: true
}
```

Note: Objects are important in JS since, Arrays and Functions are value types that are built on top of object type (subtype).

Properties can either be accessed with dot notation e.g.

```
person.name;       // "Jane Doe"
person.age;        // 21
person.loggedIn;   // true
```

Or bracket notation e.g.

```
person['name'];    // "Jane Doe"
person['age'];     // 21
person['loggedIn']; // true
```

Dot notation is shorter and generally easier to read, and is thus preferred when possible.

When using bracket notation always provide a key wrapped in a string literal i.e. "..." or '...'

Bracket notation can be useful if you want to access a property/key but the name is stored in another variable such as:

```
var obj = {
        a: "hello world",
        b: 100
};

var b = "a";
obj[ b ];          // "hello world"
obj[ "b" ];        // 100
```

Arrays

An array is an object that holds values (of any type) not particularly in named properties/keys, but rather in numerically indexed positions. For example:

0: “Jane Doe”	1: 21	2: true

```
var arr = ["Jane Doe", 21, true];

arr[0];                  // "Jane Doe"
arr[1];                  // 21
arr[2];                  // true
arr.length;              // 3

typeof arr;              // "object"
```

TIP: The best and most natural approach is to use arrays for numerically positioned values and use objects for named properties.

Functions

Functions are a main subtype of objects with a typeof “function”.

```
function foo() {
        return 21;
}
typeof foo;             // "function"
typeof foo();           // "number"
```

They also can have properties, but this should be used in limited cases:

```
foo.bar = "hello world";
typeof foo.bar;     // "string"
```

AkiraChix
SHE BUILDS • SHE SERVES • SHE LEADS

Built-In Type Methods

The built-in types and subtypes we've just discussed have behaviors exposed as properties and methods that are quite powerful and useful.

```
var a = "hello world";
var b = 3.14159;

a.length;               // 11
a.toUpperCase();        // "HELLO WORLD"
b.toFixed(4);           // "3.1416"
```

For any primitive type there is an equivalent native type e.g. primitive type string pairs with native type String, primitive type number pairs with native type Number.

When you access a built-in method like .toUpperCase(), JS automatically "boxes" or wraps the value with its object wrapper counterpart which gives access to the method.

Read more:
https://github.com/getify/You-Dont-Know-JS/blob/master/types%20%26%20grammar/ch3.md

Comparing Values

JS supports two main types of value comparison: equality and inequality

The result of any comparison is a strictly boolean value i.e. true or false

To understand comparison, you need to be aware of coercion.

Coercion

JS is a 'weak' typed or dynamically typed language. This means type is not enforced on its variables but rather inferred from its values.

There are two types of coercion: Explicit and Implicit coercion

- Explicit coercion is an obvious attempt from the author to convert a value of one type to another type.
- Implicit coercion occurs as a less-obvious side effect of another operation.

```
let a = 100;

let b = a + “ ”;        // Implicit coercion
console.log(b);         // '100'

let c = String(a);      // Explicit coercion
console.log(c);         // '100'
```

What if we try to compare two values: '100' vs 100?

```
let a = 100;
let b = '100';

a == b;  // true
a === b; // false
```

JS will likely coerce the values.

Coercion is a necessary evil of untyped languages.

TIP: Always use the strict comparison unless you can justify not doing so.

Truthy & Falsy

The specific list of "falsy" values in JS is as follows:

- " " (*empty string*)
- 0, -0
- NaN (*invalid number*)
- null
- undefined
- false

Any value that's not on this "falsy" list is "truthy." Here are some examples of those:

- "hello"
- 42
- true
- [ ], [1, "2", 3] (*arrays*)
- { }, { a: 42 } (*objects*)
- function foo() { .. } (*functions*)

Equality

There are four equality operators: ==, ===, !=, and !==.

What is the difference between the equality == and strict equality === operators?

Equality operator == checks for value equality and applies coercion

Strict equality operator === checks for both value and type equality and does not apply coercion

The logical negation operator (!) reverses the equality operators to their non-equal form. i.e. != form pairs with ==, and the !== form pairs with ===.

Note: non-equality is not inequality

However, note that when comparing two non-primitive values, like objects (functions and arrays), their values are actually held by reference.

This means that the comparison will simply check whether the references match but not the underlying values.

Are these two arrays equal?

```
var a = [1,2,3];
var b = [1,2,3];

a == b;  // false
```

What about if we compare an array with a string of similar values separated by commas?

```
var c = "1,2,3";

a == c;    // true
b == c;    // true
```

Inequality

We use relational comparison operators to test for inequality.
They are <, >, <=, and >=

```
14 > 10;  // true
```

In JS, strings can also be compared for inequality, using typical alphabetic rules

```
"bar" < "foo"  // true
```

Relational comparison operators allow coercion when testing for inequality

```
var a = 99;
var b = "100";
var c = "101";

a < b; // true
b < c; // true
```

Code Commenting

Code commenting is the practice of sprinkling short, normally single-line notes throughout your code.

These notes are called comments.

They explain how your program works, and your intentions behind it.

Code Commenting

Comments don’t have any effect on your program, but they are invaluable for people reading your code. Here’s an example of code commenting in action:

```
// listing current stock of fruits for my green kiosk

var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

Try it Yourself »

Assignment

Write a program to calculate purchases from a kiosk. For e.g. given the fruit name 'orange' the program should fetch its price (30.00) and multiply with the quantity requested (2) and return total cost in this printed format (2 Oranges for KES 60.00).

calculateFruitCost(fruitName, quantity)

For example, calculateFruitCost('orange', 2) should return "2 Oranges for KES 60.00"

Next class

- Variables
  - Function Scopes
    - Hoisting
    - Nested Scopes
- Conditionals
- Strict Mode

Course material: https://github.com/getify/You-Dont-Know-JS/blob/master/up%20%26%20going/ch2.md

Next class

- Video of the week - https://www.youtube.com/watch?v=9ooYYRLdg_g