Types and Coercion

“People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones.”

– Donald Knuth

Agenda

- Types
- Values
- Natives
- Coercion (Implicit, Explicit)
- Exercises
- Assignment

Types

Built-in Types

- null
- undefined
- boolean
- number
- string
- object
- symbol—added in ES6!

All of these types **except object** are called “**primitives**”.

The *typeof* Operator

Inspects the type of the given value.

```
typeof undefined     === "undefined";// true
typeof true          === "boolean";  // true
typeof 42            === "number";   // true
typeof "42"          === "string";   // true
typeof { life: 42 }  === "object";   // true
typeof Symbol('foo') === "symbol";   // true

// special case
// It would have been nice if it returned "null"
typeof null === "object"; // true

// function “subtype” of object
typeof function a(){ /* .. */ } === "function"; // true

// array "subtype" of object
typeof [1,2,3] === "object"; // true
```

Values as Types

- In JavaScript, **variables don't have types—values have types.**
- **undefined** Versus "**undeclared**"

```
var a = 42;
typeof a;               // "number"
a = true;
typeof a;               // "boolean"
typeof "Some String"; // "string"
```

Values

Arrays

- Arrays are just **containers** for any type of value.
- You **don't need to presize** your arrays.
- Arrays are **numerically indexed**.
- They also are objects that can have string **keys/properties** added to them.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

```
// Create an Array
let fruits = ['Apple', 'Banana'];

// Access an Array item using the index position
let first = fruits[0] // Apple
let last = fruits[fruits.length - 1] // Banana

// Loop over an Array
fruits.forEach(function(item, index, array) {
   console.log(item, index)
})

//Add an item to the end of an Array
let newLength = fruits.push('Orange')
```

Strings

- JavaScript strings are immutable, while arrays are quite mutable.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

```
// Creating strings
const string1 = "A string primitive";
const string2 = 'Also a string primitive';
const string3 = `Yet another string primitive`;

// Character access
'cat'.charAt(1) // returns "a"
'cat'[1] // returns "a"
```

Numbers

- JavaScript has just one numeric type: number. This type includes both “integer” values and fractional decimal numbers.
- Small Decimal Values, the most (in)famous side effect of using binary floating-point numbers
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

```
// Literal syntax
const someNr = 123    // one-hundred twenty-three
const anotherNR = 123.0  // same
123 === 123.0  // true

// Function syntax
Number('123')  // returns the number 123
Number('123') === 123  // true
Number("unicorn")  // NaN
Number(undefined)  // NaN

const biggestNum     = Number.MAX_VALUE
const smallestNum    = Number.MIN_VALUE
const infiniteNum    = Number.POSITIVE_INFINITY
const negInfiniteNum = Number.NEGATIVE_INFINITY
const notANum        = Number.NaN

// Testing for Integers
Number.isInteger( 42 ); // true
Number.isInteger( 42.000 ); // true
Number.isInteger( 42.3 ); // false
```

Special Values

- **null** is an empty value.
- **undefined** is a missing value.
- **NaN** invalid number
- **Infinity**
- **Zeros** - JavaScript has both a normal zero 0 and a negative zero -0.

```
var a = 2 / "foo";      // NaN
typeof a === "number";  // true
Number.isNaN( a );

//Infinities
var a = 1 / 0; // Infinity
var b = -1 / 0; // -Infinity

// Zeros
var a = 0 * 10; // 0
var b = 0 * -3; // -0
```

Value Versus Reference

- The type of the value solely controls whether that value will be assigned by value- copy or by reference-copy.
- Simple values (aka scalar primitives) are always assigned/passed by value-copy: null, undefined, string, number, boolean, and ES6's symbol.
- Compound values—objects (including arrays, and all boxed object wrappers) and functions—always create a copy of the reference on assignment or passing.
- Special refs: https://eloquentjavascript.net/04_data.html (For homework)

Natives

Most Commonly Used Natives

- String()
- Number()
- Boolean()
- Array()
- Object()
- Function()
- RegExp()
- Date()
- Error()
- Symbol()

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

Boxing Wrappers

- JavaScript provides object wrappers around primitive values, known as natives (String, Number, Boolean, etc). These object wrappers give the values access to behaviors appropriate for each object subtype (String#trim() and Array#concat(..)).

Date(..) and Error(..)

The Date(..) and Error(..) native constructors are much more useful than the other natives, because there is no literal form for either.

The main reason you'd want to create an error object is that it captures the current execution stack context into the object.

Coercion

Converting Values

- Converting a value from one type to another is often called “**type casting**,” when done **explicitly**, and “**coercion**” when done **implicitly** (forced by the rules of how a value is used).
- JavaScript coercions always result in one of the scalar primitive values, like **string**, **number**, or **boolean**.

ToString

- **null** becomes "null"
- **undefined** becomes "undefined"
- **true** becomes "true".
- **numbers** are generally expressed in the natural way you'd expect

ToNumber

- **true** becomes **1**
- **false** becomes **0**
- **undefined** becomes **NaN**,
- (curiously) **null** becomes **0**
- for **a string** value essentially works for the most part like the rules/syntax for numeric literals. If it fails, the result is NaN.

ToBoolean

Falsy values

- undefined
- null
- false
- +0, -0
- NaN
- ""

Truthy values

- anything not explicitly on the falsy list is therefore truthy.

Explicit Coercion

We can use the built-in functions: String(), Number() and Boolean, but very importantly, we do not use the new keyword in front of them.

```
// Number to String
var myNr = 42;
var str1 = String( myNr ); // "42"
var str2 = myNr.toString(); // "42"

// String to Number
var myString = "3.14";
var nr1 = Number( myString ); // 3.14
// unary operator
var nr2 = +myString; // 3.14
// parseInt
var nr3 = parseInt(myString, 10); // 3

// To Boolean
var myBool = '0';
var bool1 = Boolean(myBool);
var bool2 = !!myBool;
// The ? : ternary operator
var bool3 = myBool ? true : false;
```

Implicit Coercion Primitive to String

Primitives are coerced to string when using the binary **+ operator**, **if any operand is a string**.

```
// number + string
123 + "hello"; // "123hello"

// string + boolean
"123" + true; // "123true"

// string + boolean + number
123 + "hello" + false; // "123hellofalse"

// string + undefined
"123" + undefined; // "123undefined"

// string + null
"123" + null; // "123null"
```

Implicit Coercion Primitive to Number

Primitives are coerced to number when using **comparison operators** and when using **arithmetic operators** (except for + when one operand is a string)

```
"5" >= 1
"true" > 1
let a = "5" - 1
let b = "10" / 2
let c = true * "5"
```

Implicit Coercion Primitive to Boolean

- The test expression in an **if** (..) statement
- The test expression (**second clause**) in a for (.. ; **..** ; ..)
- The test expression in **while** (..) and **do..while**(..) loops
- The test expression (first clause) in ? : **ternary** expressions
- The || (“logical or”) and && (“logical and”) operators

```
// number and string
123 && "hello"
-> true && true
-> "hello"

// string and string
"true" || "false"
-> true || true
-> "true"

var a = "some text"
if(a) {
 console.log('true')
}
else {
 console.log('false')
}
```

Exercises

Exercises

- The test expression in an **if** (..) statement
- The test expression (**second clause**) in a for (.. ; **..** ; ..)
- The test expression in **while** (..) and **do..while**(..) loops
- The test expression (first clause) in ? : **ternary** expressions
- The || (“logical or”) and && (“logical and”) operators

```
// number and string
123 && "hello"
-> true && true
-> "hello"

// string and string
"true" || "false"
-> true || true
-> "true"

var a = "some text"
if(a) {
 console.log('true')
}
else {
 console.log('false')
}
```

Assignment

Read & understand the following

- Values https://eloquentjavascript.net/01_values.html
- Data https://eloquentjavascript.net/04_data.html
- Program structure https://eloquentjavascript.net/02_program_structure.html

Exercise the following

- All 5 chapters https://ydkjs-exercises.com/types-grammar

Build a console program - 1

Using the knowledge you gained do the following

- Display a prompt feedback into an alert to a question like: What's your favorite movie? E.g. My favorite movie is Batman (not)
- Convert a string entered in a prompt to a Number
- If the number entered is > 16 console log "You are eligible" otherwise" console log you must be at least 16 years old. It seems you are X (where X is the actual number)

Refs:

- https://www.w3schools.com/js/js_popup.asp

Build a console program - 2

Using the knowledge you gained so far build a calculator in console

- First prompt will ask you for the operator
- Second prompt will ask you for the first number
- Third prompt will ask you for the second number
- In the end the console will display: Number1 (actual number) + (actual operator) Number 2 (actual number) = Result e.g. (3 + 5 = 8)

Refs:

- https://www.w3schools.com/js/js_popup.asp
- https://www.tutorialsteacher.com/javascript/javascript-operators

Build a console program – 3 (extra mile)

Using the knowledge you gained so far build a program in console that will:

- ask you your Name, and your Age and afterwards it will display them in your console like this: Hello YourName, I'm only a console but it's nice to meet you.
- I've heard you are XY of age, that means you were born in YYYY

Refs:

- https://www.w3schools.com/js/js_popup.asp
- https://www.w3schools.com/js/js_dates.asp (you can google online how to substract time – days, months, years) – no libraries allowed, only javascript
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

References

References

- https://www.w3schools.com/js/js_datatypes.asp
- https://www.w3schools.com/js/js_popup.asp
- https://www.w3schools.com/js/js_dates.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

- **Books:**
- https://eloquentjavascript.net
- https://xiaoguo.net/~books/Program/You_Dont_Know_JS_Up_and_Going.pdf