Basic JavaScript

Junghoo Cho

cho@cs.ucla.edu

Basic JavaScript - Junghoo Cho - cho@cs.ucla.edu

JavaScript

- Originally created as simple script to manipulate Web pages
- Now runs everywhere! (including phone, desktop, server)
  - Supported by most modern browsers
  - Node.JS (JavaScript interpreter) runs almost everywhere
  - Allows running arbitrary code inside browser
    - Both blessing and curse
- Current standard is much more complex than anyone ever ir

JavaScript History (1)

- 1995 Netscape Navigator added a support for a simple script language named “LiveScript”
  - Renamed it to “JavaScript” in 1996
  - JavaScript has nothing to do with Java!
- 1997 ECMA International standardized the language submitted Netscape
  - ECMAScript: Official name of the standard
  - Javascript: What people call it

JavaScript History (2)

- 1998 ECMAScript 2
- 1999 ECMAScript 3
- ECMAScript 4 abandoned (due to disagreement)
- 2009 ECMAScript 5
- 2015 ECMAScript 6 (= ECMAScript 2015)
- Yearly release of new standard from ECMAScript 2015

JavaScript History (3)

- Our lectures are mainly based on ECMAScript 2015 (ES6)
  - ES6 removes much ugliness of old JavaScript
  - Many books and online tutorials are based on ES5
    - A lot of ES5 legacy code exist today
    - Our syntax may be different from these

Basic Syntax (1)

- Basic syntax is very similar to Java/C

```
if (cond) { stmt; } else if (cond) { stmt; }
switch (a) { case 1: break; ...; default: ...; }
while (i < 0) { stmt; }
for (i=0; i < 10; i++) { stmt; }
for (e of array) { stmt; }
try { throw 1; } catch (e) { stmt; } finally { stmt; }
```

- JavaScript Playground

Basic Syntax (2)

- Arithmetic operators: same as Java/C
  - +, - -, %, …
- Bitwise operators: same as Java/C
  - ~, &, |, ^, …
- Logical operators: same as Java/C
  - !, &&, | |

Basic Syntax (3)

- Comparison operators: mostly similar to Java/C, but
  - `==`/`!=` true if operands have the same *value* (after type conversion)
  - `===`/`!==` true only if operands have the same *value and type* (no autc conversion)

```
console.log(3 == "3");
console.log(3 === "3");
```

  - When operands are objects, `==`/`===` is true only if both operands refe same object
    - More on this later

Basic Syntax (4)

- JavaScript is *case sensitive*
  - But HTML is *NOT*
  - This discrepancy sometimes causes confusion.
- JavaScript identifiers (e.g., variable name) may have letters, r and ***$***
  - Some frameworks/libraries gives special meaning to symbols `$` and

Variable Declaration

- `let name=value;`
  - E.g. `let x=10;`
  - Dynamic variable type
- A variable can be used without an explicit `let` declaration
  - "Laissez faire" philosophy
  - Becomes a global variable
  - But this is ***strongly*** discouraged
- Constant: `const n = 42;`
  - Constant cannot be reassigned or redeclared
- Before ES6 `var` was used instead of `let`
  - More on `var` later

Function Declaration

```
function func_name(parameter1, parameter2,...)
{
    ... function body ...
    return value;
}
```

- Note that parameter/return types are not explicitly declared

Types

- JavaScript is a dynamically-typed language

```
let a = 10; // a is number type
a = "good"; // a is string type now
```

- `typeof` operator returns the current type of the variable

```
a = "good";
typeof a;
```

  - But this is not 100% true. More on this later
- Types are either *primitive type* or *object type*
  - Things start to get ugly from here

Primitive Types

1. `number`
2. `string`
3. `boolean`
4. `bigint`
5. `symbol`
6. (`null`)
7. (`undefined`)

`number` Type

- Numbers are represented as ***64-bit floating point number***
  - ***No integer numbers in JavaScript!***
- Bitwise operators (`&`, `|`, `^`, `>>`, `<<`) convert a number to a 32-bi
  - Truncate subdecimal digits if needed
- `NaN` and `Infinity` are valid numbers
- `bigint` (64-bit integer) was added to ES2020 as a primitive t
  - Add `n` behind the number. eg) 12n
  - bigint is ***not*** a number type
    - Cannot mix `number` with `bigint`. eg) `12 + 12n` not allowed

`boolean` Type

- `true` or `false`
- "falsy" values from other data types?
  - 0, `NaN`, "", `null`, `undefined`
  - Empty array `[]` or object `{}` are ***NOT*** falsy values (more on these late

`string` Type

- Single or double quotes: `'John'` or `"John"`
- String is *immutable*
  - String manipulation methods create a new string
- `length` property returns the length of the string
- Many useful string functions exist: `charAt()`, `substring()`, `i`

  ...

```
let a = "abcdef";
b = a.substring(1, 4); // b = "bcd"
```

String Template Literal

```
let s = `this is template literal
that can include ${expression}!`;
```

- Enclosed in backticks: `` `...` ``
- `${expression}` is evaluated and replaced: `${1+2}` → 3
- Can span over multiple lines
- To include special characters, `` ` ``, `$`, `{`, `}`, escape them like `\$`

`undefined` and `null` Type

- `undefined`: type of the value `undefined`
  - "uninitialized"
    - "default value" of a variable before initialization
    - "default value" of a function parameter if not passed by caller
    - return value from a funtion if nothing is explicitly returned
- `null`: type of the value `null`
  - "absence of object": return `null` if no object can be returned
  - `typeof null` returns `object` (!)
- `undefined` and `null` are often interchangeably used, but the different in principle

```
undefined == null;  // true
undefined === null; // false
```

`symbol` Type

- `symbol` type is mainly used to create a unique identifier
- Example

```
let Sym1 = Symbol("Sym");
let Sym2 = Symbol("Sym");

console.log(Sym1 == Sym2); // returns "false"
// Symbols are guaranteed to be unique

console.log(Sym1.description) // returns "Sym"
```

Type Conversion

- `number`s and `string` are automatically type converted to eac
  - But as usual, some surprises…
  - `"3"*"4"=12`
  - `1+"2"="12"`
- For explicit type conversion, use `Number()`, `String()`, `Boole` `parseFloat()`, `parseInt()`, …

Object Type (1)

- All non-primitive types in JavaScript are *object type*
- *Object*: data with a set of "properties"

```
let o = { x: 1, y: "good" };
let c = o.x + o["y"];
```

  - `o["x"]` is identical to `o.x`
  - Objects are essentially an "associative array" or a "dictionary."

Object Type (2)

- Object can be nested

```
let o = { x: 1, y: 2, z:{ x: 3, y: 4 } };
```

- Properties can be dynamically added, removed, and listed

```
let o = new Object();
o.x = 10;
o.y = 30;
delete o.x;
Object.keys(o);
```

Object Type (3)

- Object assignment is *by copying the reference*, not by copying whole object

```
let o = { x: 10, y: 20 };
let p = o;
o.x = 30;
console.log(p.x);
```

- Object comparison is *by reference* not by value

```
let o = { x: 10 };
let p = { x: 10 };
console.log(o == p);
```

Array (1)

- Array is an object with integer-indexed items
  - There is no separate "array type"
- Created with `new Array()`, or `[ 1, 2, 3 ]`
- `length` property returns the size of the array
  - Can be used to resize array by setting its value

```
let a = new Array(1, 2, 3);
let b = [1, 2];
console.log(a.length);
```

Array (2)

- Array can be sparse and its element types may be heterogen

```
let a = new Array();
a[0] = 3;
a[2] = "string";
let b = [1, "good", , [2, 3] ];
console.log(a.length);
```

  - Size of an array automatically increases whenever needed
  - Elements with no initial value are set to `undefined`

Regular Expression

- RegExp is a special object that describes a pattern to search f string

```
let r = /a?b*c/;
```

- Can be used in the following functions
  - String: search(), match(), replace(), split()
  - RegExp: exec(), test()
- Examples

```
/ABC/.test(str);          // true if str has substr ABC
/ABC/i.test(str);         // i ignores case
/[Aa]B[C-E]/.test(str);
'123abbbc'.search(/ab*c/); // 3 (position of 'a')
'12e34'.search(/[^\d]/);   // 2 [^x]: except x, \d: digit
```

Exception Handling

```
try {
    throw new Error("I am thrown off!");
} catch (err) {
    console.log(err.message);
} finally {
    console.log("Finally, I am here");
}
```

- Any value/object can be `thrown`
  - But `Error` object is most common because stacktrace is available: `E`

JavaScript Object Notation (JSON)

- The standard syntax to represent literal objects in JavaScript

```
[{ "x": 3, "y": "Good" }, { "x": 4, "y": "Bad" }]
```

- In JSON,
  - Object property names *require* double quotes
  - Strings need *double quotes*, not single quotes
  - JSON values cannot be functions or undefined

JSON: Data Exchange

- One of the two most popular Web data exchange format
  - `JSON.stringify(obj)`
    - JavaScript object → JSON string
  - `JSON.parse(str)`
    - JSON string → JavaScript object

```
let x = '[{ "x": 3, "y": "Good" }, { "x": 4, "y": "Bad" }]';
let o = JSON.parse(x);
let n = o[0].x + o[1].x;
console.log(n);
```

References

- Javascript: The Definitive Guide by David Flanagan
  - Strongly recommended if you plan to code in JavaScript extensively
- ECMAScript: ECMA 262
- JSON: ECMA 404