

JS Basics - I

@toshita_18




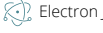
What?

programming language, high-level, object-oriented, multi-paradigm

Role

Html: content
Css: presentation
JavaScript: interaction

Use

Frontend: Dynamic effects 
Backend: Server logic 
Native mobile apps 
Native desktop apps 

Versions

ES5, ES6, ES7, ES8.....ES11
ES: ECMAScript (JS standardisation)
Updates every year

Linking JS

Internally:

```
<head>
  <script> ...JS code....</script>
</head>
```

Only advantage is one less page to load

Externally: (Best practice)

In a separate file (`script.js`), then link

```
<body>
  ...
  <script src='script.js'></script>
</body>
```

Variables

Variable: Container to hold *value* and is given a *label*

Value: Smallest piece of information

Declaring a variable:

```
let variable_label = value;
```

Naming conventions:

Allowed:

variablesMustBeDescriptive
camelCase Uppercase snake_case
underscore_ dollar\$
letters_123Numbers
PI (all uppercase for constants)
Not allowed:
1_cannotStartWithNumbers
ampersand&OtherSymbols
function | let | new (reserved keyword)

Data types (Primitive)

Values are either objects or have primitive types (*Everything is an object in JS except primitive values*)

Primitive data types:

Number String BigInt

null Boolean undefined

Dynamic typing: Do not have to specify the above data types when declaring a variable

Comments

```
// This is a comment and is ignored
/* This is
a multiline
comment */
```

let, const & var for declaration

let - for undefined or temporary values

const - for variables holding permanent data

var - old way, use let instead

Using above keywords will ensure that variables are declared in *current scope* and not *globally*

Basic operators

Operators - to combine & transform values

Arithmetic

plus(+) multiply(*) divide(/) minus(-) exponent(**)
concatenation(+)

Assignment

equal(=) += -= *= /= increment(++), decrement(--)

Comparison

greater than(>) greater than equal to(>=)
less than(<) less than equal to(<=)

typeof

to determine type of any value

```
console.log(typeof true);
```

Bug in JS: `typeof null` returns object but null is both a variable's value and its type

```
console.log(typeof null);
```

String and template literals

```
const str = "With double quotes"
const str2 = 'With single quotes'
const concat = str + str2
const age = 10
console.log(`I'm "${age}" years old`);
```

Template literals (using back ticks `)

```
console.log(`I'm ${age} years old`);
const multi = `this is a
                multiline string`
```

Back ticks can be also used for regular strings

Type conversion & type coercion

Type coercion: JS converts type implicitly `'12'+2=122`, `'123'-'10'=113`, `'23'*2=46`

Here + converts num to string but -,*,/ converts string to num

Type conversion: Coder converts type explicitly
`Number('12')+2=14`, `String(2)`, `Boolean('false');`

Truthy & falsy values

falsy values - values that returns false when converted to boolean via `Boolean()`

They are: false | 0 | '' | NaN | null | undefined

Truthy values - values returning true

Equality operators (== vs ===)

`==` Loose equality operator: Checks whether two values are equal(not their type)

```
20 == 20 → true
```

```
20 == '20' → true
```

`===` Strict equality operator: Checks whether two values are exactly equal(with their types)

```
20 === 20 → true
```

```
20 === '20' → false
```

Different operators (!= vs !==)

Similar to equality operators but instead checks whether 2 values are different from each other

`!=` Loose different operator

`!==` Strict different operator

Use only strict equality and strict different operators for all comparisons

Boolean logic & logical operators

`X && Y` → Returns true only when both are true

`X || Y` → Returns true when either is true

`!X` → true becomes false and vice versa