Front-end <Web /> Development

Lesson 8: JS (Part 2)

University of Tehran

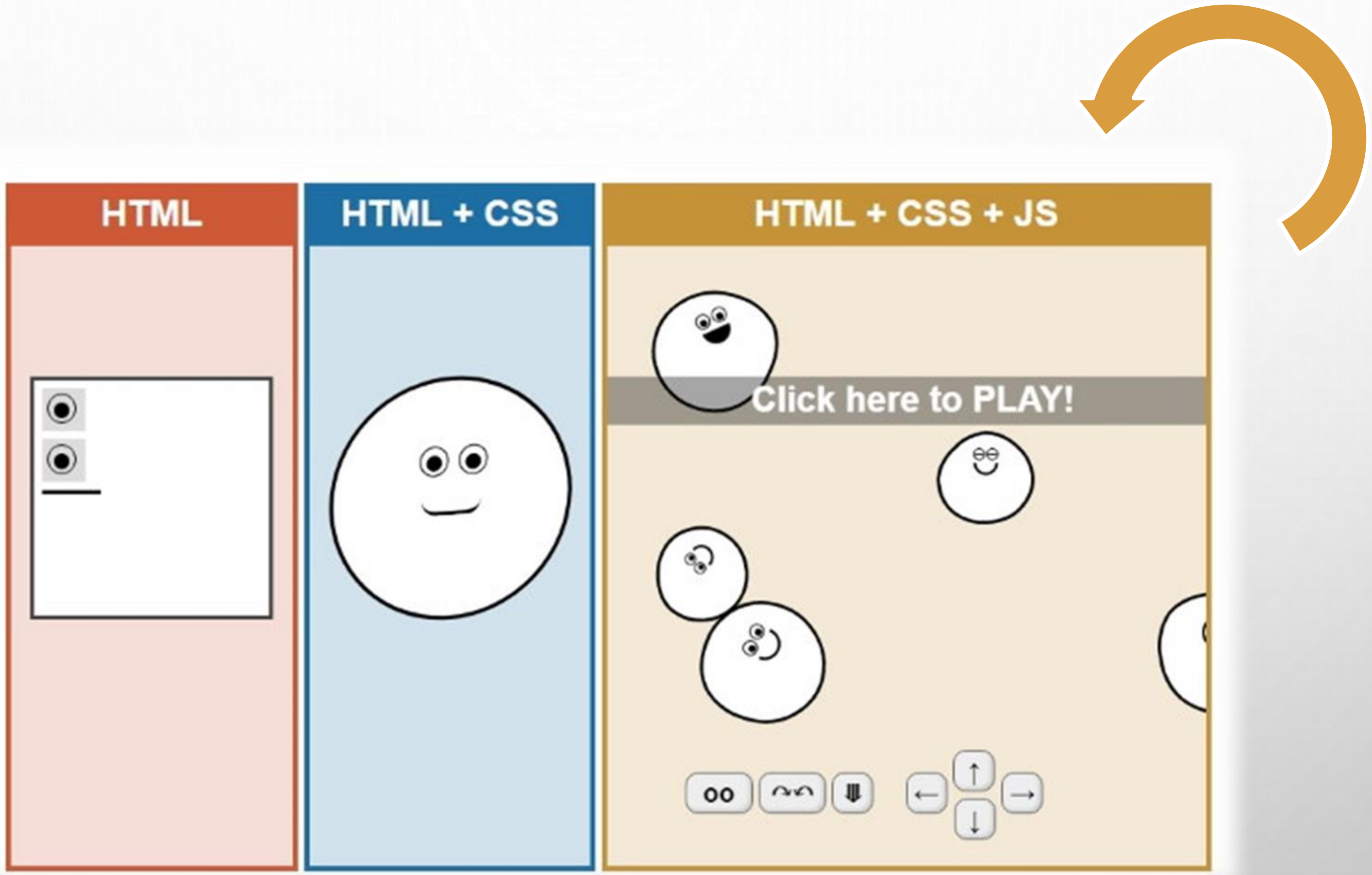
ACM
Summer of Code 2024

Lecturer:
Misagh Mohaghegh

A Short Review…

- JavaScript and its version history
- Adding JS to HTML (end of body & head defer)
- Node.js, TypeScript, Babel
- JS variable types (number, string, boolean, object, array)
- JS variable declarations (var, let, const)
- Equality vs strict equality (== vs ===)
- Function declaration vs function expression
- Arrow function
- Programming basics (if, ternary, switch, while, for)
- Basic DOM manipulation (getElementById, querySelector,
- innerHTML, style, classList, setAttribute)

Continue…



JavaScript

String

Types

JavaScript has 7 primitive types:

- String ("test")
- Number (12.2)
- Boolean (true)
- BigInt (240n)
- Symbol (Symbol("st"))
- Null (null)
- Undefined (undefined)

The rest are object values:

- Object
- Array
- Function
- Map & Set
- Date
- Regex
- JSON

Strings

Strings are put between quotes (double or single):

```
let str = "He's not here";
let str2 = 'Is this the "great" tower?';
```

We can get a string's length using the length property:

```
str2.length == 26
```

To put quotations in a string, we have to escape them using the backslash character:

```
let str = "A double \"quote\" inside";
let str2 = "Escaping backslash: \\";
```

Strings

New lines (\n):

```
let str = "A new\nline"; // A new
                         // line
```

String concatenation:

```
let str = "Hello " + "World!";
let str2 = "He said: " + str;
```

Template Strings

Template strings were introduced in ES6.

They use backticks (the character next to 1 in keyboard):

```
let str = `A template string, "A", 'B'`;
```

They allow multi-line strings:

```
let str = "Hello " +
          "World!";

let str2 = `Hello
there,
World!`;
```

String Interpolation

String interpolations allows easy inclusion of variables in strings:

```
let str = `Welcome, ${firstName}!`;
let str = `That will be: ${price * 10}`;
```

We had to concat strings before ES6:

```
let str = 'Welcome, ' + firstName + '!';
```

String Methods

Strings are **immutable**, they cannot be changed once created.

String methods all return a new string and do not modify the original string.

We saw the length **property** [str.length], now we check some **methods** [str.method()].

To access the character at an index, we can use the **[]** operator or the **charAt** method:

```
let str = "Hello";
str.length == 5
str.charAt(0) == "H"
str[1] == "e"
// Do not confuse with arrays
// Strings are immutable
str[1] = "B"; // does not work
```

Substring

To extract a specific part of a string (called a substring), there are 3 methods:

```
let str = "abcde";
str.substr(2, 2) == "cd" // deprecated, do not use substr(start, length)

str.substring(2, 4) == "cd" // substring(start, end), end is not included
str.substring(1) == "bcde"

str.slice(2, 4) == "cd" // same as substring but supports negative index
str.slice(-2) == "de"
str.slice(-4, -1) == "bcd"
```

Text Transform

We can use the following 2 methods to transform text.

In CSS, we saw the **text-transform** property which could change the visual appearance of a text to uppercase, lowercase, or capitalize.

The string methods actually return a new changed string and do not just change the visual appearance.

```
let str = "hey YOU!";
str.toUpperCase() == "HEY YOU!"
str.toLowerCase() == "hey you!"

function capitalize(str) {
    return str[0].toUpperCase() + str.slice(1);
}
capitalize(str) == "Hey YOU!"
```

Trim

Trimming is removing whitespace (space, newline, tab, …) from the start and end of a string.

This action is done quite often so there is a method for it:

```
let str = "   Example  \n ";
str.trim() == "Example"
str.trimStart() == "Example  \n "
str.trimEnd() == "   Example"
```

Padding & Repeat

Padding is adding characters at start or end of a string to make it reach a certain length:

```
let str = "XY";
str.padStart(5) == "   XY"
str.padEnd(5, "a") == "XYaaa"
"1".padStart(2, "0") == "06"
```

Use the repeat method to get a string with copies of the original repeated:

```
let str = "AB";
str.repeat(3) == "ABABAB"
```

Replace

Replaces the first instance of a string with another (and returns a new string):

```
let str = "This is Mark Paul & Kent Paul";
str.replace("Paul", "AB") == "This is Mark AB & Kent Paul"
str.replace("paul", "AB") == "This is Mark Paul & Kent Paul"
// Replace string is case-sensitive.
// Regex should be used if we want a case-insensitive replace:
str.replace(/paul/i, "AB") == "This is Mark AB & Kent Paul"
// To replace all instances:
str.replace(/Paul/g, "AB") == "This is Mark AB & Kent AB"

// ECMAScript 2021:
str.replaceAll("Paul", "AB") == "This is Mark AB & Kent AB"
```

Split

The split method splits our string into multiple parts in an array:

```
let str = "Example string here";
str.split(" ") == ['Example', 'string', 'here']
str.split() == ['Example string here']
"a,b,c".split(",") == ['a', 'b', 'c']

// Actually, the == is never true because JS objects are never equal unless they are
// the exact same object reference:
const a = [1, 2];
const b = [1, 2];
a == b // false
const c = a;
a == c // true
```

Search

Use the following methods to search the string for another string and get the starting index of it:

```
let str = "This is a text here text";
str.indexOf('is') == 2 // the first instance of 'is' is on index 2
str.indexOf(' is ') == 4
str.indexOf('what') == -1 // -1 is returned when nothing is found
str.indexOf('text') == 10
str.lastIndexOf('text') == 20
str.includes('This') == true
str.startsWith('is') == false
str.endsWith('xt') == true
str.search(/regex/) == -1 // the search method takes a regex
```

Number

Numbers

JS numbers can be both integers or decimals.

Their type is 'number' and the Number object is their wrapper.

```
let num = 10;
let num2 = 10.24;
let num3 = 12e3; // 12000
let num4 = 12e-3; // 0.012

// Numbers are accurate up to 15 digits
Number.MAX_SAFE_INTEGER == 9007199254740991

// Decimals are not always accurate (floating point precision error)
0.1 + 0.2 == 0.30000000000000004
```

String and Numbers

The + operator concatenates strings, and sums numbers.

```
let str = "10";
let num = 20;
str + num == "1020"
num + str == "2010"

"10" + "20" == "1020"
1 + 2 + "3" == "33" // left to right evaluation

// other operators (-, *, /, ...) try to turn the string into a number:
"10" * "2" == 20
```

Other Numbers

Invalid operations produce **NaN** (Not a Number):

```
let num = 122 / "what";
num == NaN
"Hi" * 2 == NaN
typeof NaN == 'number'
```

Large numbers are **Infinity**:

```
let num = 10e400;
num == Infinity
typeof Infinity == 'number'
let b = 122 / 0; // division by zero
b == Infinity
```

To String

Use the **toString** method to turn a number into a string:

```
let num = 122;
num.toString() == "122"
num.toString(2) == "1111010" // binary (base 2)
num.toString(16) == "7a" // hex (base 16)
```

From String

There are many ways to turn a string into a number:

```
let str = "10";
let num = +str;
typeof str == 'string'
typeof num == 'number'

+"10" == 10
Number("10") == 10

parseInt("10") == 10
parseInt("10.2") == 10
parseInt("10abc8") == 10
```

BigInt

BigInt is another numeric type that allows numbers higher than 2^{53} - 1 (9007199254740991)

```
let a = 9007199254740991 99n; // notice the 'n' at the end
typeof a == 'bigint'
// Arithmetic between Number and BigInt is not allowed
```

Check if a number is an integer:

```
Number.isInteger(10) == true
Number.isInteger(12.2) == false
Number.isSafeInteger(10) == true // lower than Number.MAX_SAFE_INTEGER
                                 // and higher than Number.MIN_SAFE_INTEGER
```

Array

Arrays

Arrays are an object type and can store multiple items of different types like a list where order matters.

```
let arr = [1, 2, "test"];
typeof arr == 'object'
arr.length == 3
// access elements with the [] operator
arr[0] == 1
arr[arr.length – 1] == "test"

arr[0] = { name: "me" };
```

Looping

There are many ways to loop over all array items:

```
for (let i = 0; i < arr.length; i++) {
    console.log(`Item ${i}: ${arr[i]}`);
}

for (let x of arr) {
    console.log(x);
}
```

```
for (const [i, x] of arr.entries()) {
    console.log(`Item ${i}: ${x}`);
}

arr.forEach((x, i) => {
    console.log(`Item ${i}: ${x}`);
});

arr.forEach(x => console.log(x));
```

Adding New Items

To append an item to the end of the array, use the push method.

```
let x = [1, 2, 3];
x.push(4);
x == [1, 2, 3, 4]
x.pop();
x == [1, 2, 3]

// appending to the start of the array:
x.unshift(0); // [0].concat(x), [0, ...x] (spread operator)
x == [0, 1, 2, 3]
x.shift()
x == [1, 2, 3]
```

Check Array

To check if an object type is actually an array:

```
let x = [1, 2, 3];
typeof x == 'object'

x instanceof Array
Array.isArray(x)
```

To String

To convert an array to a spring representation of its elements, use the **toString** method.

The **join** method can be used to change the element separator.

```
let x = [1, 2, 3];
x.toString() == "1,2,3"
x.join(" ") == "1 2 3"
```

Splice

The splice method can be used to insert or delete elements at a specific index:

```
let x = [1, 2, 3];
x.slice(1) == [2, 3]

// arr.splice(start index, remove count, new items...)
x.splice(1, 0, 45) => [1, 45, 2, 3]
x.splice(1, 1) => [1, 3]
```

Document Object Model

Topics

The following topics were discussed in the class and shown in code:

- The BOM (Browser Object Model) window object
- The DOM (Document Object Model) document object
- Elements vs Nodes
- DOM navigation (nextElementSibling, previousElementSibling, parentElement, children)
- Node navigation (nextSibling, previousSibling, parentNode, childNodes)
- DOM manipulation (createElement, appendChild, removeChild)
- Events (addEventListener, removeEventListener)
- Event bubbling and capturing

Thank you for your attention.