

CMSC335

Web Application Development with JavaScript



JavaScript3

Department of Computer Science
University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

Array Methods

- **indexOf** - returns position of element in array
- **join** - returns string with all elements in the array
- **pop** - removes & returns last element
- **push** - adds to the end (returns length)
- **reverse** - reverses the array
- **shift** - removes & returns first element
- **unshift** - adds new element to the beginning
- **splice** - adds and/or elements from array (modifies original array); returns array
- **slice** - copies (shallow copy) elements from an array (does not modify original array); returns array
- **concat** - returns copy of joined arrays
- **fill** - fill elements of an array
- **Example:** ArrayMethods.html, ArraySlice.html (after opening console, execute script again to see array in table format)

typeof and instanceof operator

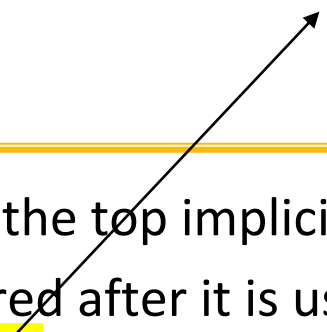
- **typeof**
 - Returns “object” for all reference types
- **instanceof** operator
 - Returns **true** if a value is an instance of the specified type and false otherwise
 - **instanceof** can identify **inherited** types
- **Note: every object is an instance of Object**
- Checking if an object is an array or not
 - Although **instanceof** can identify arrays, use **Array.isArray()** instead, as **instanceof** will not work in all cases
- **Example:** InstanceOf.html

let and const

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo");
elem.innerHTML = x;

var x; // Declare x
```



- **var** declarations are moved to the top implicitly
 - So, a variable can be declared after it is used (e.g., assigning a value to it) -- called **Hoisting**
 - Only declaration will be hoisted, not initialized value
- **let** replaces **var** for variable declarations and provides block scoping
 - **Example:** BlockScope.html
 - **Does not allow hoisting!**
- **const** allows you to declare a constant variable that has block scope
 - **Example:** Const.html
- No block scope and no const before ES6
 - **Example:** NoBlockScope.html

for...of Statement

- **for...of** - Creates a loop iterating over iterable objects. Works on objects that have a method that returns an iterator
- Creates a loop iterating over iterable objects, including:
 - built-in String
 - Array
 - Array-like objects
 - TypedArray
 - Map
 - Set
 - User-defined iterables
- **Example:** ForOf.html

Template Literals

- String literals that allow embedded expressions
- Can replace placeholders in text with variables or exprs
- Defined using the **backtick** character (character below ~ in keyboard)
 - **``embedded string expression``**
 - Placeholders identified with **`${expression}`**
 - » **`${x}, ${x * y}`**
 - To escape a back-tick in a template literal, use backslash before the back-tick
- **Simpler for multi-line strings**
 - Space matters
 - **Example:**

```
const string = `Hello
  terps!`
```
- **Example:** TemplateLiteral.html

Null and undefined

- **null**
 - A value indicating no value (nothing)
 - Type: “object”
- **undefined**
 - Value associated with **uninitialized** variables
 - Type: “undefined”
 - **Example:**
 - » let x; /* In a function */
 - » **undefined** is returned by a function when no explicit value is returned (**IMPORTANT case**) – You forgot a return
 - » Value associated with object properties that do not exist
- **==** considers **null** and **undefined** equal
- **===** considers **null** and **undefined** different

Truthy vs Falsy

- A falsy value is:
 - A value that is considered as **false** in a boolean context
 - Falsy values are:
 - » false
 - » 0
 - » ""
 - » null
 - » undefined
 - » NaN
- A **truthy** value is:
 - A value that is considered as **true** in a boolean context
 - All values are truthy unless they are defined as falsy
- **Example:** TruthyFalsy.html

NaN

- NaN
 - Generated when arithmetic operations result in undefined or unrepresentable value
 - Generated when attempting to coerce to numeric values non-numeric values for which no primitive numeric value is available
- **Global isNaN function** (i.e., `window.isNaN()`)
 - Determines (returns true or false) whether an argument is not a number.
It attempts to coerce the argument to a number
 - Interesting cases:
 - » `isNaN({})`, `isNaN([])`, `isNaN([389])`, `isNaN(true)`, and `isNaN(false)`
- **Number.isNaN()**
 - More robust version of `isNaN()`
 - **Compare a value to NaN only if the value is a Number-type value**
 - Return false for all other cases

NaN

- The following comparisons return false

NaN == NaN

NaN === NaN

- **Remember**
 - **! isNaN()** allow us to determine whether an expression is a number
 - isNaN(20) : false
 - You may want to write a function called **isNumber** that returns ! isNaN(x)
- **Example:** NaN.html

Numeric Values

- **Infinity** is a global property
 - Default: `Number.POSITIVE_INFINITY`
- **isFinite()**
 - Returns false if argument is NaN, positive/negative infinity;
 - Otherwise, it returns true.
- **isFinite() vs. Number.isFinite()**
 - `isFinite()` function converts the value to a Number, then tests it
 - `Number.isFinite()` does not convert the values to a Number, and will return false for any value that is not of the type Number
- **Example:** `NumericValues.html`

Sorting

- **Example:** `Sorting.html`