

JavaScript Reference

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

JavaScript Console

- Allow us to view JavaScript errors
- **console** object functions
 - log → General message
 - info → Informational message
 - error → Error message
 - warn → Warning message
- In Chrome
 - Tools → JavaScript console
 - Notice that different icons are used
 - You can practice JavaScript by typing code at the console
- **Example:** consoleEx.html

Built-in Types

- **Object** – generic object
- **Array** – list of values (numerically indexed)
- **Function**
- **Error** – runtime error
- **Date** – date / time
- **RegExp** – regular expression
- Many of built-in type have a **literal form** that enables you to define a value without explicitly creating an object (using **new**)
- The typical function definition is based on a literal form

Primitive Wrapper Types

- Three types: Boolean, String, and Number
- Primitive wrapper types simplify working with primitives
- Wrapper types are automatically created when needed
- **Example:** Wrapper.html

Global Object

- ECMAScript defines a global object
 - In JavaScript **window** implements the global object
 - All functions and variables defined globally become part of the global object
- Some **functions** that are part of the Global object
 - isNaN()
 - parseFloat()
 - parseInt()
 - eval()
 - isFinite()
 - decodeURI()

Global Object

- Some **properties** that are part of the Global object
 - NaN
 - undefined
 - Object → Constructor for Object
 - Array → Constructor for Array
 - Function → Constructor for Function
 - Number → Constructor for Number
 - String → Construct or for String
 - Date → Constructor
 - Error → Constructor
 - RegExp → Constructor
- ECMAScript also defines the Math object

Functions

- Functions are objects
- The name of the function is a reference value
- Functions can be passed and returned from other functions
- Functions can be defined inside of other functions
- Function Declaration

```
function name (<comma-separated list of parameters>) {  
    statements  
}
```

- Functions are invoked by using the () operator
- We don't use **var** for parameters (e.g. function print(x, y))
- Parameters are passed by value
- There is no main function like in other languages
- Returning values via return

Functions

- Three approaches to define functions
 - **Function declaration**
 - Read and available before any code is executed
 - When a function is hoisted it is internally moved to the beginning of the current scope
 - Function declaration hoisting allows calling the function after its declaration
 - Functions can appear in any other
 - **Function expression**
 - **Using Function constructor**
- **Example:** DefiningFunctions.html
- **Example:** FunctionsAsData*.html
- Function overloading is not possible
 - Second function redefines the first one

One-Dimensional Arrays

- **Array** → Collection of values that can be treated as a unit or individually
`var a = new Array(4);`
- **Indexing** → We access an element using []
 - First element associated with index 0 (e.g., a[0])
- An element of an array can be of any type and an array can hold different types of elements
- The length property represents the length of the array (e.g., a.length)
- We can print the contents of an array by using alert

Definition of One-Dim Arrays

- Using **literal form**
 - Comma separated list of elements within square brackets

```
var a = [2, 3, 5];  
var b = []; // empty array
```

- Using **Array constructor**

```
var c = new Array();  
var e = new Array(4);           // defines array of size 4
```

- **Example:** ArraysOneDim.html
- **Example:** ArraysLengthProp.html

Two-Dimensional Arrays

- JavaScript does not support actual two-dimensional arrays
- You can simulate two-dimensional arrays by using an array of arrays
- About two-dimensional arrays
 - You can pass them and return them from functions like one-dimensional arrays
 - Any modifications in the function will be permanent
 - You can have ragged arrays
- **Example:** ArraysTwoDim.html

String Methods

- Comparison based on < and >
- **concat** – returns a new string representing concatenation of strings
- **includes** – determines whether one string is found within another
- **startsWith** – whether string begins with characters from another string
- **endsWith** – whether string ends with characters of another string
- **indexOf** – index of first character in string (or -1 if not found)
- **lastIndexOf** – index of last occurrence of character in the string (or -1 if not found)
- **repeat** – returns string repeated **n** times
- **splice** – extracts section of string
- **split** – splits a string into array of strings
- **toLowerCase**
- **toUpperCase**
- **trim** – trims whitespaces
- **Example:** StringMethods.html
- Reference:
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Getting String Characters

- The function `charAt` or `[]` allows us to retrieve the character associated with a particular index position in a string. Access is similar to array indexing (first character at 0).
- Example
 - `var x = "Wednesday";`
 - `var secondCharacter = x.charAt(1); // variable has "e";`
 - `var lengthOfString = x.length; // variable has 9`
- **Example:** `CharAt.html`

Array Methods

- **fill** - fill elements of an array
- **concat** - returns copy of joined arrays
- **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 removes elements from an array
- **Example:** ArrayMethods.html
- **Example:** Sorting.html
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

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
- Regarding arrays
 - Although **instanceof** can identify arrays, use **Array.isArray()** instead as **instanceof** will not work in all cases (e.g., when a array is passed from one frame to another).
- **Example:** InstanceOf.html

let/const/

- No block scope so far (E6 introduces it)
 - **Example:** NoBlockScope.html
- **let** replaces var for variable declarations and provides block scoping
 - **Example:** BlockScope.html
 - let is the new var
- **const** allows you to declare a constant variable that has block scope
 - **Example:** const.html

for of

- Works on objects that have a method that returns an iterator
- **Example:** ForOf.html

Template Literals

- Allows you to replace placeholders in text
 - Defined using the backtick character
 - Placeholders identified with `${}`
 - **Example:** `TemplateLiteral.html`

Random Values

- Example: RandomValues.html

Null and undefined

- **null** → indicates no value (nothing)
- **undefined**
 - Value associated with uninitialized variables
 - `var x; //` in a function
 - Value returned by function when no explicit value is returned (IMPORTANT case)
 - Value associated with object properties that do not exist
- `==` considers **null** and **undefined** equal
- `===` considers **null** and **undefined** different

NaN

- NaN
 - Generated when arithmetic operations result in undefined or unrepresentable value
 - Generated when attempting to coerce to a numeric value a non-numeric value
- Global **isNaN** function → determines (returns true or false) whether an argument is not a number. **It attempts to convert the argument to a number**
- `Number.isNaN()` → More robust version of `isNaN()`

NaN

- The following comparisons return false
 NaN == NaN, NaN === NaN
- Remember → **!**isNaN() allow us to determine whether an expression is a number
 - Notice: isNaN(20) → False
 - You may want to write a function call isNumber that returns !isNaN(x)
- **Example:** NaN.html

Numeric Values

- **Example:** NumericValues.html
- **Infinity** is a global property
- `isFinite()` – returns false if argument is NaN, positive/negative infinity; otherwise, it returns true.
- `isFinite()` vs. `Number.isFinite()`

About prompt

- Returns null when cancel is selected
- **Example:** Null.html, ValidityCheck.html

Debugging

- **Chrome**

- Select Inspect after loading the script, and **Sources**. This will open the debugger on the rightmost pane. Click on a source line to set a break point.

- **Chrome**

- You can add in your code the statement **debugger**; which will invoke the debugger when you run the script. To access the debugger, open the script, select Inspect, and run. You will see the debugger on the rightmost pane.

- **Firefox Debugger**

- Open script you want to debug with Firefox
- You will find the debugger at Tools→Web Developer→Debugger
- To set a breakpoint click on the line number
 - Right-click on a line number provides additional options
- Reload page to run script
- Typical stop, set over and step into option can be found on stop of “Sources” and “Call Stack” tabs
- Call Stack – allows you to change stack frame

- <http://www.cs.umd.edu/~nelson/classes/utilities/JavaScript/JavaScriptDebugging/>