



# Basics of JavaScript

Tessema Mengistu (Ph.D.)

Department of Computer Science

Virginia Tech

[Mengistu@vt.edu](mailto:Mengistu@vt.edu)



# Outline

- Overview of JavaScript
- Primitives, Operations, and Expressions
- Control Statements
- Arrays
- Functions
- Events



# Overview of JavaScript

- Originally developed by Brendan Eich at Netscape as LiveScript
  - Joint venture with Sun Microsystems later in 1995
- Standardized as ECMA-262 by the **European Computer Manufacturers Association**
  - Current version - ECMAScript 2023 (ES14)
- JavaScript executes in a browser
  - Broadly supported by all browsers: Chrome, Firefox, Edge, etc.



# Uses of JavaScript

- JavaScript can work with forms
- JavaScript can interact with the internal model of the web page – DOM (Document Object Model)
- JavaScript is used to provide more complex user interface than plain forms with HTML/CSS can provide



# General Syntactic Characteristics

- All JavaScript scripts are embedded in HTML documents
  - Either directly, as in

```
<script type = "text/javascript">  
    // JavaScript script  
</script>
```

- Or indirectly, as a file specified in the `src` attribute of `<script>`, as in

```
<script type = "text/javascript"  
    src = "myScript.js">  
</script>
```



# General Syntactic Characteristics (continued)

- Language Basics:
  - Identifier
    - Begin with a letter or underscore, followed by any number of letters, underscores, and digits
    - Case sensitive
  - Comments
    - Both `//` and `/* ... */`
  - Statements can be terminated with a semicolon
    - The interpreter will insert the semicolon if missing at the end of a line
    - But this can lead to a real problem
      - Example

```
return
    x;
```

# Variables

- JavaScript is dynamically typed, that is, variables do not have declared types and do not have to be declared.
  - A variable can hold different types of values at different times during program execution
- A variable is explicitly declared using the keyword **let** or **const**

```
let counter = 10;  
const pi = 3.14159265,  
counter = "Elway",  
stop_flag = true;
```



# Datatypes

- Five primitive types
  - number
  - string
  - boolean
  - Undefined
  - null
- Reference types
  - Array
  - Objects





# Numeric and String Literals

- Numeric values are represented internally as double-precision, floating-point values
  - Numeric literals can be either integer or float
  - Float values may have a decimal and/or exponent
- A string literal is delimited by either single or double quotes
  - There is no difference between single and double quotes
  - The empty string `''` or `""` has no characters

# Template Literals

- Enclosed by backticks ( ` ` ) instead of double or single quotes
- Allows multi-line string and **string interpolation** with embedded expressions
  - Placeholders are indicated by the dollar sign and curly braces
    - `${expression}`
    - Example:

```
const bookName = "Just JavaScript";
console.log(`The      book      name      is
${bookName} `);
```



## Other Primitive Types

- **boolean**
  - Two values: **true** and **false**
- **null**
  - As a primitive value, null, simply means “no value” (no content).
  - Using undeclared & unassigned variable typically results in the null value
    - Usually causes an error.
- **undefined**
  - The undefined primitive value indicates the nonexistence status that a variable has not been assigned a value
  - However, undefined is not a reserved word (just indicates such a conception).



# JavaScript Objects

- Objects are collections of properties
- Properties are either data properties or method properties
- Data properties are either primitive values or references to other objects
- The special Object object is the ancestor of all objects in a JavaScript program
  - This Object itself has no data properties, but several method properties



# JavaScript Objects

- The new expression is used to create an object
  - This includes a call to a constructor
  - The `new` operator creates a blank object, the constructor creates and initializes all properties of the object
- Objects can also be created using **object literal syntax**
  - Object in `{ }` with key:value pairs
- Properties of an object are accessed using a dot notation: `object.property` or array notation `object["property"]`
- Properties are not variables, so they are not declared
- The number of properties of an object may vary dynamically in JavaScript



# JavaScript Objects

- Create (using new) an object and add some properties

```
const date= new Date();  
let m = date.getMonth();
```

```
const my_car = {  
    make: "Ford",  
    model:"Edge"  
}  
// change a property  
my_car.make = "Toyota";  
// create using new
```



# JavaScript Objects

- Reference copy

```
const my_car = {  
    make: "Ford",  
    model: "Edge"  
}  
  
const my_otherCar = my_car;  
my_otherCar.model = "Ranger";  
console.log(my_car.model); //Ranger
```

- Spread operator

- {...}

```
const my_otherCar = {...my_car};  
my_otherCar.model = "Ranger";  
console.log(my_car.model); //Edge
```

```
const my_otherCar = {...my_car, year: 2023};
```

# Arrays

- An Array is a list of elements indexed by a numerical value
  - JavaScript treats arrays as an Object
- Array indexes in JavaScript begin at 0
- Dynamic size: array size can be modified even after created.





# Arrays

- Arrays can be created using the new Array operation
  - new Array with one parameter creates an empty array of the specified number of elements

```
new Array(10)
```

- new Array with two or more parameters creates an array with the specified parameters as elements

```
new Array(10, 20)
```

- Literal arrays can be specified using square brackets to include a list of elements

```
let alist = [100, 101, 102, 103];
```

- The elements in an array can be of different types



# The “for-of” and “for-in” Loop

- Specifically for JavaScript
- Syntax

```
for (identifier of array)  
    statement or compound statement
```

```
for (identifier in array)  
    statement or compound statement
```

- The loop lets the identifier take on each element in in the array
- Printing the elements of an array:

```
let alist = [100, 101, 102, 103];  
for (let elem of alist)  
    console.log("Id: " + elem);  
for (let elem in alist)  
    console.log("Id: " + aList[elem]);
```



## Characteristics of `Array` Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (Array created with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array



# Array Methods

- `length` // field, not a method
- `join(separator)` // *into a string*
- `reverse()`
- `map()`
- `reduce()`
- `sort(sortfunc)`
- `concat (array1, ..., arrayN)`
- `slice(start, end),splice()`
- `push(),unshift()`
- `pop(),shift()`
- ...



## The `String` Wrapper Object

- Only one (data) property: **length**
  - Note this is not a method!
- Character positions in strings begin at index **0**

Method	Parameters	Result
<code>charAt</code>	A number	Returns the character in the String object that is at the specified position
<code>indexOf</code>	One-character string	Returns the position in the String object of the parameter
<code>substring</code>	Two numbers	Returns the substring of the String object from the first parameter position to the second
<code>toLowerCase</code>	None	Converts any uppercase letters in the string to lowercase
<code>toUpperCase</code>	None	Converts any lowercase letters in the string to uppercase

# The Date Object's Methods

toLocaleString	A string of the Date information
getDate	The day of the month
getMonth	The month of the year, as a number in the range of 0 to 11
getDay	The day of the week, as a number in the range of 0 to 6
getFullYear	The year
getTime	The number of milliseconds since January 1, 1970
getHours	The number of the hour, as a number in the range of 0 to 23
getMinutes	The number of the minute, as a number in the range of 0 to 59
getSeconds	The number of the second, as a number in the range of 0 to 59
getMilliseconds	The number of the millisecond, as a number in the range of 0 to 999



# Control Structures

- The *if-then* and *if-then-else* are similar to that in other programming languages

- *switch* statement

- Ternary operator - *?:*

- While

```
while (control expression)
    statement or compound statement
```

- For

```
for (initial expression; control expression; increment expression)
    statement or compound statement
```

- do/while

```
do
    statement or compound statement
while (control expression)
```



# Function

- Function definition syntax

- A function definition consist of a header and a compound statement
- A function header:

*function function-name(optional-formal-parameters)*

- The return statements

- A return statement causes a function to cease execution and to pass control to the caller
- A return statement may include a value which is sent back to the caller
  - This value may be used in an expression by the caller
- A return statement without a value implicitly returns *undefined*





# Functions

- Functions are objects in JavaScript
- Functions may, therefore, be assigned to variables and to object properties
  - Object properties that have function values are methods of the object
- Example

```
function fun() {  
    document.write("This surely is fun! <br/>");  
}  
  
ref_fun = fun; // Now, ref_fun refers to the fun object  
fun(); // A call to fun  
ref_fun(); // Also a call to fun
```

# Arrow Functions

```
const characters =  
  ['J', 'a', 'v', 'a', 'S', 'c',  
   'r', 'i', 't'];
```

```
const changedCharacters =  
  characters.map(  
    function (c) {  
      if (c === c.toLowerCase())  
      {  
        return c.toUpperCase();  
      }  
      else {  
        return c.toLowerCase();  
      }  
    })
```

```
const characters =  
  ['J', 'a', 'v', 'a', 'S', 'c',  
   'r', 'i', 't'];  
  
const changedCharacters =  
  characters.map((c) => {  
    if (c === c.toLowerCase())  
    {  
      return c.toUpperCase();  
    }  
    else {  
      return c.toLowerCase();  
    }  
  });
```

# Async/await

- JavaScript is a single-threaded programming language
  - Only process one line of code at a time

```
console.log("This is sync");  
display();  
function display() {  
    const text = getTxts();  
    console.log(text);  
}  
function getTxts() {  
    return "JavaScript";  
}
```



# Async/await

```
console.log("Learning");  
display();  
async function display() {  
    const what = await get texts();  
    console.log(what);  
}  
function get texts() {  
    return new Promise((resolve, reject) => {  
        setTimeout(() => resolve("React"),  
1000);  
    }));  
}
```



# Async/await

- A `Promise` object
  - Represents a proxy for a value that may become available at a later point in time
  - Three states:
    - `pending` - initially
    - `fulfilled` - the function gets executed successfully
    - `rejected` - the function fails



# Document Object Model – the DOM

- DOM specifications describe an abstract model of a document and its elements
  - Each HTML doc is mapped to a tree structure
  - Elements mapped to **nodes** objects and attributes to properties
  - Methods are the main interfaces
  - Different languages will need to bind the interfaces to their specific implementations
    - In JavaScript, data are represented as properties and operations as methods

# Element Access in JavaScript

- Using:

- *querySelector()*
- *getElementById()*
- *getElementsByClassName(class)*
- *getElementsByTagName(tag)*

- Example

```
let objs= document.getElementsByTagName("div");  
    for(let i=0;i<objs.length;i++)  
        let htmlElement = obj[i];  
        ...
```

# ... the DOM

- Using DOM, embedded JavaScript code can dynamically change the document being displayed in a browser window on a variety of aspects:
  - Elements can be moved or repositioned
  - Style can be changed
  - Visibility can be changed
  - Element contents can be changed
  - ...



# Getting and Setting CSS properties

- JavaScript can access the CSS properties via the style attribute of the DOM object
- E.g.

```
let obj = document.getElementById("banner");  
let color = obj.style.color;
```

```
obj.style.position = "absolute";  
obj.style.top = "100px";
```



# Events

- Actions that get fired inside the browser
  - Most commonly through user interactions
- We can attach an event to a specific element, like a button, or to the entire browser window



# Common Events & Event Attributes of Tags

## *Event*

blur

change

click

focus

load

mousedown

mousemove

mouseout

mouseover

mouseup

select

submit

unload

## *Tag Attribute*

onblur

onchange

onclick

onfocus

onload

onmousedown

onmousemove

onmouseout

onmouseover

onmouseup

onselect

onsubmit

onunload



# References

- W3Schools
- Mozilla Developers
  - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/>