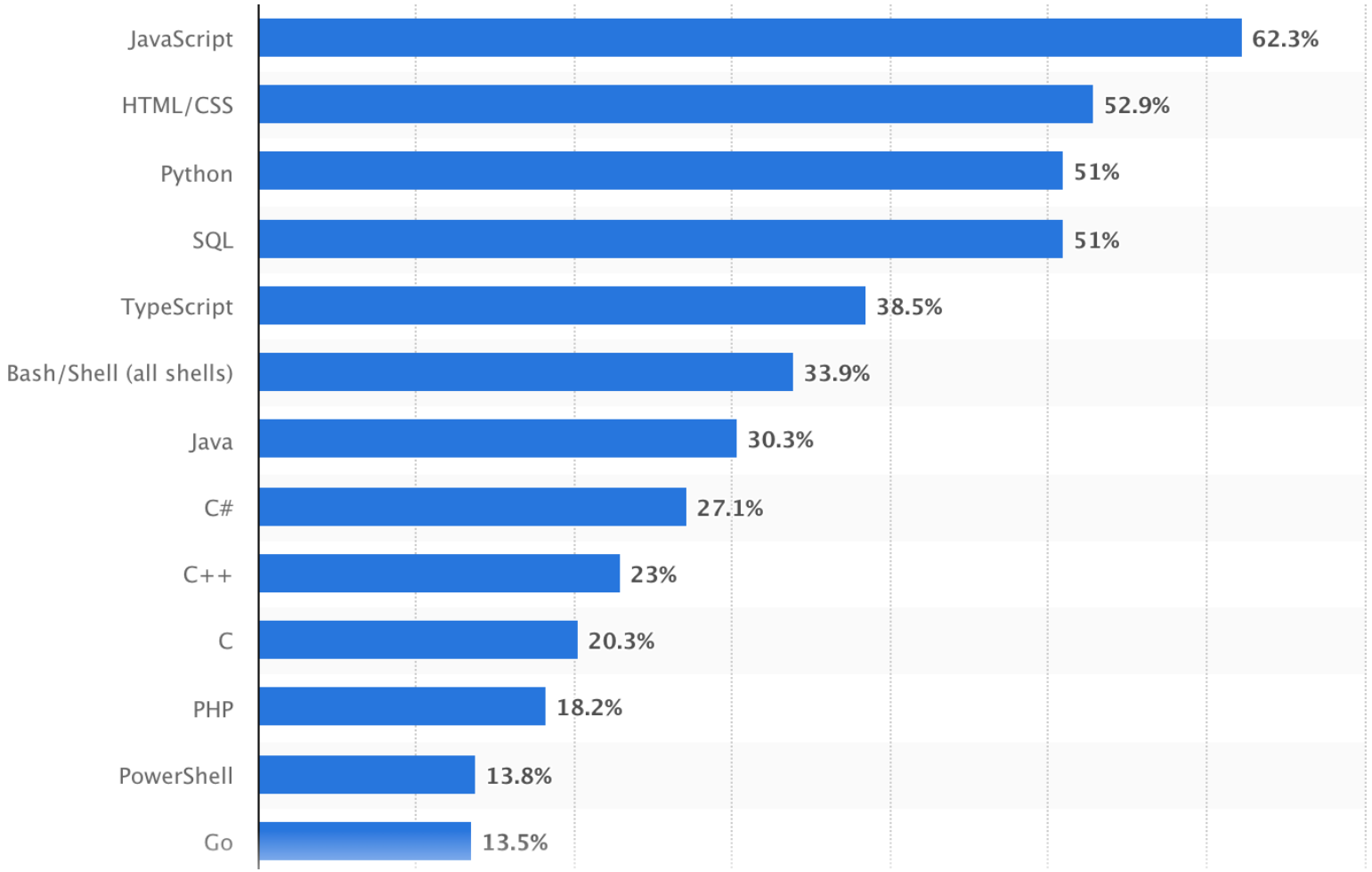
JavaScript

Most used programming languages as of 2024



src: https://www.statista.com/

Content

- Scripts
  - common tasks for client-side scripts

- JavaScript
  - data types & expressions
  - control statements
  - functions & libraries
  - date, document, string, array, user-defined classes

Client-Side Programming

- HTML is good for developing static pages
  - can specify text/image layout, presentation, links, …
  - Web page looks the same each time it is accessed

- Client-side programming
  - programs are written in a separate programming (or scripting) language, e.g., JavaScript,
  - programs are embedded in the HTML of a Web page
  - the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies

- limitations of client-side scripting
  - since script code is embedded in the page, it is viewable to the world
  - for security reasons, scripts are limited in what they can do

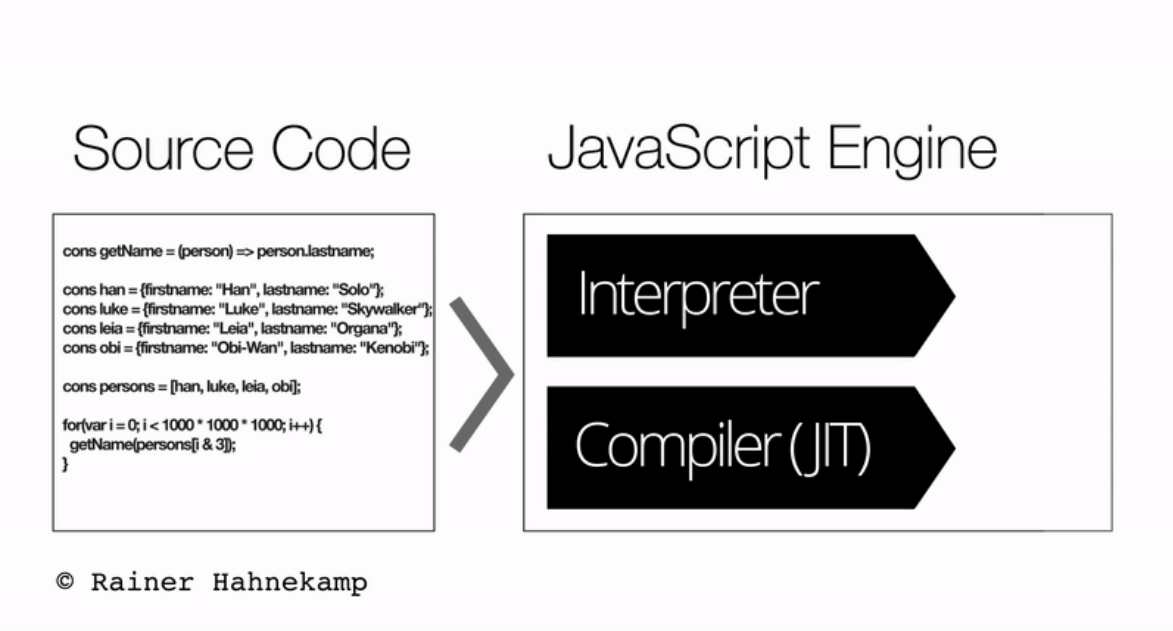
JavaScript/ ECMAScript

Year	ECMA	Browser
1995		Javascript was invented by Brendan Eich
1996		Netscape 2 has Javascript 1.0
1997		Javascript become an ECMA standard (ECMA-262)
1997	ES1	IE4 was the first browser to support ES1
2015	ES6	ECMAScript6 was released
2016	ECMAScript 2016	
2016-2017		Full support for ECMAScript2016 by modern browsers (Chrome, Safari, Firefox, Edge)

- ECMA - European Computer Manufacturers Association
- ECMAScript is a standard for scripting languages including JavaScript, JScript, and ActionScript

JavaScript

- JavaScript is mainly interpreted, but modern JavaScript engines, like V8 (in Google Chrome), SpiderMonkey (in Firefox) use JIT (Just-In-Time compilation) to boost performance.
    - Execute the code using an interpreter
    - Identify frequently used (hot path) code and compile it into opitimzed machine code



JavaScript

- Client-side: JS can run on browsers as a scripting language that allows you to create dynamically updating content, control multimedia, animate images,…
- Server-side: JS can run on server side with the appearance of NodeJS – a Javascript runtime environment.

JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Page</title>
</head>

<body>
  <script type="text/javascript">
   document.write("<p>Hello world!</p>");
   document.write(" <p>How are you?</p> ");
  </script>
  <p>Here is some static text.</p>
</body>

</html>
```

- Use <script> tag to add Javascript code to a page
- document.write
  - displays text in the page
  - text can include HTML tags

Hello world!

How are you?

Here is some static text as well.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_myfirst

JavaScript Data Types & Variables

```
<script>
// Number:
let weight = 7.5;

// String:
let color = "Yellow";

// Booleans
let x = true;

// Object:
const person = {firstName:"John",
lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo"];

// Date object:
const date = new Date("2024-09-25");
</script>
```

- JavaScript has 8 data types
  - String
  - Number
  - Bigint
  - Boolean
  - Undefined: undefined
  - Null: null
  - Symbol
  - Object
- variable names contain letters, digits, an underscores that start with a letter or an underscore, case sensitive

JavaScript Data Types & Variables

- Same variable can be used to hold different data types:

```
<script>
let x;    // Now x is undefined
x = 5;    // Now x is a Number
x = ”A"; // Now x is a String
</script>
```

JavaScript Declaration

- var: function scope or global scope, used in all JS code from 1995 to 2015

```
if (true) {
    var noBlockScope = true;
}
console.log(noBlockScope)
=> true
```

```
function foo_a() {
    var functionScope = true;
}
foo_a()
console.log(functionScope)
=> Uncaught ReferenceError: functionScope is not defined
```

- let: block scope, added to JS in 2015

```
if (true) {
    let functionScope = true;
}
console.log(functionScope)
=> Uncaught ReferenceError: functionScope is not defined
```

- const: same as let, except the user cannot update it
- *Note: use var keyword only in code written for old browsers*

JavaScript Operators & Control Statements

```
<!DOCTYPE html>
<html>
<body>
 <script type="text/javascript">
    const distanceToSun = 93.3e6 * 5280
* 12;
    let thickness = .002;
    let foldCount = 0;
    while (thickness < distanceToSun)
    {
        thickness *= 2;
        foldCount++;
    }
    document.write("Number of folds = "
+foldCount);
  </script>
</body>
</html>
```

Operators & control statements:

+, -, *, /, %, ++, --, …

==, !=, <, >, <=, >=

&&, ||, !,===,!==

if , if-else, switch

while, for, do-while, …

User-Defined Functions

- function definitions are similar to C++/Java, except:
  - no return type for the function (since variables are loosely typed)
  - no variable typing for parameters (since variables are loosely typed)
  - by-value parameter passing only (parameter gets copy of argument)

```
function isPrime(n)
{
  if (n < 2) {
    return false;
  }
  else if (n == 2) {
    return true;
  }
  else {
      for (let i = 2; i <= Math.sqrt(n); i++) {
        if (n % i == 0) {
          return false;
        }
      }
      return true;
  }
}
```

Function Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript">
    function isPrime(n)
    // Assumes: n > 0
    // Returns: true if n is prime
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }
  </script>
</head>
<body>
 <script type="text/javascript">
    let testNum = parseFloat(prompt("Enter a positive integer", "7"));
    if (isPrime(testNum)) {
      document.write(testNum + " <b>is</b> a prime number.");
    }
    else {
      document.write(testNum + " <b>is not</b> a prime number.");
    }
  </script>
</body>
</html>
```

Another Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript">
    function randomInt(low, high)
    // Assumes: low <= high
    // Returns: random integer in range [low..high]
    {
      return Math.floor(Math.random()*(high-low+1)) + low;
    }
  </script>
</head>
<body>
   <script type="text/javascript">
      const roll1 = randomInt(1, 6);
      const roll2 = randomInt(1, 6);
      document.write("<img src='" + roll1 + ".gif'/>");
      document.write("<br/>");
      document.write("<img src='" + roll2 + ".gif'/>");
    </script>
</body>
</html>
```

JavaScript Libraries

Better still: if you define functions that may be useful to many pages, store in a separate library file and load the library when needed load a library using the src attribute in the script tag

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

Library Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript"
    src="random.js">
  </script>
</head>
<body>
   <script type="text/javascript">
      const roll1 = randomInt(1, 6);
      const roll2 = randomInt(1, 6);
      document.write("<img src='" + roll1 + ".gif'/>");
      document.write("<br/>");
      document.write("<img src='" + roll2 + ".gif'/>");
    </script>
</body>
</html>
```

Callback Function

- We can pass functions as parameters to other functions and call them inside the outer function

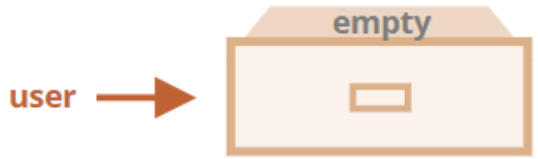
```
function greeting(name) {
  alert(`Hello, ${name}`);
}

function processUserInput(callback) {
  const name = prompt("Please enter your name.");
  callback(name);
}

processUserInput(greeting);
```

```
setTimeout(myFunction, 3000);

function myFunction() {
  document.getElementById("demo").innerHTML = "I love You !!";
}
```

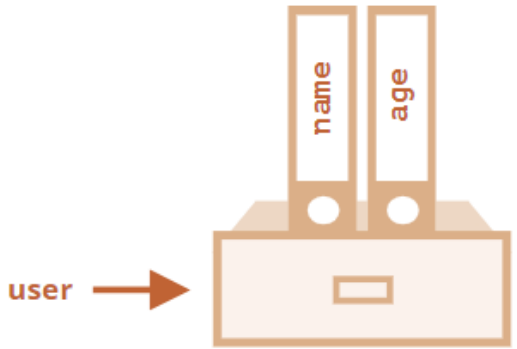
Objects

Objects are used to store keyed collections of various data and more complex entities. An object contains list of properties. A property is a “key:value” pair, where key is a string and value can be anything.

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

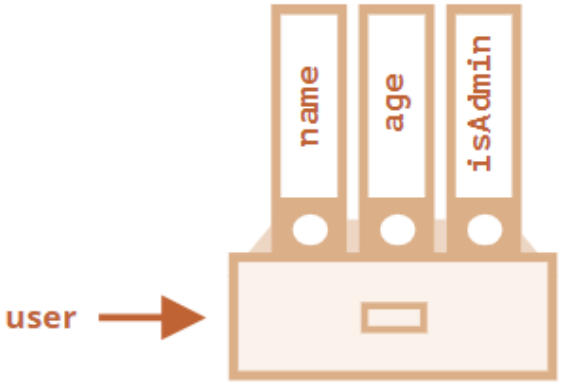



```
let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30        // by key "age" store value 30
};
console.log(user.name) // John
console.log(user.age)  // 30
```

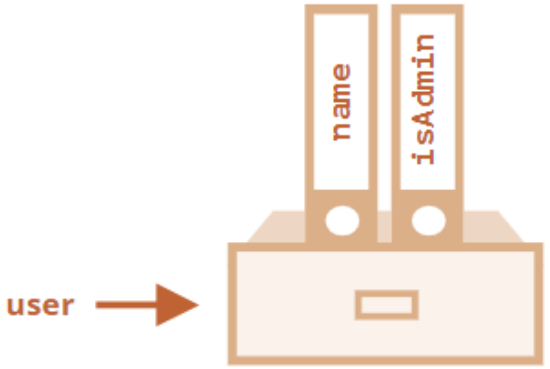


Objects

```
1 user.isAdmin = true;
```

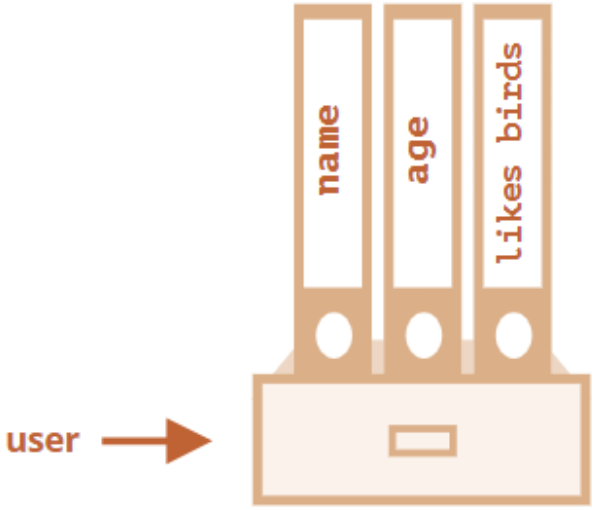



```
1 delete user.age
```



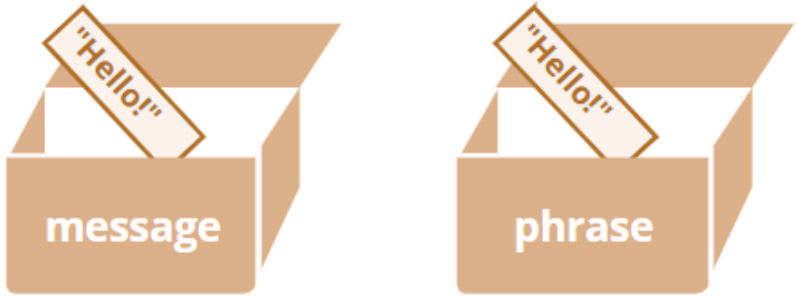
Objects

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // multiword property name must be quoted
};
console.log(user.like birds) // syntax error
console.log(user["like birds"]) //true
```

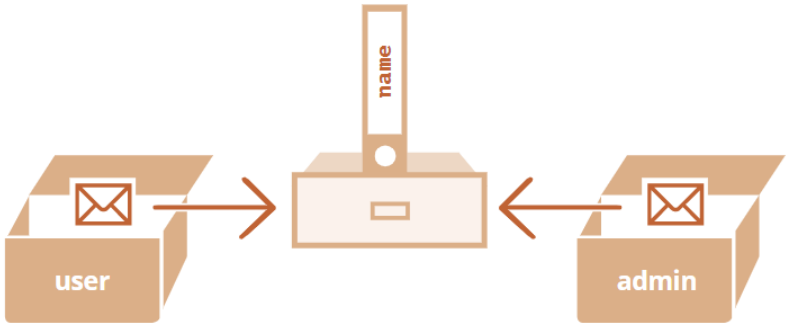


Object references

```
1 let message = "Hello!";
2 let phrase = message;
```

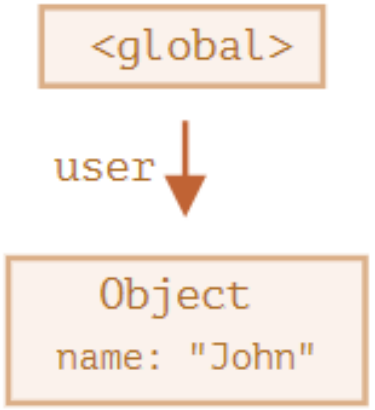



```
1 let user = { name: "John" };
2
3 let admin = user; // copy the reference
4
5 admin.name = "Peter"
6 console.log(user.name) // Peter
```

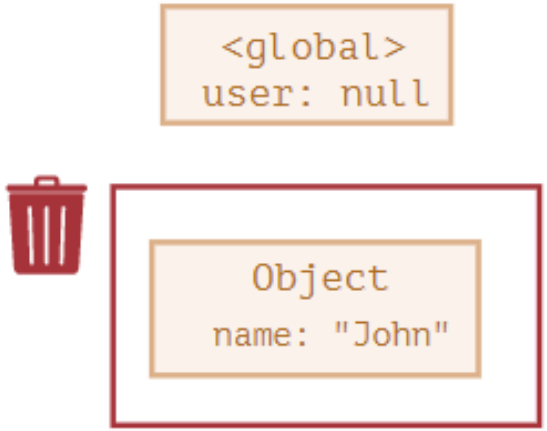


Garbage collection

```
1 // user has a reference to the object
2 let user = {
3   name: "John"
4 };
```

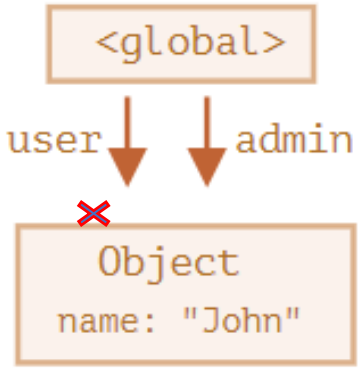



```
1 // object has no reference, garbage coll
  will junk the data and free the memory
2 user = null;
```

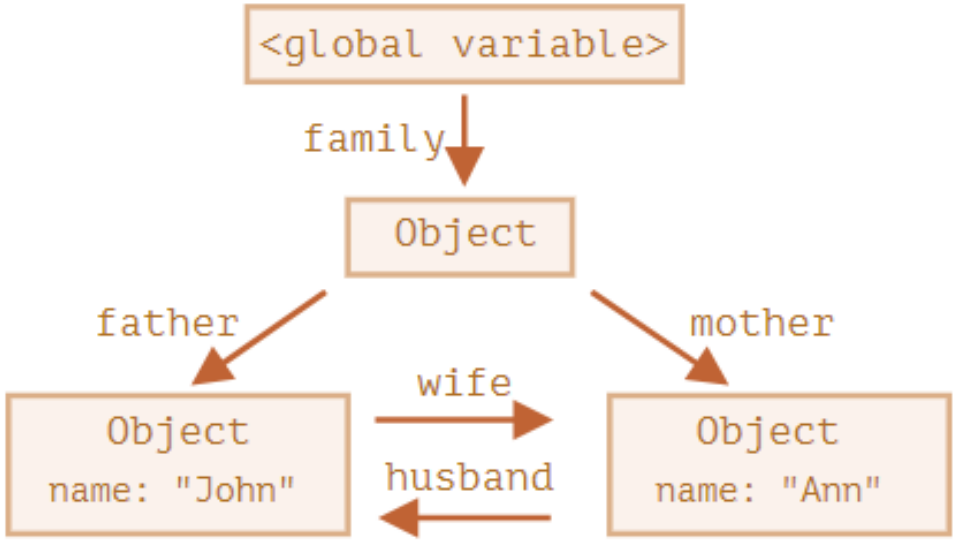


Garbage collection

```
let user = {
  name: "John"
};

let admin = user;
user = null; // object is still reachable via admin variable, so it must stay in memory
```

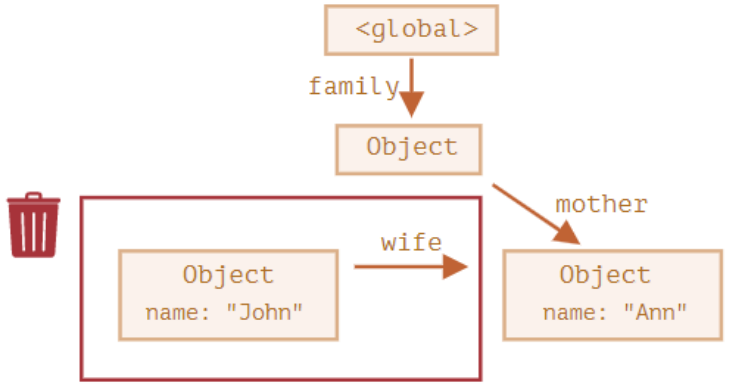



```
function marry(man, woman) {
  woman.husband = man
  man.wife = woman

  return {
    father: man,
    mother: woman,
  }
}

let family = marry(
  {name: 'John'},
  {name: 'Ann'}
)
```

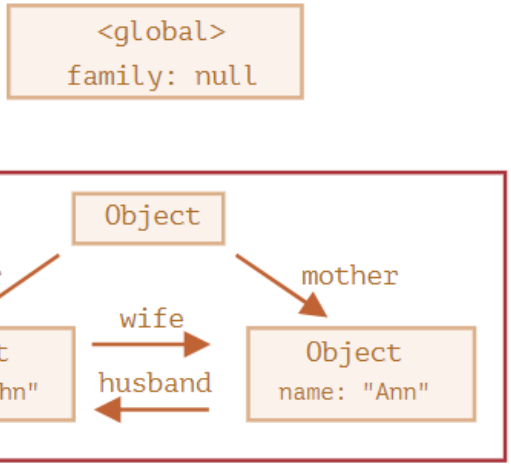


Garbage collection

```
1 delete family.father;
2 delete family.mother.husband;
3 // if we remove only one reference, all objects
  would still be reachable. But if we delete both,
  John has no incoming reference and will be junk.
```




```
1 family = null; // family object has been unlinked
  from root, so the whole island becomes unreachable
  and will be removed
```



String Object

- a String object encapsulates a sequence of characters, enclosed in quotes. Properties include
  - length : stores the number of characters in the string methods include
  - charAt(index): returns the character stored at the given index
  - substring(start, end): returns the part of the string between the start (inclusive) and end (exclusive) indices
  - toUpperCase(): returns copy of string with letters uppercase
  - toLowerCase(): returns copy of string with letters lowercase
- to create a string, assign using new or just make a direct assignment (new is implicit)
  - word = new String("foo"); word = "foo";

  properties/methods are called exactly as in C++/Java
  - word.length word.charAt(0)

String example: Palindromes

```
function strip(str)
{
  let copy = "";
  for (let i = 0; i < str.length; i++) {
    if ((str.charAt(i) >= "A" && str.charAt(i) <= "Z") ||
        (str.charAt(i) >= "a" && str.charAt(i) <= "z")) {
      copy += str.charAt(i);
    }
  }
  return copy;
}

function isPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
  str = strip(str.toUpperCase());

  for(let i = 0; i < Math.floor(str.length/2); i++) {
    if (str.charAt(i) != str.charAt(str.length-i-1)) {
      return false;
    }
  }
  return true;
}
```

palindrome

noon Radar

Madam, I'm Adam.

- must strip non-letters
- make all chars uppercase in order to be case-insensitive
- finally, traverse and compare chars

```
<head>
 <title>Palindrome Checker</title>
  <script type="text/javascript">
    function strip(str)
    {
       // CODE AS SHOWN ON PREVIOUS SLIDE
    }
    function isPalindrome(str)
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }
  </script>
</head>
<body>
  <script type="text/javascript">
    const text = prompt("Enter a word or phrase", "Madam, I'm Adam");
    if (isPalindrome(text)) {
      document.write("'" + text + "' <b>is</b> a palindrome.");
    }
    else {
      document.write("'" + text + "' <b>is not</b> a palindrome.");
    }
  </script>
</body>
```

Math Object

```
<!DOCTYPE html>
<html>
<head>
  <title>Random Dice Rolls</title>
</head>
<body>
  <div style="text-align:center">
    <script type="text/javascript">
      const roll1 = Math.floor(Math.random()*6) + 1;
      const roll2 = Math.floor(Math.random()*6) + 1;
      document.write(”
         <img src='" + roll1 +".gif'/>");
      document.write(”
         <img src='" + roll2 + ".gif'/>");
    </script>
  </div>
</body>
</html>
```

Math object contains functions and constants

```
Math.sqrt
Math.pow
Math.abs
Math.max
Math.min
Math.floor
Math.ceil
Math.round

Math.PI
Math.E
```

`Math.random` function returns a real number in [0..1)

Math Object

- ceil(4.7)=? 5
- floor(4.7)=? 4
- round(4.7)=? 5

- ceil(4.2)=? 5
- floor(4.2)=? 4
- round(4.2)=? 4

Arrays

- Arrays store a sequence of items, accessible via an index
  - since JavaScript is loosely typed, elements do not have to be the same type
  - to create an array, allocate space using new (or can assign directly)

```
let items = new Array(10);        // allocates space for 10 items
let items = new Array(); // if no size given, will adjust dynamically
let items = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
```

  - to access an array element, use [] (as in C++/Java)

```
for (i = 0; i < 10; i++) {
    items[i] = 0;   // stores 0 at each index
}
```

  - the length property stores the number of items in the array

```
for (i = 0; i < items.length; i++) {
    document.write(items[i] + "<br>");    // displays elements
}
```

Array Example

```
<!DOCTYPE html>
<html>
<head>
 <title>Dice Statistics</title>
 <script type="text/javascript" src="random.js">
 </script>
</head>
<body>
 <script type="text/javascript">
    const numRolls = 60000;
    const diceSides = 6;
    let rolls = new Array(dieSides+1);
    for (i = 1; i < rolls.length; i++) {
        rolls[i] = 0;
    }
    for(i = 1; i <= numRolls; i++) {
        rolls[randomInt(1, dieSides)]++;
    }
    for (i = 1; i < rolls.length; i++) {
        document.write("Number of " + i + "'s = " +rolls[i] + "<br />");
    }
  </script>
</body>
</html>
```

Arrays (cont.)

- Arrays have predefined methods that allow them to be used as stacks, queues, or other common programming data structures.

```
let stack = new Array();
stack.push("blue");
stack.push(12);             //  stack is now the array ["blue", 12]
stack.push("green");        //  stack = ["blue", 12, "green"]
let item = stack.pop();     //  item is now equal to "green"

let q = [1,2,3,4,5,6,7,8,9,10];
item = q.shift();       //  item is now equal to 1, remaining
                        //  elements of q move down one position
                        //  in the array, e.g. q[0] equals 2
q.unshift(125);  //  q is now the array [125,2,3,4,5,6,7,8,9,10]
q.push(244);     //  q = [125,2,3,4,5,6,7,8,9,10,244]
```

Date Object

- Date object can be used to access the date and time
  - create a Date object

    ```
    const today = new Date();     // sets to current date & time
    const newYear = new Date(2025,0,1); //sets to Jan 1, 2025  12:00AM
    ```

  - methods include:

getFullYear()	year (yyyy)
getMonth()	month (0, 11)
getDate()	day (1, 31)
getDay()	weekday (0, 6)
getHours()	hour (0, 23)
getMinutes()	minute (0, 59)
getSeconds()	second (0, 59)
getMilliseconds()	millisecond (0, 999)
getTime()	milliseconds since 1/1/1970 00:00:00 UTC (Unix epoch)

Date Example

```
<body>
  Time when page was loaded:
  <script type="text/javascript">
    const now = new Date();
    document.write("<p>" + now + "</p>");
    let time = "AM";
    let hours = now.getHours();
    if (hours > 12) {
        hours -= 12;
        time = "PM"
    }
    else if (hours == 0) {
        hours = 12;
    }
    document.write("<p>" + hours + ":" +
    now.getMinutes() + ":" +
    now.getSeconds() + " " +
    time + "</p>");
  </script>
</body>
```

by default, a date will be displayed in full, e.g.,

Wed, Sep 25 22:55:20 2024

can pull out portions of the date using the methods and display as desired

here, determine if "AM" or "PM" and adjust so hour between 1-12

10:55:20 PM

Another Example

```
<body>
  <p>Elapsed time in this year:
  <script type="text/javascript">
    const now = new Date();
    const newYear = new Date(2025,0,1);
    let secs = Math.round((now-newYear)/1000);
    let days = Math.floor(secs / 86400);
    secs -= days*86400;
    let hours = Math.floor(secs / 3600);
    secs -= hours*3600;
    let minutes = Math.floor(secs / 60);
    secs -= minutes*60
    document.write(days + " days, " +
                   hours + " hours, " +
                   minutes + " minutes, and " +
                   secs + " seconds.");
  </script>
  </p>
</body>
```

Document Object

```
<body>
  <table width="100%">
    <tr>
      <td><i>
        <script type="text/javascript">
        document.write(document.URL);
        </script>
      </i></td>
      <td style="text-align: right;"><i>
        <script type="text/javascript">
document.write(document.lastModified);
        </script>
      </i></td>
    </tr>
  </table>
</body>
```

document.write(...)

method that displays text in the page

document.URL

property that gives the location of the HTML document

document.lastModified

property that gives the date & time the HTML document was last changed

User-Defined Objects

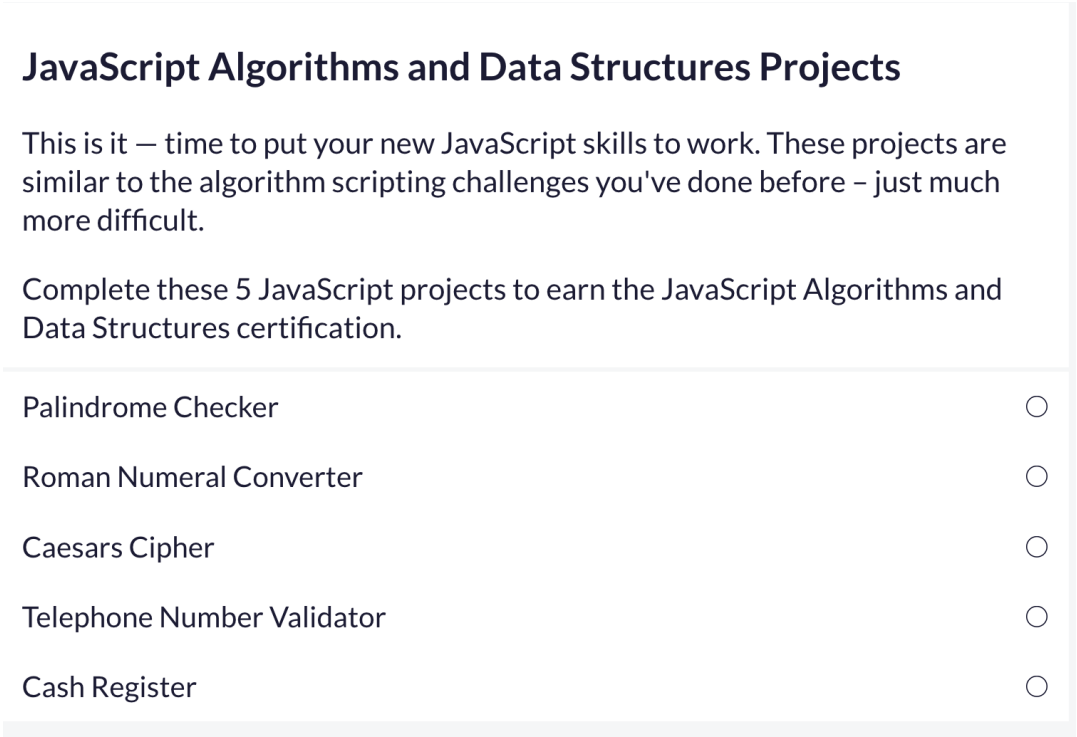
- User can create a class by using keyword “class”

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    const date = new Date();
    return date.getFullYear() - this.year;
  }
}

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

Exercise

- Do 5 exercises on Algorithms and Data Structures
- Link: https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#javascript-algorithms-and-data-structures-projects



JavaScript Algorithms and Data Structures Projects

This is it — time to put your new JavaScript skills to work. These projects are similar to the algorithm scripting challenges you've done before – just much more difficult.

Complete these 5 JavaScript projects to earn the JavaScript Algorithms and Data Structures certification.

Palindrome Checker ○

Roman Numeral Converter ○

Caesars Cipher ○

Telephone Number Validator ○

Cash Register ○