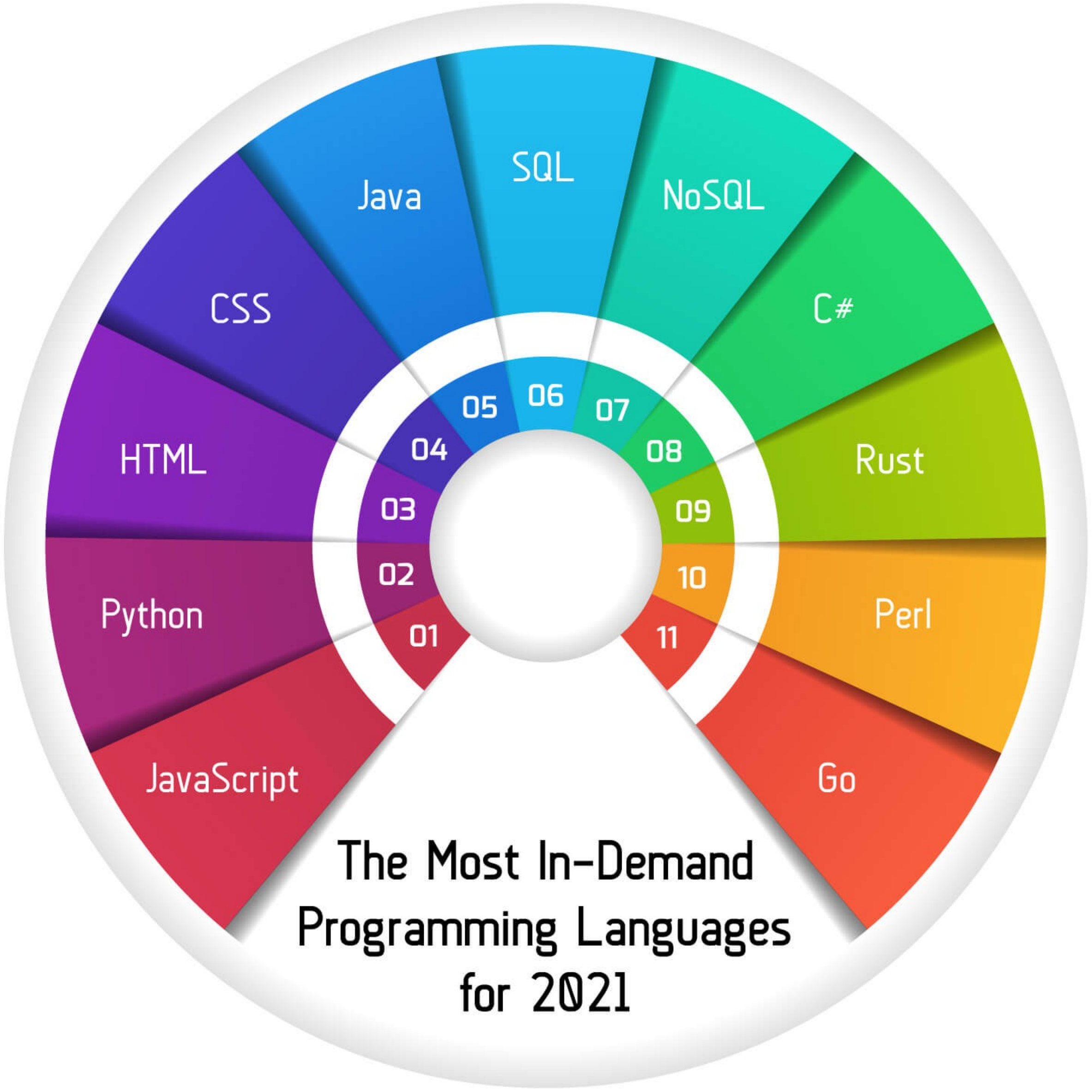
JavaScript



src: https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/

Content

- Scripts
  - common tasks for client-side scripts
- JavaScript
  - data types & expressions
  - control statements
  - functions & libraries
  - strings & arrays
  - Date, document, navigator, 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, with (HTML) tags to identify the program component
    - the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML
    - could also allow the user (client) to input information and process it, might be used to validate input before it's submitted to a remote server

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies
- defining programs with Web interfaces
  - utilize buttons, text boxes, clickable images, prompts, etc
- 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

- **ECMA International**
  - European Computer Manufacturers Association
  - Non-profit organization that develops standards in computer hardware, communications, and programming languages.

- JavaScript: A general purpose scripting language that conforms to the ECMAScript specification
- 1997: ECMA-262 standard
- 2022: ECMAScript 13 was released in June 2022

JavaScript

- Javascript is a lightweight, interpreted or just-in-time compiled programming language
- Many non-browser environtments use it such as Node.js, Apache CouchDB, Adobe Acrobat
- 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

```
<html>
<!-- CS443  js01.html  16.08.06 -->

<head>
  <title>JavaScript Page</title>
</head>

<body>
  <script type="text/javascript">
    // silly code to demonstrate output

    document.write("<p>Hello world!</p>");

    document.write(" <p>How are <br/> " +
                   " <i>you</i>?</p> ");
  </script>

  <p>Here is some static text as well.</p>

</body>
</html>
```

view page

- Use <script> tag to add Javascript code to a page
- document.write displays text in the page
  - text to be displayed can include HTML tags
- JavaScript comments similar to C++/Java

  // starts a single line comment
  /*…*/ enclose multi-line comments

JavaScript Data Types & Variables

```
<html>
<!-- CS443  js02.html  16.08.06 -->

<head>
  <title>Data Types and Variables</title>
</head>
<body>
  <script type="text/javascript">
    var x, y;
    x= 1024;

    y=x;  x = "foobar";
    document.write("<p>x = " + y + "</p>");
    document.write("<p>x = " + x + "</p>");
  </script>
</body>
</html>
```

view page

- JavaScript has 7 primitive data types

  String : "foo" 'how do you do?' "I said 'hi'."

  Number: 12 3.14159 1.5E6

  Bigint (ES2020, số > 2^{53} -1): 9007199254740991n

  Boolean : true false

  Undefined: undefined

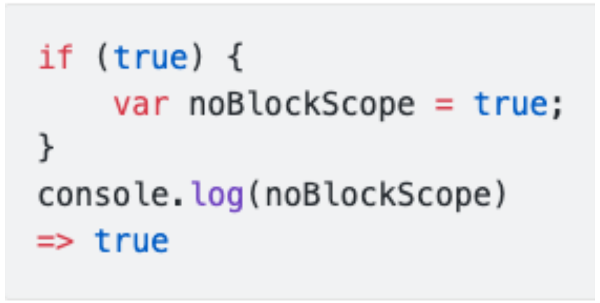
  Symbol: s = Symbol(‘first name’);

  null: null

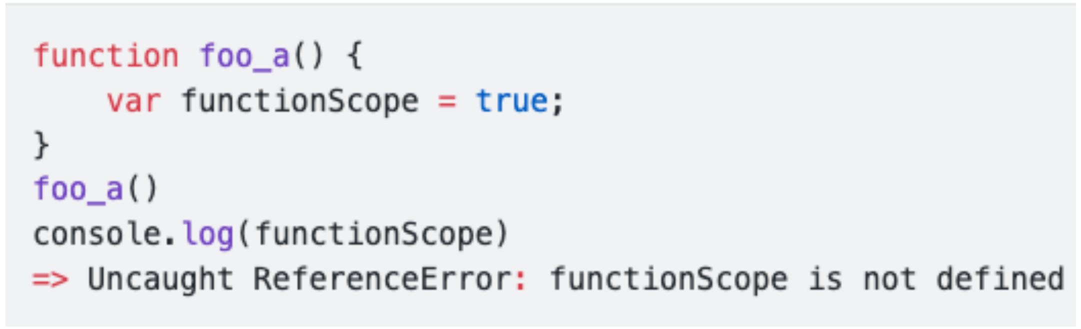
variable names are sequences of letters, digits, an underscores that start with a letter or an underscore, case sensitive

JavaScript Declaration

- var: function scope or global scope

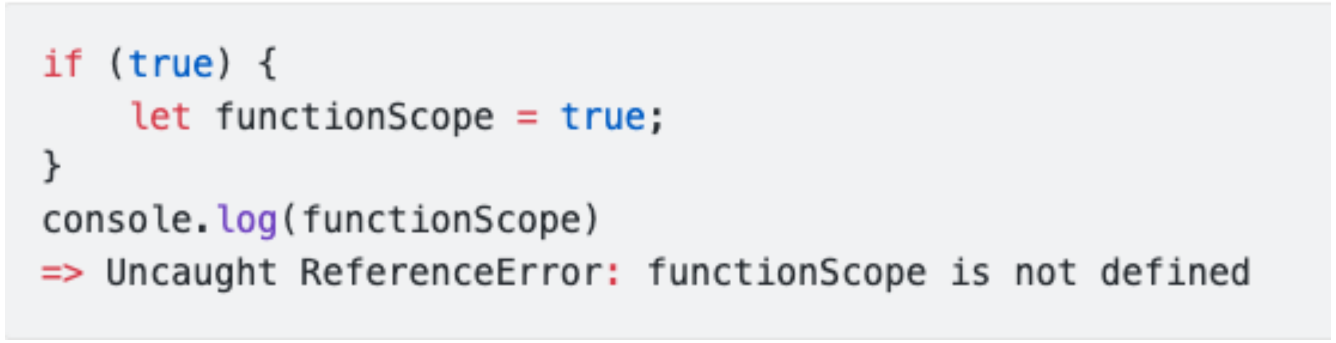


```
if (true) {
    var noBlockScope = true;
}
console.log(noBlockScope)
=> true
```



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

- let: block scope



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

- const: same as let, except the user cannot update it

JavaScript Operators & Control Statements

```
<html>
<!-- CS443 js03.html  08.10.10 -->

<head>
  <title>Folding Puzzle</title>
</head>

<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>
```

view page

standard C++/Java operators & control statements are provided in JavaScript

- `+, -, *, /, %, ++, --, …`
- `==, !=, <, >, <=, >=`
- `&&, ||, !,===,!==`
- `if , if-else, switch`
- `while, for, do-while, …`

PUZZLE: Suppose you took a piece of paper and folded it in half, then in half again, and so on.
How many folds before the thickness of the paper reaches from the earth to the sun?

JavaScript Math Routines

```
<html>
<!-- CS443  js04.html  08.10.10 -->
<head>
  <title>Random Dice Rolls</title>
</head>
<body>
  <div style="text-align:center">
    <script type="text/javascript">
      let roll1 = Math.floor(Math.random()*6) + 1;
      let roll2 = Math.floor(Math.random()*6) + 1;
      document.write("<img
src='http://www.csc.liv.ac.uk/"+
           "~martin/teaching/CS443/Images/die" +
           roll1 + ".gif' alt='dice showing ' +
roll1 />");
      document.write("&nbsp;&nbsp;");
      document.write("<img
src='http://www.csc.liv.ac.uk/"+
           "~martin/teaching/CS443/Images/die" +
           roll2 + ".gif' alt='dice showing ' +
roll2 />");
    </script>
  </div>
</body>
</html>
```

view page

the built-in 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)

Interactive Pages Using Prompt

```
<html>
<!-- CS443  js05.html  08.10.10 -->
<head>
  <title>Interactive page</title>
</head>
<body>
<script type="text/javascript">
 let userName = prompt("What is your name?",
"");
 let userAge = prompt("Your age?", "");
 let userAge = parseFloat(userAge);
    document.write("Hello " + userName + ".")
    if (userAge < 18) {
      document.write("  Do your parents know "
+.  "you are online?");
    }
    else {
      document.write("  Welcome friend!");
    }
</script>
  <p>The rest of the page...</p>
</body>
</html>
```

view page

crude user interaction can take place using prompt

1st argument: the prompt message that appears in the dialog box

2nd argument: a default value that will appear in the box (in case the user enters nothing)

the function returns the value entered by the user in the dialog box (a string)

if value is a number, must use parseFloat (or parseInt) to convert

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)
// Assumes: n > 0
// Returns: true if n is prime, else false
{
  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

```
<html>
<!-- CS443  js06.html  16.08.2006 -->
<head>
  <title>Prime Tester</title>
  <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">
    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>
```

view page

Function definitions (usually) go in the <head> section

<head> section is loaded first, so then the function is defined before code in the <body> is executed

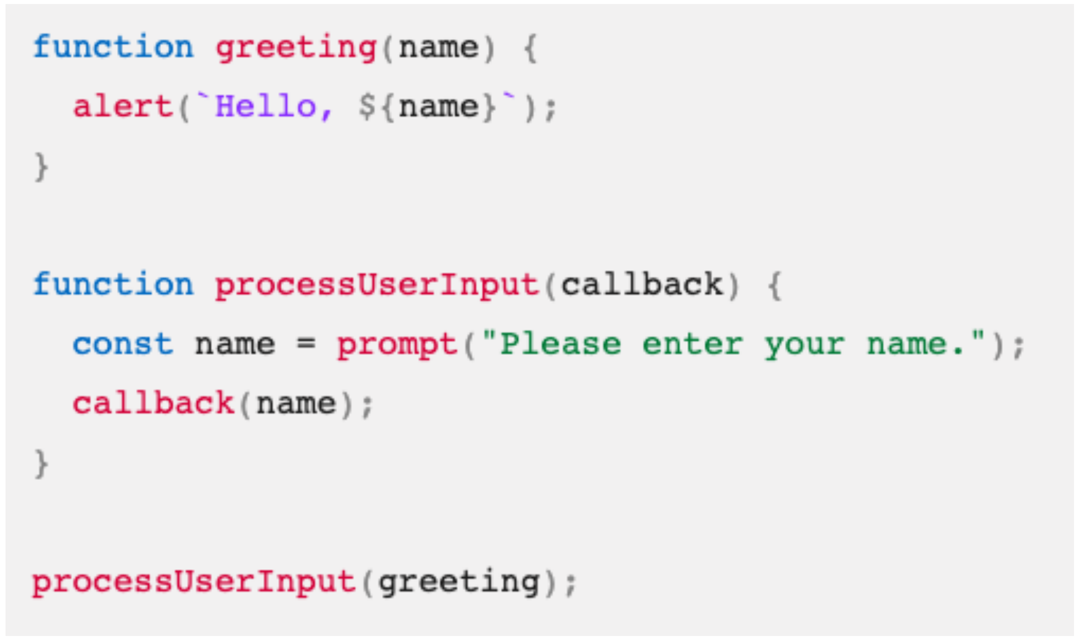
Another Example

```
<html>
<!-- CS443  js07.html  11.10.2011 -->
<head>
  <title> Random Dice Rolls Revisited</title>
  <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>
  <div style="text-align: center">
    <script type="text/javascript">
      roll1 = randomInt(1, 6);
      roll2 = randomInt(1, 6);
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll2 + ".gif'/>");
</script>
  </div>
</body>
</html>
```

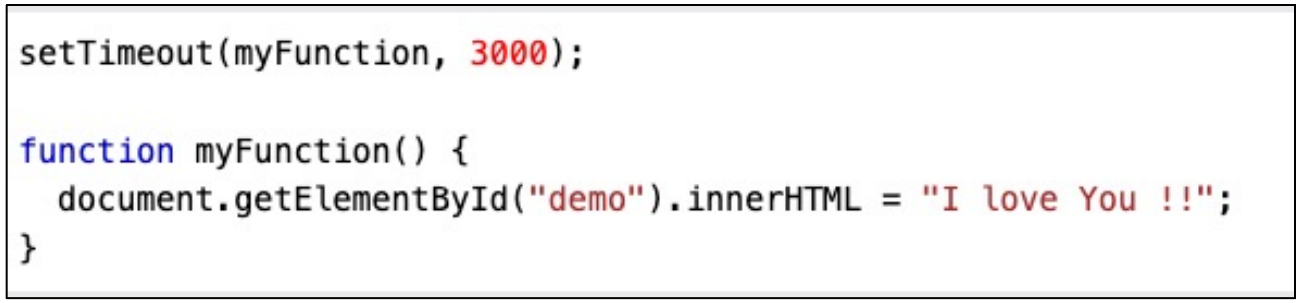
view page

Callback Function

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



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

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

processUserInput(greeting);
```



```javascript
setTimeout(myFunction, 3000);

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

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 (put nothing between the beginning and ending tags)

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

Library Example

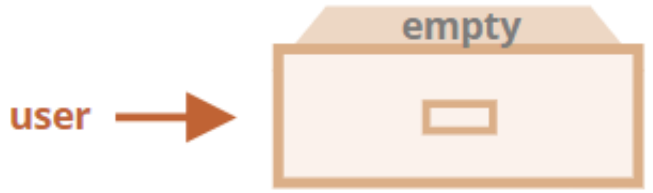
```
<html>
<!-- CS443  js08.html  11.10.2011 -->

<head>
  <title> Random Dice Rolls Revisited</title>

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

<body>
  <div style="text-align: center">
    <script type="text/javascript">
      roll1 = randomInt(1, 6);
      roll2 = randomInt(1, 6);
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll2 + ".gif'/>");
    </script>
  </div>
</body>
</html>
```

view page

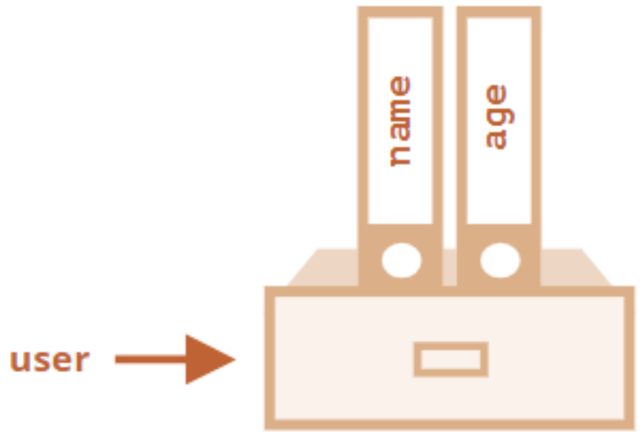
Objects

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
```

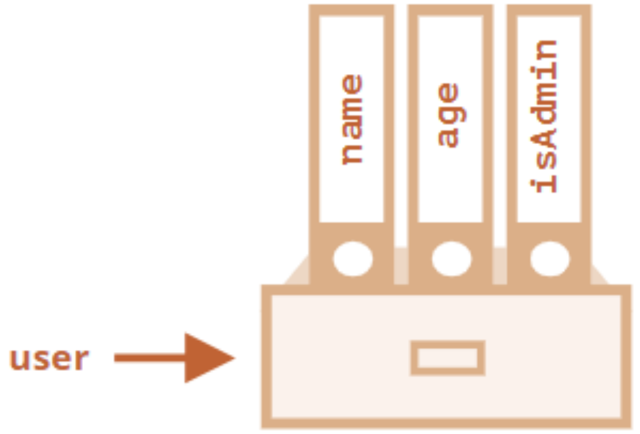


```
let user = {      // an object
  name: "John",   // by key "name" store value "John"
  age: 30         // by key "age" store value 30
};
console.log(user.name) // John
console.log(user.age)  // 30
```

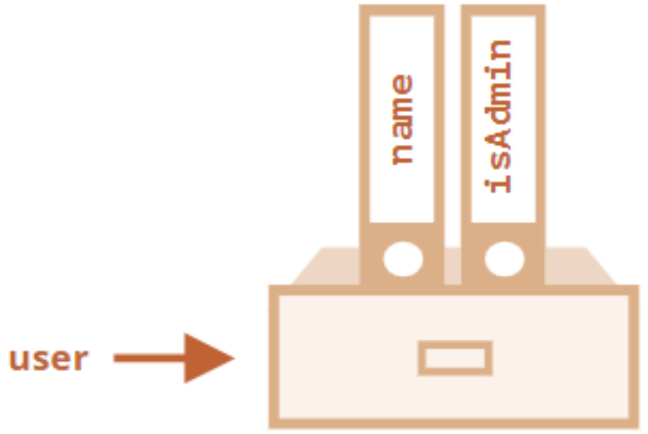


Objects

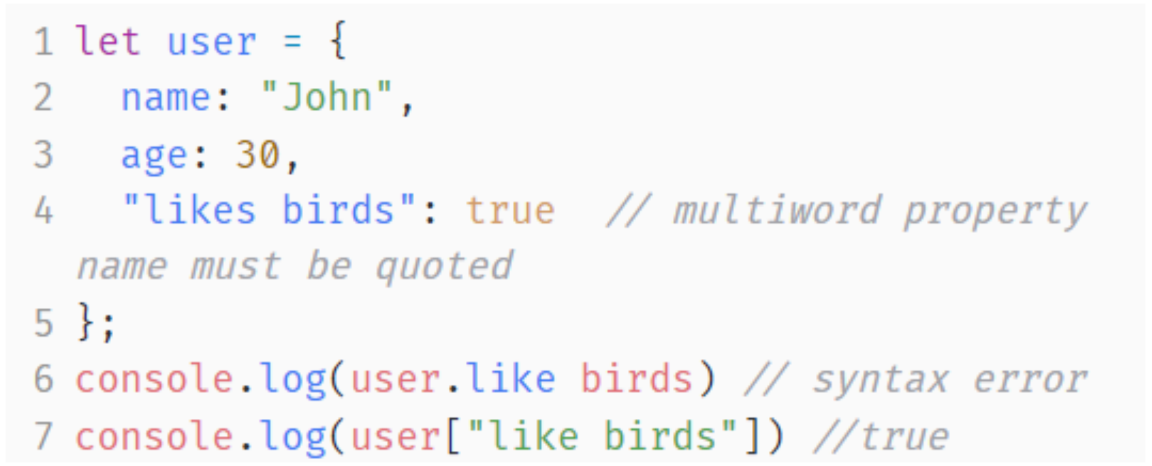
```
user.isAdmin = true;
```



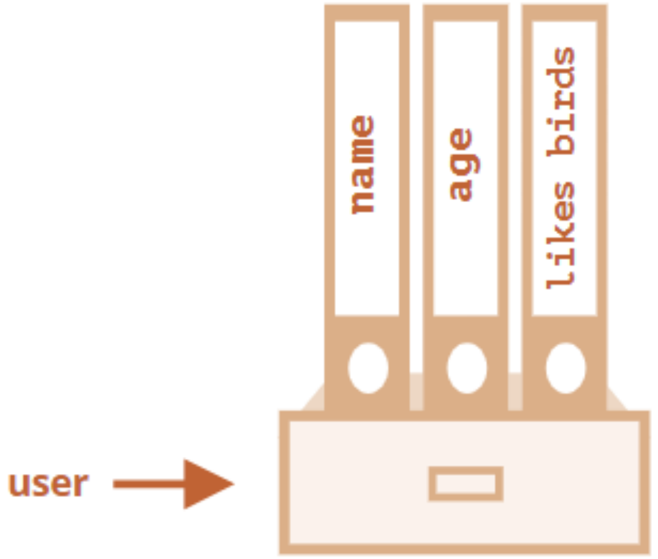
```
delete user.age
```



Objects



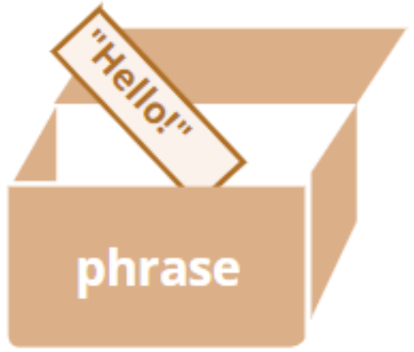
```
let user = {
  name: "John",
  age: 30,
  "likes birds": true   // multiword property
name must be quoted
};
console.log(user.like birds) // syntax error
console.log(user["like birds"]) //true
```



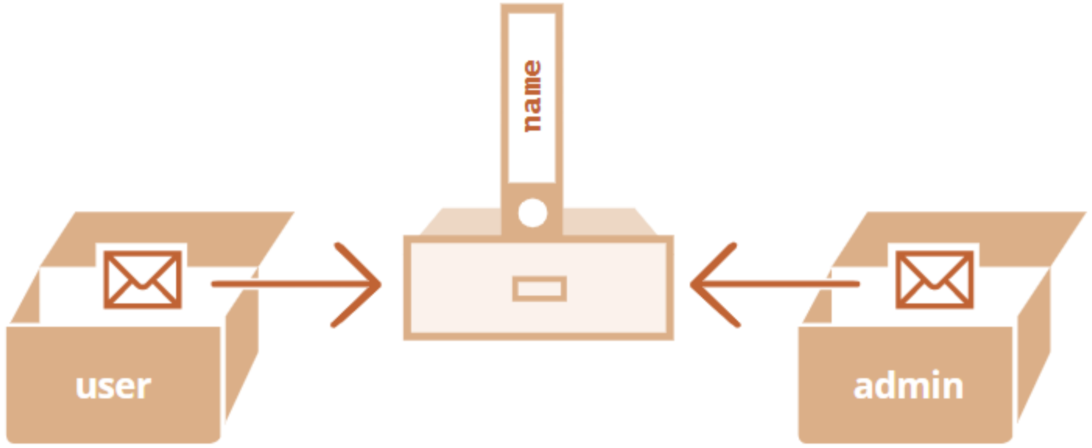
Object references

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

"Hello!"
message

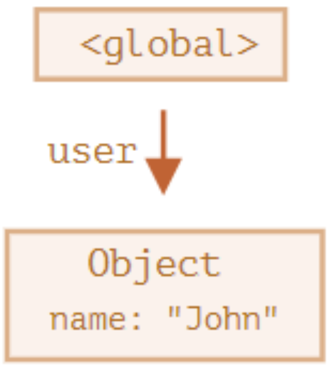


```
let user = { name: "John" };

let admin = user; // copy the reference

admin.name = "Peter"
console.log(user.name) // Peter
```

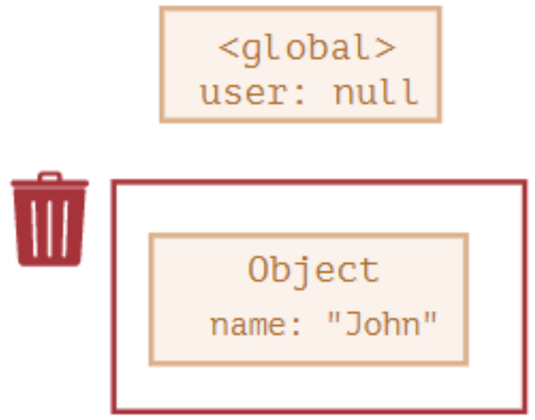


Garbage collection

```javascript
// user has a reference to the object
let user = {
  name: "John"
};
```

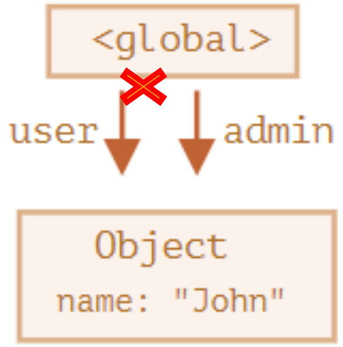


```javascript
// object has no reference, garbage collector will junk the data and free the memory
user = null;
```

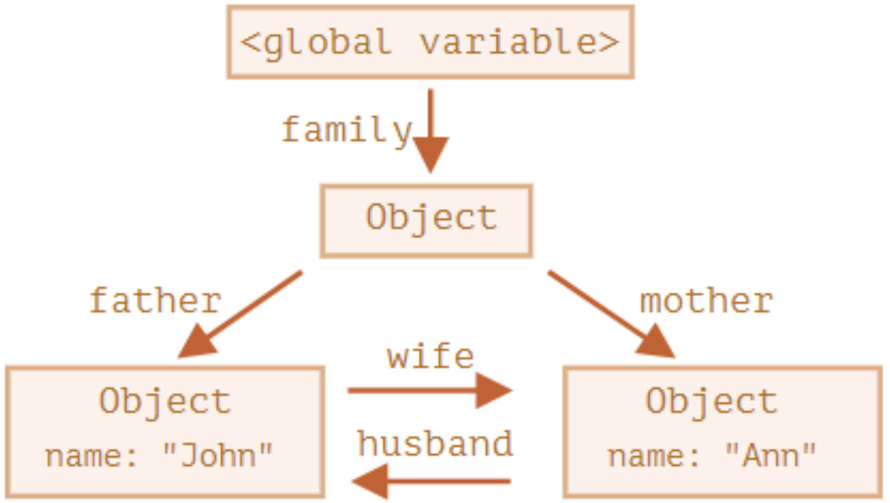


Garbage collection

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

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

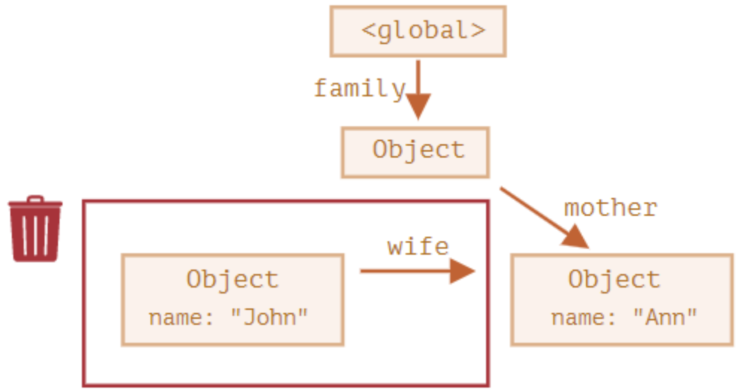


```
function marry(man, woman) {
  woman.husband = man
  man.wife = woman

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

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

```

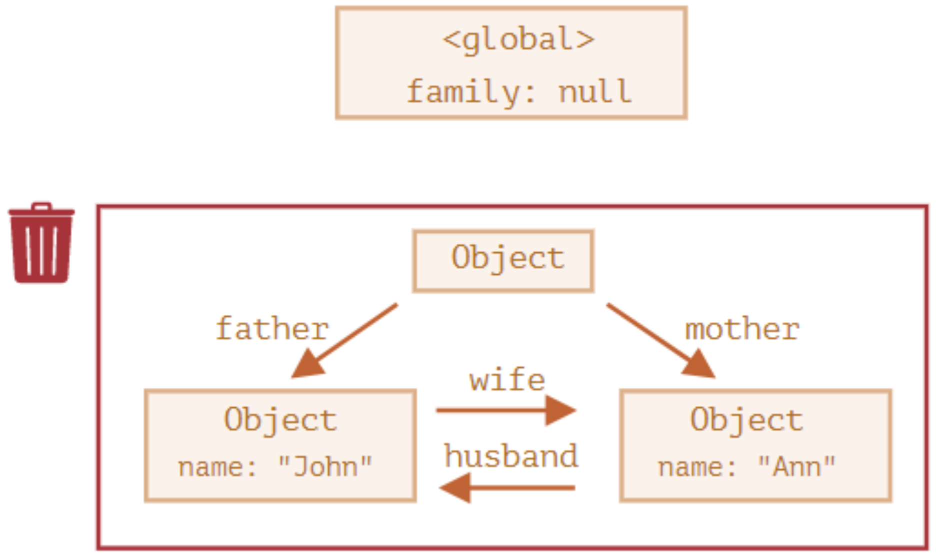


Garbage collection

```
delete family.father;
delete family.mother.husband;
// 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.
```



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



JavaScript Objects

- 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 (in this case) 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)
// Assumes: str is a string
// Returns: str with all but letters removed
{
  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;
}
```

suppose we want to test whether a word or phrase is a palindrome

noon   Radar
Madam, I'm Adam.

must strip non-letters out of the word or phrase

make all chars uppercase in order to be case-insensitive

finally, traverse and compare chars from each end

```
<html>
<!-- CS443  js09.html  11.10.2011 -->
<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">
    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>
</html>
```

view page

JavaScript 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)

```
items = new Array(10);  // allocates space for 10 items
items = new Array();    // if no size given, will adjust dynamically
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

```
<html>
<!-- CS443  js10.html  11.10.2011 -->
<head>
 <title>Dice Statistics</title>
 <script type="text/javascript"
src="http://www.csc.liv.ac.uk/~martin/teaching/CS443/JS/rand
om.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>
```

view page

suppose we want to simulate dice rolls and verify even distribution

keep an array of counters:

-initialize each count to 0

-each time you roll X, increment rolls[X]

-display each counter

Arrays (cont.)

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

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

var 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

- String & Array are the most commonly used objects in JavaScript
  - other, special purpose objects also exist
- the Date object can be used to access the date and time
  - to create a Date object, use new & supply year/month/day/… as desired

```
today = new Date();        // sets to current date & time
newYear = new Date(2002,0,1); //sets to Jan 1, 2002  12:00AM
```

  - methods include:

```
newYear.getFullYear()     can access individual components of a date
newYear.getMonth()        number (0, 11)
newYear.getDay()          number (1, 31)
newYear.getHours()        number (0, 23)
newYear.getMinutes()      number (0, 59)
newYear.getSeconds()      number (0, 59)
newYear.getMilliseconds() number (0, 999)
```

Date Example

```
<html>
<!-- CS443  js11.html  16.08.2006 -->
<head>
  <title>Time page</title>
</head>
<body>
  Time when page was loaded:
  <script type="text/javascript">
    now = new Date();
    document.write("<p>" + now + "</p>");
    time = "AM";
    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>
</html>
```

view page

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

Sun Feb 03 22:55:20 GMT-0600 (Central Standard Time) 2002

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

```
<html>
<!-- CS443  js12.html  12.10.2012 -->
<head>
  <title>Time page</title>
</head>
<body>
  <p>Elapsed time in this year:
  <script type="text/javascript">
    now = new Date();
    newYear = new Date(2012,0,1);
    secs = Math.round((now-newYear)/1000);
    days = Math.floor(secs / 86400);
    secs -= days*86400;
    hours = Math.floor(secs / 3600);
    secs -= hours*3600;
    minutes = Math.floor(secs / 60);
    secs -= minutes*60
    document.write(days + " days, " +
                   hours + " hours, " +
                   minutes + " minutes, and " +
                   secs + " seconds.");
  </script>
  </p>
</body>
</html>
```

view page

you can add and subtract Dates:
the result is a number of milliseconds

here, determine the number of seconds since New Year's day (note: January is month 0)

divide into number of days, hours, minutes and seconds

Document Object

Internet Explorer, Firefox, Opera, etc. allow you to access information about an HTML document using the document object

```
<html>
<!-- CS443  js13.html  2.10.2012 -->
<head>
  <title>Documentation page</title>
</head>
<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>
</html>
```

view page

`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

- can define new objects, but the notation can be somewhat awkward
  - simply define a function that serves as a constructor
  - specify data fields & methods using `this`
  - no data hiding: can't protect data or methods

```
// CS443        Die.js       11.10.2011 //
// Die class definition
////////////////////////////////////////////

function Die(sides)
{
   this.numSides = sides;
   this.numRolls = 0;
   this.roll = roll;    // define a pointer to a function
}

function roll()
{
    this.numRolls++;
    return Math.floor(Math.random()*this.numSides) + 1;
}
```

define Die function (i.e., the object's constructor)

initialize data fields in the function, preceded with "this"

similarly, assign method to separately defined function (which uses this to access data)

Object Example

```
<html>
<!-- CS443  js15.html  11.10.2011 -->
<head>
  <title>Dice page</title>
  <script type="text/javascript"
         src="Die.js">
  </script>
</head>
<body>
 <script type="text/javascript">
    die6 = new Die(6);       die8 = new Die(8);
    roll6 = -1;    // dummy value to start loop
    roll8 = -2;    // dummy value to start loop
    while (roll6 != roll8) {
      roll6 = die6.roll();
      roll8 = die8.roll();

      document.write("6-sided: " + roll6 +
                     "&nbsp;&nbsp;&nbsp;&nbsp;" +
                     "8-sided: " + roll8 + "<br />");
    }
    document.write("<br />Number of rolls: " +
                   die6.numRolls);
  </script>
</body>
</html>
```

view page

create a Die object using new (similar to String and Array)

here, the argument to Die initializes numSides for that particular object

each Die object has its own properties (numSides & numRolls)

Roll(), when called on a particular Die, accesses its numSides property and updates its NumRolls

More to learn…

- Accessing elements on the page using JavaScript functions
- JavaScript and forms
- Events, capturing user input
- The Document Object Model, and manipulating the webpage