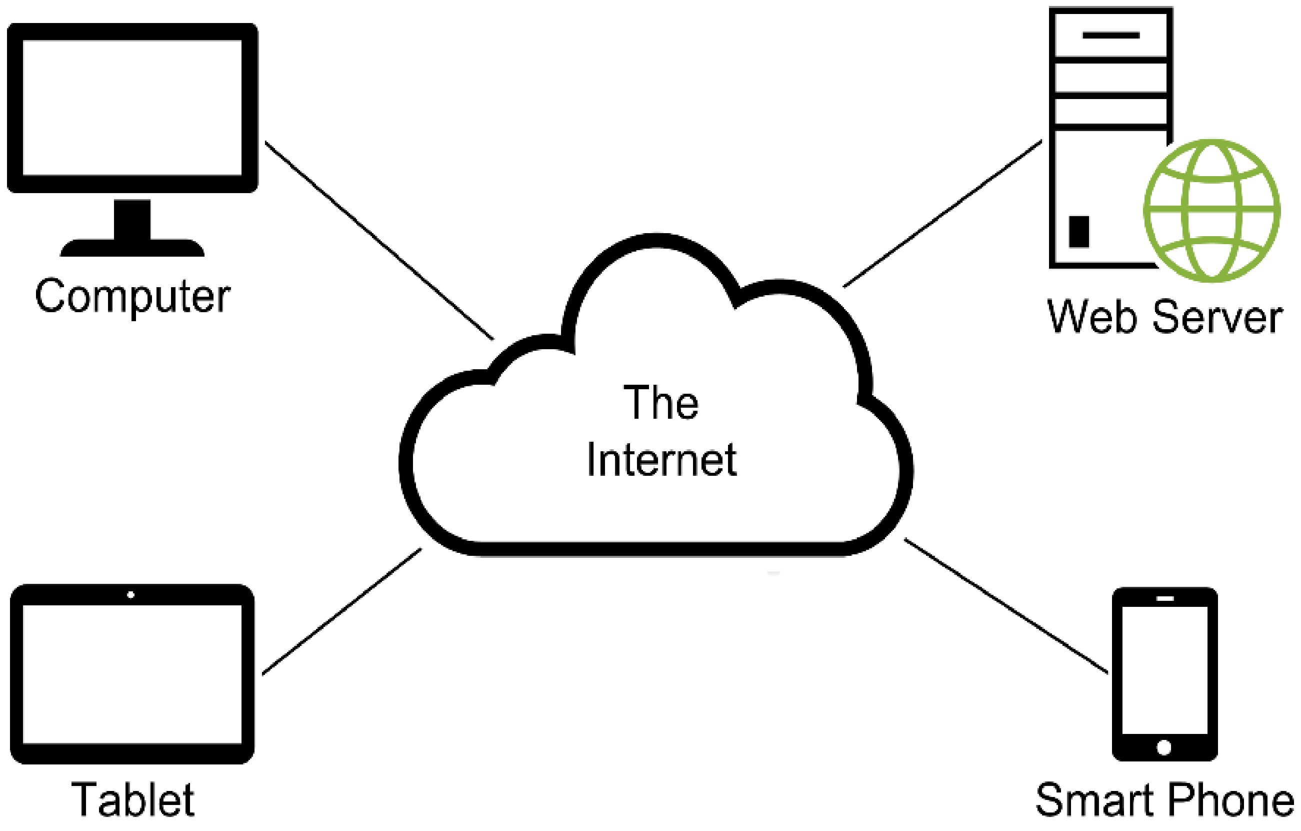
Revise PROG8660

Revise PROG8660

The components of a web application



Terms related to web applications

client

web browser

web server

network

intranet

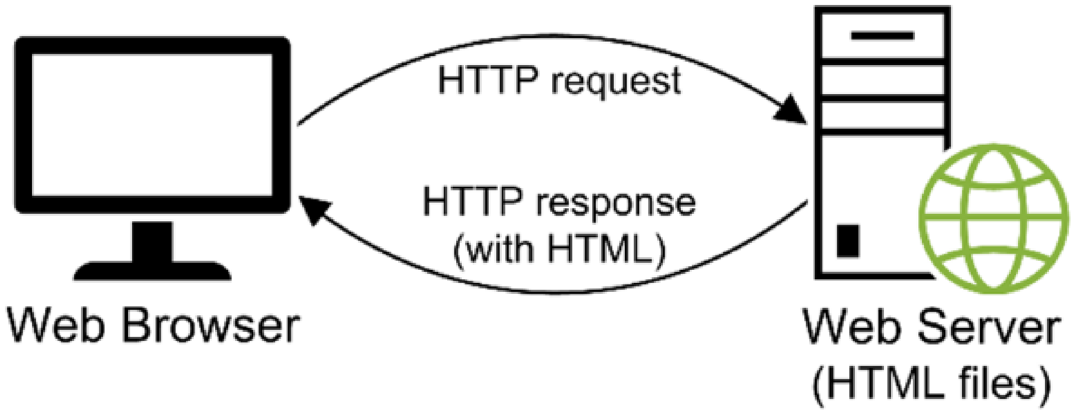
local area network (LAN)

Internet

wide area network (WAN)

Internet Service Provider (ISP)

How a web server processes a static web page



Terms related to static web pages

Hypertext Markup Language (HTML)

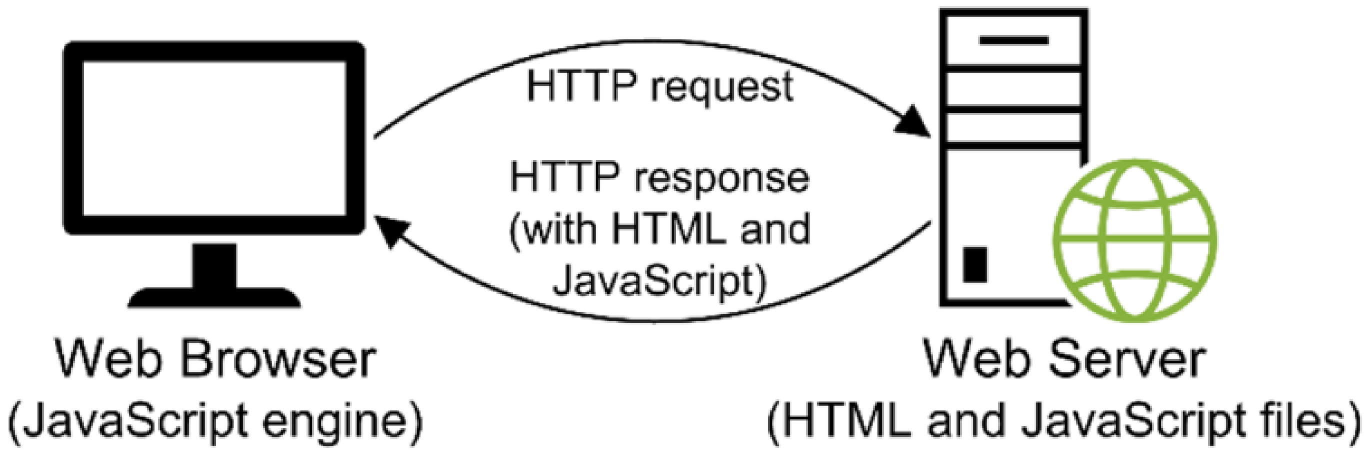
static web page

HTTP request

HTTP response

rendering a page

How JavaScript fits into this architecture


HTTP request
HTTP response (with HTML and JavaScript)
Web Browser (JavaScript engine)
Web Server (HTML and JavaScript files)

Terms related to client-side processing

scripting language

JavaScript engine

jQuery

client-side processing

Two attributes of the script element

```
src
type
```

A script element in the body section that loads an external JavaScript file

```
<body>
    ...
    <script src="calculate_mpg.js"></script>
</body>
```

Terms related to including JavaScript

external JavaScript file

embedded JavaScript

The basic syntax rules for JavaScript

JavaScript is case-sensitive.

Each JavaScript statement ends with a semicolon.

JavaScript ignores extra whitespace within statements.

When JavaScript is in *strict mode*, it disallows certain JavaScript features and coding practices that are considered unsafe.

Reserved words in JavaScript

`abstract`	`else`	`instanceof`	`switch`
`arguments`	`enum`	`int`	`synchronized`
`boolean`	`eval`	`interface`	`this`
`break`	`export`	`let`	`throw`
`byte`	`extends`	`long`	`throws`
`case`	`false`	`native`	`transient`
`catch`	`final`	`new`	`true`
`char`	`finally`	`null`	`try`
`class`	`float`	`package`	`typeof`
`const`	`for`	`private`	`var`
`continue`	`function`	`protected`	`void`
`debugger`	`goto`	`public`	`volatile`
`default`	`if`	`return`	`while`
`delete`	`implements`	`short`	`with`
`do`	`import`	`static`	`yield`
`double`	`in`	`super`	

JavaScript’s primitive data types

Number

String

Boolean

Symbol

Null

Undefined

BigInt (ES2020)

How to declare and initialize a variable

Syntax

```
let variableName = value;
```

Examples

```
let count = 1;                // integer value of 1
let subtotal = 74.95;         // decimal value of 74.95

let name = "Joseph";          // string value of "Joseph"
let email = "";               // empty string

let isValid = false;          // Boolean value of false

let total = subtotal;         // assigns value of subtotal variable

let x = 0, y = 0;             // declares and initializes
                              // 2 variables
```

How to reassign the value of a variable

```
let count = 1;
count = count + 1;            // value of count is now 2
```

JavaScript's arithmetic operators

Operator	Name	Description
`+`	Addition	Adds two operands.
`-`	Subtraction	Subtracts the right operand from the left operand.
`*`	Multiplication	Multiplies two operands.
`/`	Division	Divides the right operand into the left operand. The result is always a floating-point number.
`%`	Modulus	Divides the right operand into the left operand and returns the remainder.
`++`	Increment	Adds 1 to the operand.
`--`	Decrement	Subtracts 1 from the operand.

The order of precedence for arithmetic operators

Order	Operators	Direction	Description
1	`++`	Left to right	Increment operator
2	`--`	Left to right	Decrement operator
3	`*   /   %`	Left to right	Multiplication, division, modulus
4	`+   -`	Left to right	Addition, subtraction

A floating-point result that isn’t precise

```
const subtotal = 74.95;           // subtotal = 74.95
const salesTax = subtotal * .1;
                            // salesTax = 7.495000000000001
```

A problem with some arithmetic operations

When you do some types of arithmetic operations with decimal values, the results aren’t always precise. That’s because decimal values are stored internally as floating-point numbers.

The primary problem with this is that an equality comparison may not return true.

Constants used by the following examples

```
const firstName = "Grace", lastName = "Hopper";
```

How to concatenate string variables with the + operator

```
const name = lastName + ", " + firstName;
                               // name is "Hopper, Grace"
```

How to concatenate string variables with the += operator

```
let name = lastName;           // name is "Hopper"
name += ", ";                  // name is "Hopper, "
name += firstName;             // name is "Hopper, Grace"
```

How to concatenate string variables with a template literal

```
const name = `${lastName}, ${firstName}`;
                               // name is "Hopper, Grace"
```

Common methods of the window object

```
alert(string)
prompt(string, default)
```

The syntax for calling a method of an object

```
objectName.methodName(parameters)
```

A statement that calls the alert() method of the window object

```
window.alert("This is a test of the alert method");
```

Two methods of the window object

```
parseInt(*string*)
parseFloat(*string*)
```

The parsing that's done by the parse methods

Only the first number in the string is returned.

Leading and trailing spaces are removed.

If the first character cannot be converted to a number, NaN is returned.

A method of the Number object

```
toFixed(n)
```

An example that uses the toFixed() method

```
const pi = 3.14159;
alert(pi.toFixed(3));          // displays 3.142
```

The relational operators

Operator	Name
`==`	Equal to
`!=`	Not equal to
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to

The logical operators

Operator	Name	Description
&&	AND	Returns a true value if both expressions are true. This operator only evaluates the second expression if necessary.
||	OR	Returns a true value if either expression is true. This operator only evaluates the second expression if necessary.
!	NOT	Reverses the value of the Boolean expression.

The syntax of the if statement

```
if ( condition-1 ) { statements }
[ else if ( condition-2 ) { statements }
  ...
  else if ( condition-n ) { statements } ]
[ else { statements } ]
```

An if statement with an else clause

```
if ( age >= 18 ) {
    alert ("You may vote.");
} else {
    alert ("You are not old enough to vote.");
}
```

The syntax of a while loop

```
while ( *condition* ) { *statements* }
```

A while loop that adds the numbers 1 through 5

```
let sum = 0;
let i = 1;

while (i <= 5) {
    sum += i;                       // adds i to sum
    i++;
}
alert(sum);                         // displays 15
```

The syntax of a do-while loop

```
do { statements } while ( condition );
```

A do-while loop that makes sure a user enters a positive number

```
let years = null;

do {
    years = parseInt(prompt("Enter number of years.\n" +
        "(Must be valid positive number)"));
}
while ( isNaN(years) || years <= 0 );
```

The syntax for creating an array

Using the Array() constructor and the new keyword

```
const arrayName = new Array(length);
```

Using an array literal

```
const arrayName = [];
```

A statement that creates an array

```
const totals = [];
```

The syntax for referring to an element of an array

```
arrayName[index]
```

Statements that refer to the elements in an array

```
totals[2]        // refers to the third element – 411
totals[1]        // refers to the second element – 212.25
```

The syntax of the for-in loop

```
for ( indexInitialization in array ) {
    statements
}
```

Code that totals the numbers in the array using a for-in loop

```
let sum = 0;
for (let i in totals) {    // i holds the current index
  sum += totals[i];
}
alert(sum);                // displays 900.95
```

The syntax of a for-of loop

```
for ( valueInitialization of array ) {
    statements
}
```

Code that totals the numbers in the array using a for-of loop

```
let sum = 0;
for (let val of totals) {  // val holds the current value
  sum += val;
}
alert(sum);                // displays 900.95
```

Some of the host objects in a browser environment

Object	Description
`window`	Represents the open browser window. This is the *global object* for JavaScript.
`document`	Represents the HTML document in the browser window. Allows you to work with the Document Object Model (DOM).
`navigator`	Contains information about the browser.
`history`	Contains the URLs that a user has visited in the browser.

Some of the JavaScript native objects

Object	Description
`Object`	The base object that all other JavaScript objects inherit.
`Array`	An object that stores a collection of data.
`Date`	An object that stores a date.
`Function`	An object that stores a predefined collection of JavaScript statements.
`Number`	A wrapper object for working with the primitive number data type.
`String`	A wrapper object for working with the primitive string data type.
`Boolean`	A wrapper object for working with the primitive boolean data type.
`Math`	A utility object with static methods for working with numbers.

Three methods of the document object

```
querySelector(selector)
querySelectorAll(selector)
write(string)
```

Common ways to code selectors (part 1)

Using an element id

```
// returns the object for the HTML element
// whose id is "rate"
const rate = document.querySelector("#rate");
```

Using an element type

```
// returns an array of objects for all the <a> elements
// in the document
const links = document.querySelectorAll("a");
```

Using a class

```
// returns an array of objects for all the elements
// assigned to the error class
const errors = document.querySelectorAll(".error");
```

Common ways to code selectors (part 2)

Using a descendant selector

```
// returns an array of all the h2 elements that are
// descendants of the element whose id is "faqs".
const h2s = document.querySelectorAll("#faqs h2");
```

Using a combination of selectors

```
// returns an array of all the div elements assigned
// to the closed class
const minus = document.querySelectorAll("div.closed");
```

Using multiple selectors

```
// returns an array of elements specified by two
// descendant selectors
const elements =
    document.querySelectorAll("#faqs h2, div p");
```

How to create a JavaScript object

The syntax for creating an object

```
const variableName = new ObjectType();
```

A statement that creates a Date object

```
const today = new Date();
```

A few of the methods of the Date object

```
toDateString()
getFullYear()
getDate()
getMonth()
```

Examples that use a Date object

```
const today = new Date();
alert( today.toDateString() );
alert( today.getFullYear() );
alert( today.getDate() );
alert( today.getMonth() );
```

Properties and methods of a String object

One property

`length`

A few of the methods

`indexOf(`*`search`*`,`*`position`*`)`

`substr(`*`start`*`,`*`length`*`)`

`substring(`*`start`*`,`*`stop`*`)`

`toLowerCase()`

`toUpperCase()`

Examples that use a String object

```
const name = "Grace Hopper";
const nameUpper = name.toUpperCase();
const nameLength = name.length;        // nameLength = 12
const index = name.indexOf(" ");       // index = 5
const firstName = name.substr(0, index);
```

The syntax for an arrow function

```
const *constantName* = (*parameters*) => {
    *// statements that run when the function is executed*
};
```

The code for a function expression

```
const calculateTax = function(subtotal, taxRate) {
    const tax = subtotal * taxRate;
    return tax.toFixed(2);
};
```

The code rewritten as an arrow function

```
const calculateTax = (subtotal, taxRate) => {
    const tax = subtotal * taxRate;
    return tax.toFixed(2);
};
```

An arrow function with one parameter

```
const calculateTax = subtotal => {
    const tax = subtotal * 0.074;
    return tax.toFixed(2);
};
```

An arrow function with one parameter that executes one statement

```
const calculateTax = subtotal =>
    (subtotal * 0.074).toFixed(2);
```

An arrow function with no parameters that executes one statement

```
const getCurrentUrl = () => document.location.href;
```

How to call an arrow function

```
const url = getCurrentUrl();
```

Common events

Object	Event
`window`	`load`
`document`	`DOMContentLoaded`
`button`	`click`
`control` or `link`	`focus`
`blur`	
`control`	`change`
`select`	
`element`	`click`
`dblclick`	
`mouseover`	
`mousein`	
	`mouseout`

Code that attaches a click event handler for a button when the DOM is loaded

Using a function expression as the event handler

```
const processEntries = () => {
    // code that processes entries
};

document.addEventListener("DOMContentLoaded", () => {
    $("#calculate").addEventListener(
        "click", processEntries);
});
```

Using an anonymous function as the event handler

```
document.addEventListener("DOMContentLoaded", () => {

    $("#calculate").addEventListener("click", () => {
        // code that processes entries
    });

});
```

A property and a method of the Event object

Property

```
currentTarget
```

Method

```
preventDefault()
```

Common HTML elements that have default actions for the click event

Element	Default action for the click event
`<a>`	Load the page or image in the href attribute.
`<input>`	Submit the form if the type attribute is set to submit.
`<input>`	Reset the form if the type attribute is set to reset.
`<button>`	Submit the form if the type attribute is set to submit.
`<button>`	Reset the form if the type attribute is set to reset.

Testing vs. debugging

The goal of testing

To find all errors before the application is put into production.

The goal of debugging

To fix all errors before the application is put into production.

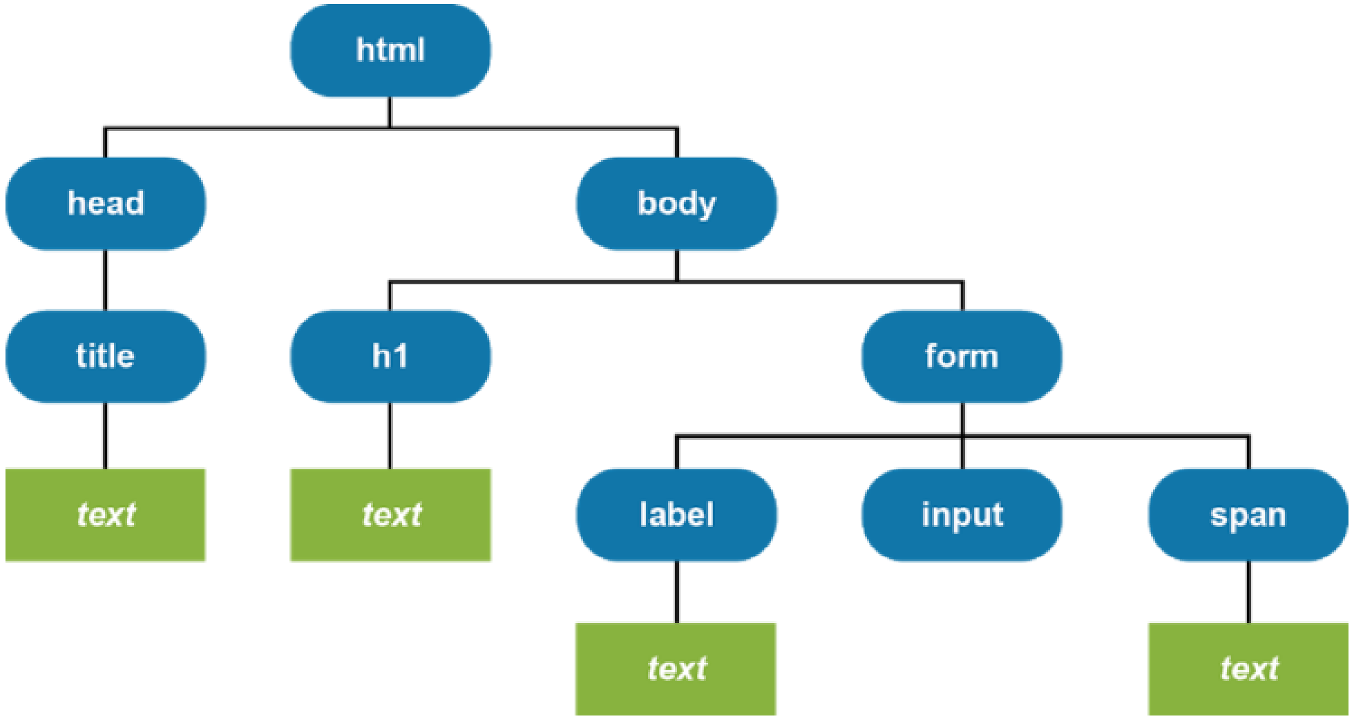
The three types of errors that can occur

Syntax errors

Runtime errors

Logic errors

The DOM for the web page



The DOM nodes commonly used

```
Document
Element
Attr
Text
```

Some of the properties of the Node interface

```
parentNode
childNodes
firstChild
lastChild
nextElementSibling
nodeValue
textContent
```

Typical properties available with the DOM HTML specification

Element	Property	Attribute
`all`	`id`	The id attribute
	`title`	The title attribute
	`className`	The class attribute. To set multiple class names, separate the names with spaces.
	`tagName`	The name of the tag, like div, h1, h2, a, or img.
`<a>`	`href`	The href attribute
`img`	`src`	The src attribute
	`alt`	The alt attribute
`input`	`disabled`	The disabled attribute

Some attributes of the HTML img element

```
src
alt
width
height
```

Two methods for working with a timer that calls a function once

```
setTimeout(function,delay)
clearTimeout(timer)
```

Two methods for working with a timer that calls a function repeatedly

```
setInterval(function,interval)
clearInterval(timer)
```

The download page for jQuery

https://jquery.com/download/

How to include jQuery 3.4.1 after you've downloaded it to your computer

```
<script src="jquery-3.4.1.min.js"></script>
```

How to include jQuery 3.4.1 from a CDN

```
<script
    src="https://code.jquery.com/jquery-3.4.1.min.js">
</script>
```

How to include the slim version of jQuery 3.4.1 from a CDN

```
<script
    src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
</script>
```

The syntax for a jQuery selector

```
$("selector")
```

How to select elements by element, id, and class

By element type: All <p> elements in the entire document

```
$("p")
```

By id: The element with “faqs” as its id

```
$("#faqs")
```

By class: All elements with “minus” as a class

```
$(".minus")
```

The syntax for calling a jQuery method

```
$("selector").methodName(parameters)
```

Some common jQuery methods

```
val()
val(value)
text()
text(value)
next([type])
submit()
focus()
```

The syntax for a jQuery event method

```
$(selector).eventMethodName( () => {
    // the statements of the event handler
});
```

Two common jQuery event methods

Event method	Description
`ready(handler)`	The event handler runs when the DOM is ready.
`click(handler)`	The event handler runs when the selected element is clicked.

A summary of the most useful jQuery selectors

```
[attribute]               :hidden
[attribute=value]         :last
:contains(text)           :last-child
:empty                    :lt(n)
:eq(n)                    :not(selector)
:even                     :nth-child
:first                    :odd
:first-child              :only-child
:gt(n)                    :parent
:has(selector)            :text
:header                   :visible
```

Some of the most useful jQuery event methods

```
ready(handler)
click(handler)
dblclick(handler)
mouseenter(handler)
mouseover(handler)
mouseout(handler)
hover(handlerIn,handlerOut)
```

Chapter 12

How to work with numbers, strings, and dates

Objectives (part 1)

Applied

1. Use the properties and methods of Number, String, and Date objects in your applications.
2. Use the random() method of the Math object to generate a random integer within a specific range.

Knowledge

1. Describe these special numerical values: Infinity, -Infinity, NaN, Number.MAX_VALUE, Number.MIN_VALUE, Number.MAX_SAFE_INTEGER, Number.MIN_SAFE_INTEGER, and Number.EPSILON.
2. Describe these methods of a Number object: toFixed() and toString().
3. Describe these static methods of the Number type: isNaN(), isFinite(), isInteger(), and isSafeInteger().

Objectives (part 2)

4. Describe these properties and methods of the Math object: PI, abs(), round(), ceil(), floor(), trunc(), pow(), sqrt(), min(), max(), and random().
5. Describe these properties and methods of a String object: length, charAt(), concat(), indexOf(), substr(), substring(), toLowerCase(), toUpperCase(), startsWith(), endsWith(), includes(), trimStart(), trimEnd(), trim(), padStart(), padEnd(), and repeat().
6. Describe how to use the split() method of a String object to create an array.
7. Describe how to create a Date object for the current date and time or for a specified date and time.

Objectives (part 3)

8. Describe these methods of the Date object: toString(), toDateString(), toTimeString(), getTime(), getFullYear(), getMonth(), getDate(), getDay(), getHours(), getMinutes(), getSeconds(), getMilliseconds(), setFullYear(), setMonth(), setDate(), setHours(), setMinutes(), setSeconds(), and setMilliseconds().
9. Describe the coding that's required to add or subtract dates.
10. Describe how to use the Internationalization API to format numbers and dates.

Static properties of the Number object

Property	Shortcut
`Number.MAX_VALUE`	
`Number.MIN_VALUE`	
`Number.MAX_SAFE_INTEGER`	
`Number.MIN_SAFE_INTEGER`	
`Number.POSITIVE_INFINITY`	`Infinity`
`Number.NEGATIVE_INFINITY`	`-Infinity`
`Number.EPSILON`	
`Number.NaN`	`NaN`

Examples of static properties of the Number object (part 1)

Example 1: Testing for Infinity, -Infinity, and NaN

```
if ( result == Infinity ) {
    alert( "The result is greater than " + Number.MAX_VALUE );
} else if ( result == -Infinity ) {
    alert( "The result is less than " + Number.MIN_VALUE );
} else if ( isNaN(result) ) {
    alert( "The result is not a number" );
} else {
    alert( "The result is " + result );
}
```

Example 2: Division by zero

```
alert(  0 / 0 );      // displays NaN
alert( 10 / 0 );      // displays Infinity
alert( -1 / 0 );      // displays -Infinity
```

Examples of static properties of the Number object (part 2)

Example 3: Safe and unsafe integers

```
const safe1 = Number.MAX_SAFE_INTEGER - 1;
const safe2 = Number.MAX_SAFE_INTEGER - 2;
const tooBig1 = Number.MAX_SAFE_INTEGER + 1;
const tooBig2 = Number.MAX_SAFE_INTEGER + 2;
alert( safe1 == safe2 );        // displays false
alert( tooBig1 == tooBig2 );    // displays true;
                                // ints not stored accurately
```

Instance methods of a Number object

```
toFixed(digits)
toString(base)
```

Examples of instance methods of the Number object

Example 1: Using the toFixed() method

```
const subtotal = 19.99, rate = 0.075;
const tax = subtotal * rate;                // tax is 1.49925
alert( tax.toFixed(2) );                    // displays 1.50
```

Example 2: Implicit use of the toString() method for base 10 conversions

```
const age = parseInt( prompt("Please enter your age.") );
alert( "Your age is " + age );
```

Static methods of a Number object

```
Number.isNaN(value)
Number.isFinite(value)
Number.isInteger(value)
Number.isSafeInteger(value)
```

Examples of static methods of a Number object (part 1)

Example 1: isNaN() vs Number.isNaN()

```
alert( isNaN("four") );                   // displays true
alert( isNaN([1,2,3]) );                  // displays true
alert( isNaN(NaN) );                      // displays true

alert( Number.isNaN("four") );            // displays false
alert( Number.isNaN([1,2,3]) );           // displays false
alert( Number.isNaN(NaN) );               // displays true
```

Example 2: Using the isFinite() method

```
alert( Number.isFinite( 10 / 0 ) );      // displays false
alert( Number.isFinite( -1 / 0 ) );      // displays false
alert( Number.isFinite(200) );            // displays true
```

Examples of static methods of a Number object (part 2)

Example 3: Using the isInteger() and isSafeInteger() methods

```
const tooBig = Number.MAX_SAFE_INTEGER + 1;

alert( Number.isInteger(tooBig) );      // displays true
alert( Number.isSafeInteger(tooBig) );  // displays false
alert( Number.isSafeInteger(tooBig - 1) );
                                        // displays true
```

One static property of the Math object

```
Math.PI
```

Example of the PI property

```
const area = Math.PI * 3 * 3;
// area is 28.274333882308138
```

Common static methods of the Math object

```
Math.abs(x)
Math.round(x)
Math.ceil(x)
Math.floor(x)
Math.trunc(x)
Math.pow(x, power)
Math.sqrt(x)
Math.min(x1, x2, ...)
Math.max(x1, x2, ...)
```

Examples of common methods of the Math object (part 1)

Example 1: The abs() method

```
const result_1a = Math.abs(-3.4);     // result_1a is 3.4
```

Example 2: The round() method

```
const result_2a = Math.round(12.5);   // result_2a is 13
const result_2b = Math.round(-3.4);   // result_2b is -3
const result_2c = Math.round(-3.5);   // result_2c is -3
const result_2d = Math.round(-3.51);  // result_2d is -4
```

Example 3: The floor(), ceil(), and trunc() methods

```
const result_3a = Math.floor(12.5);   // result_3a is 12
const result_3b = Math.ceil(12.5);    // result_3b is 13
const result_3c = Math.trunc(12.5);   // result_3c is 12
const result_3d = Math.floor(-3.4);   // result_3d is -4
const result_3e = Math.ceil(-3.4);    // result_3e is -3
const result_3f = Math.trunc(-3.4);   // result_3f is -3
```

Examples of common methods of the Math object (part 2)

Example 4: The pow() and sqrt() methods

```
const result_4a = Math.pow(2,3);         // result_4a is 8
const result_4b = Math.pow(125, 1/3);    // result_4b is 5
const result_4c = Math.sqrt(16);         // result_4c is 4
```

Example 5: The min() and max() methods

```
const result_5a = Math.max(12.5, -3.4);
                                      // result_5a is 12.5
const result_5b = Math.min(12.5, -3.4);
                                      // result_5b is -3.4
```

How to use the exponentiation operator instead of Math.pow()

```
const result_7a = 2 ** 3;      // same as Math.pow(2,3)
```

The random() method of the Math object

```
Math.random()
```

How to generate a random number

```
const result = Math.random();
```

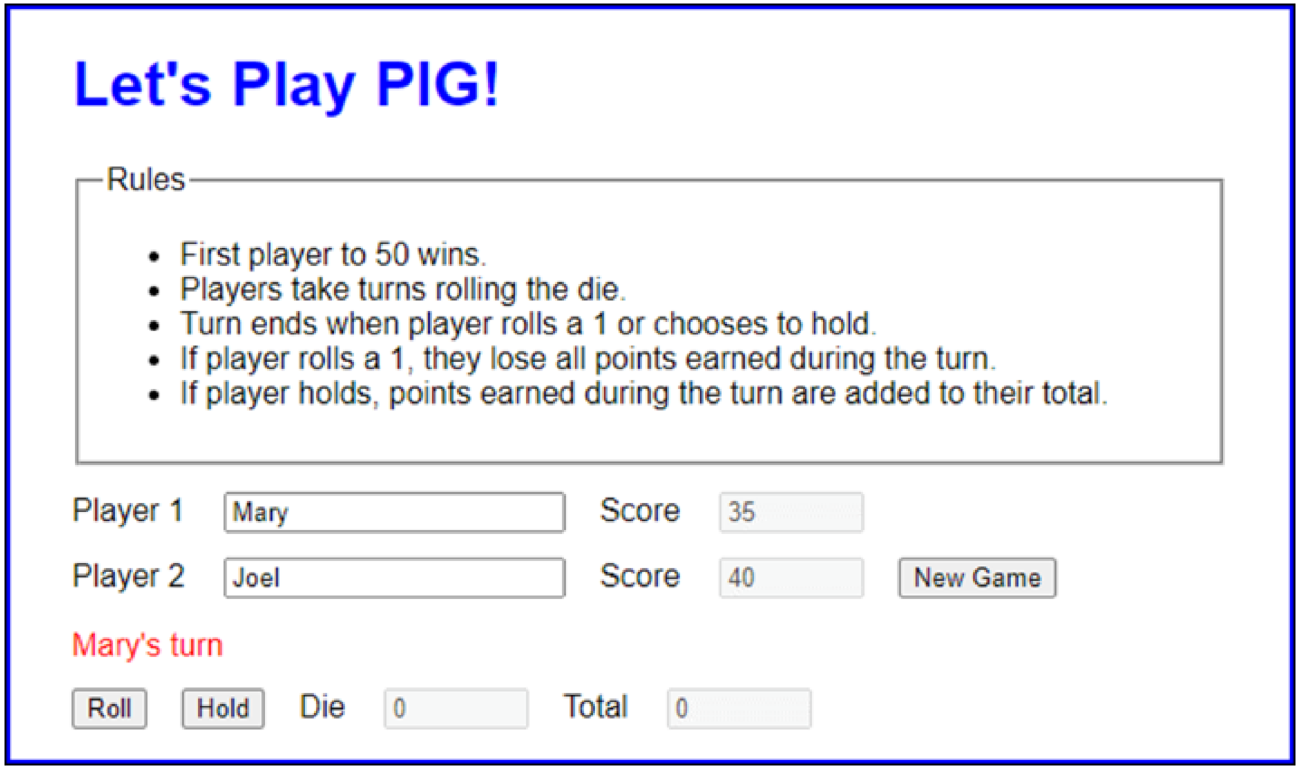
A function that generates a random number

```
const getRandomNumber = max => {
    let random = null;

    if (!isNaN(max)) {
        // value >= 0.0 and < 1.0
        random = Math.random();

        // value is an integer between 0 and max - 1
        random = Math.floor(random * max);

        // value is an integer between 1 and max
        random = random + 1;
    }
    // if max is not a number, will return null
    return random;
};
```

A statement that calls the getRandomNumber() function

```
// returns an integer that ranges from 1 through 100
const randomNumber = getRandomNumber(100);
```

The user interface for the PIG application



Let's Play PIG!

Rules

- First player to 50 wins.
- Players take turns rolling the die.
- Turn ends when player rolls a 1 or chooses to hold.
- If player rolls a 1, they lose all points earned during the turn.
- If player holds, points earned during the turn are added to their total.

Player 1	Mary	Score	35	
Player 2	Joel	Score	40	New Game

Mary's turn

Die 0 Total 0

The CSS for the PIG application

```
label {
    display: inline-block;
    margin-right: 1em;
}
input {
    margin-top: 1em;
    margin-right: 1em;
}
#turn div {
    color: red;
    margin-top: 1em;
}
#score1, #score2, #die, #total {
    width: 5em;
}
.hide {
    display: none;
}
```

The HTML for the PIG application (part 1)

```
<body>
    <main>
        <h1>Let's Play PIG!</h1>
        <fieldset>
            <legend>Rules</legend>
            <ul>
                <li>First player to <span id="winning_total">
                    </span> wins.</li>
                    ...
            </ul>
        </fieldset>

        <label for="player1">Player 1</label>
        <input type="text" id="player1" >
        <label for="score1">Score</label>
        <input type="text" id="score1" value="0" disabled><br>

        <label for="player2">Player 2</label>
        <input type="text" id="player2">
        <label for="score2">Score</label>
        <input type="text" id="score2" value="0" disabled>

        <input type="button" id="new_game" value="New Game"><br>
```

The HTML for the PIG application (part 2)

```
        <section id="turn" class="hide">
            <div><span id="current"></span>'s turn</div>

            <input type="button" id="roll" value="Roll">
            <input type="button" id="hold" value="Hold">

            <label for="die">Die</label>
            <input type="text" id="die" disabled>

            <label for="total">Total</label>
            <input type="text" id="total" disabled>
        </section>
    </main>
    <script
        src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
    </script>
    <script src="pig.js"></script>
</body>
```

The JavaScript for the PIG application (part 1)

```
"use strict";
const winningTotal = 50;

const getRandomNumber = max => {
    let rand = null;
    if (!isNaN(max)) {
        rand = Math.random();
        rand = Math.floor(rand * max);
        rand = rand + 1;
    }
    return rand;
};

const changePlayer = () => {
    if ( $("#current").text() == $("#player1").val() ) {
        $("#current").text( $("#player2").val() );
    } else {
        $("#current").text( $("#player1").val() );
    }
    $("#die").val("0");
    $("#total").val("0");
    $("#roll").focus();
};
```

The JavaScript for the PIG application (part 2)

```
$( document ).ready( () => {
    $("#new_game").click( () => {
        $("#score1").val("0");
        $("#score2").val("0");

        if ( $("#player1").val() == "" ||
             $("#player2").val() == "" ) {
            alert("Please enter two player names.");
        } else {
            $("#turn").removeClass("hide");
            changePlayer();
        }
    });

    $("#roll").click( () => {
        let total = parseInt( $("#total").val() );
        const die = getRandomNumber(6);

        if (die == 1) {
            total = 0;
            changePlayer();
        } else {
            total = total + die;
        }
```

The JavaScript for the PIG application (part 3)

```
            $("#die").val(die);
            $("#total").val(total);
        });

        $("#hold").click( () => {
            let score = 0;
            const total = parseInt( $("#total").val() );

            if ( $("#current").text() == $("#player1").val() ) {
                score = $("#score1");
            } else {
                score = $("#score2");
            }

            score.val( parseInt( score.val() ) + total );
            if (score.val() >= winningTotal) {
                alert( $("#current").text() + " WINS!" );
            } else {
                changePlayer();
            }
        });
        $("#winning_total").text(winningTotal);
        $("#player1").focus();
    });
```

One property of a String object

```
length
```

Example of the length property

```
const message = "JavaScript";
const result_1 = message.length;        // result_1 is 10
```

Methods of a String object

```
charAt(position)
concat(string1,string2, ...)
indexOf(search[,start])
substr(start,length)
substring(start)
substring(start,end)
toLowerCase()
toUpperCase()
```

Examples of methods of String objects (part 1)

Constant used by the following examples

```
const message = "JavaScript";
```

Example 1: The charAt() method

```
const letter = message.charAt(4);     // letter is "S"
```

Example 2: The concat() method

```
const result_3 = message.concat(" rules");
                     // result_3 is "JavaScript rules"
```

Example 3: The indexOf() method

```
const result_4a = message.indexOf("a");
                                      // result_4a is 1
const result_4b = message.indexOf("a", 2);
                                      // result_4b is 3
const result_4c = message.indexOf("s");
                                      // result_4c is -1
```

Examples of methods of String objects (part 2)

Example 4: The substr() and substring() methods

```
const result_5a = message.substr(4, 5);
                          // result_5a is "Scrip"
const result_5b = message.substring(4);
                          // result_5b is "Script"
const result_5c = message.substring(0, 4);
                          // result_5c is "Java"
```

Example 5: The toLowerCase() and toUpperCase() methods

```
const result_6a = message.toLowerCase();
                          // result_6a is "javascript"
const result_6b = message.toUpperCase();
                          // result_6b is "JAVASCRIPT"

// compare two strings ignoring case
alert( result_6a.toLowerCase() ==
    result_6b.toLowerCase() )          // displays true
```

More methods of a String object

```
startsWith(string[,start])
endsWith(string[,length])
includes(string[,start])
trimStart()
trimEnd()
trim()
padStart(length[,string])
padEnd(length[,string])
repeat(times)
```

More examples of methods of String objects (part 1)

Constants used by the following examples

```
const str = "jQuery";
const len = str.length;
```

Example 1: The startsWith(), endsWith(), and includes() methods

```
const result_1a = str.startsWith("j");     // result_1a is true
const result_1b = str.startsWith("j", 2);  // result_1b is false
const result_1c = str.endsWith("ry");      // result_1c is true
const result_1d = str.endsWith("ry", 2);   // result_1d is false
const result_1e = str.includes("Que");     // result_1e is true
const result_1f = str.includes("Que", 2);  // result_1f is false
```

More examples of methods of String objects (part 2)

Example 2: The padStart() and padEnd() methods

```
const result_2a = str.padStart(len + 3);
                              // result_2a is "   jQuery"
const result_2b = str.padEnd(len + 3, ".");
                              // result_2b is "jQuery..."
```

Example 3: The trimStart(), trimEnd(), and trim() methods

```
const str_2 = str.padStart(len + 3).padEnd(len + 6);
                              // str_2 is "   jQuery   "
const result_3a = str_2.trimStart();
                              // result_3a is "jQuery   "
const result_3b = str_2.trimEnd();
                              // result_3b is "   jQuery"
const result_3c = str_2.trim();
                              // result_3c is "jQuery"
```

Example 4: The repeat() method

```
const result_4a = "Hey ".repeat(3);
                              // result_4a is "Hey Hey Hey "
```

A String method that creates an array

```
split(separator[, limit])
```

Two constants that are used by the following examples

```
const fullName = "Grace M Hopper";
const date = "7-4-2021";
```

How to split a string that's separated by spaces into an array

```
const words = fullName.split(" ");
                    // words is ["Grace", "M", "Hopper"]
const len = words.length;                         // len is 3
const lastName = words[len - 1]; // lastName is "Hopper"
```

How to split a string that's separated by hyphens into an array

```
const dateParts = date.split("-");
                 // dateParts is ["7", "4", "2021"]
const month = dateParts[0];      // month is "7"
const year = dateParts[2];       // year is "2021"
```

How to split a string into an array of characters

```
const characters = fullName.split("");
```

How to get just one element from a string

```
const firstName = fullName.split(" ", 1);
                                // firstName is ["Grace"]
const len = firstName.length;            // len is 1
```

A statement that uses the firstName constant like a regular string

```
const greeting = `Hello, ${firstName}!`;
                                // greeting is "Hello, Grace!"
```

How it works if the string doesn't contain the separator

```
const dateParts = date.split("/");
                               // dateParts is ["7-4-2021"]
len = dateParts.length;        // len is 1
```

How it works if the string contains the separator at the beginning or end

```
const path = "/directory/";
const pathParts = path.split("/");
                      // pathParts is ["", "directory", ""]
```

Summary of how the Date constructor works

Parameters	Description
None	Creates a new Date object set to the current date and time.
String value	Creates a new Date object set to the date and time of the string.
Numeric values	Creates a new Date object set to the year, month, day, hours, minutes, seconds, and milliseconds of the numbers. Year and month are required.
Date object	Creates a new Date object that's a copy of the Date object it received.
Invalid values	Creates a new Date object that contains "Invalid Date".

How to create a Date object that represents the current date and time

```
const now = new Date();
```

How to create a Date object by specifying a date string

```
const electionDay = new Date("11/3/2020");
const grandOpening = new Date("2/16/2021 8:00");
const departureTime = new Date("4/6/2021 18:30:00");
```

How to create a Date object by specifying numeric values

Syntax of the constructor for the Date object

```
new Date( year, month, day, hours, minutes, seconds,
          milliseconds )
```

Examples

```
const electionDay = new Date(2020, 10, 3);
                                     // 10 is November
const grandOpening = new Date(2021, 1, 16, 8);
                                     // 1 is February
const departureTime = new Date(2021, 3, 6, 18, 30);
                                     // 3 is April
```

How to create a Date object by copying another Date object

```
const invoiceDate = new Date("8/8/2021");
const dueDate = new Date( invoiceDate );
// You can then add a number of days to due_date.
```

What happens when an invalid date is passed to the Date Constructor

```
const leapDate = new Date("2/29/2021");  // Invalid Date
```

Date strings that can lead to unexpected results in some browsers

```
const electionDay = new Date("11-3-2020");
                                         // Invalid Date
const electionDay = new Date("11/3/20");
                                         // 11/3/1920
```

How to check whether an object is a Date object

```
if (electionDay instanceof Date) { ... }
```

The formatting methods of a Date object

```
toString()
toDateString()
toTimeString()
```

Examples of the formatting methods

```
const birthday = new Date( 2001, 0, 7, 8, 25);

alert( birthday.toString() );
                   // "Sun Jan 07 2001 08:25:00 GMT-0800"

alert( birthday.toDateString() );  // "Sun Jan 07 2001"

alert( birthday.toTimeString() );  // "08:25:00 GMT-0800"
```

The get methods of a Date object

```
getTime()
getFullYear()
getMonth()
getDate()
getDay()
getHours()
getMinutes()
getSeconds()
getMilliseconds()
```

The set methods of a Date object

```
setFullYear(year)
setMonth(month)
setDate(day)
setHours(hour)
setMinutes(minute)
setSeconds(second)
setMilliseconds(ms)
```

How to display the date in your own format

```
const departTime = new Date(2021, 3, 16, 18, 30);
// April 16, 2021 6:30pm

const year = departTime.getFullYear();
// add 1 since months start at 0
const month = departTime.getMonth() + 1;
const day = departTime.getDate();

let dateText = year + "-";
// pad month if 1 digit
dateText += month.toString().padStart(2, "0") + "-";
// pad day if 1 digit
dateText += day.toString().padStart(2, "0");

// final dateText is "2021-04-16"
```

How to calculate the days until the New Year

```
const now = new Date();               // get current date and time
const newYear = new Date(now);        // copy current date and time
newYear.setMonth(0);                  // set month to January
newYear.setDate(1);                   // set day to the 1st
newYear.setFullYear( newYear.getFullYear() + 1 );

// time in milliseconds
const timeLeft = newYear.getTime() - now.getTime();
// milliseconds in a day: hrs * mins * secs * milliseconds
const msInOneDay = 24 * 60 * 60 * 1000;
// convert milliseconds to days
const daysLeft = Math.ceil( timeLeft / msInOneDay );

let message = "There ";
if (daysLeft == 1) {
    message += "is one day";
}
else {
    message += "are " + daysLeft + " days";
}
message += " left until the New Year.";

// If today is November 3, 2020, message is
// "There are 59 days left until the New Year."
```

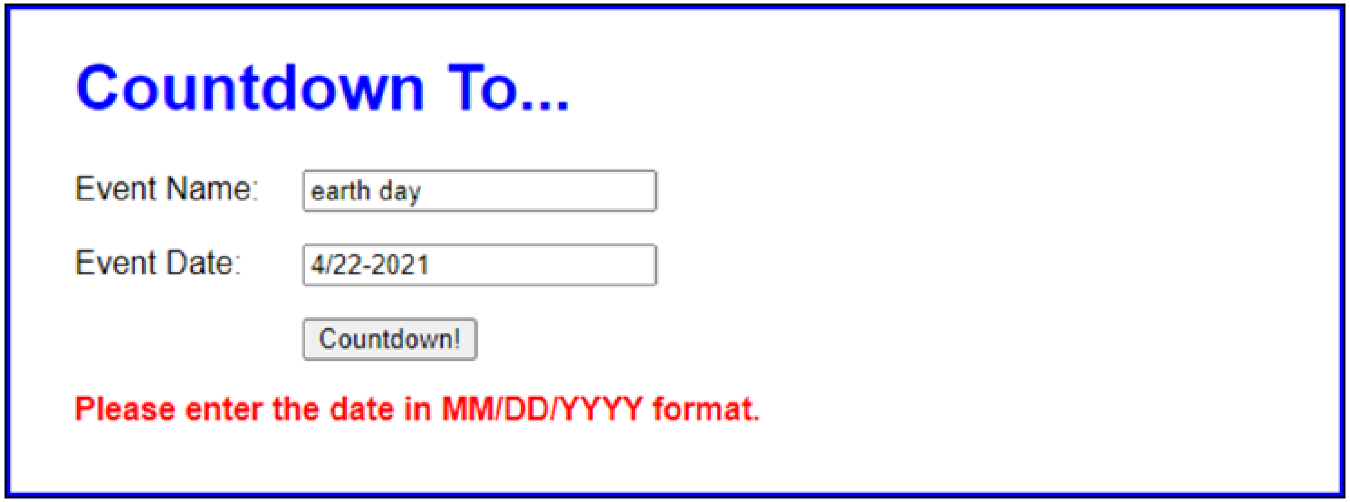
How to calculate a due date

```
const invoiceDate = new Date();
const dueDate = new Date( invoiceDate );
dueDate.setDate( dueDate.getDate() + 21 );
// due date is 3 weeks later
```

How to find the end of the month

```
const endOfMonth = new Date();

// Set the month to next month
endOfMonth.setMonth( endOfMonth.getMonth() + 1 );

// Set the date to one day before the start of the month
endOfMonth.setDate( 0 );
```

The user interface for the Count Down application



Event Name: earth day

Event Date: 4/22-2021

Please enter the date in MM/DD/YYYY format.

The HTML for the Count Down application (part 1)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, initial-scale=1">
    <title>Count Down</title>
    <link rel="stylesheet" href="count_down.css">
</head>

<body>
    <main>
        <h1>Countdown To...</h1>
        <div>
            <label for="event">Event Name:</label>
            <input type="text" name="event" id="event">
        </div>
        <div>
            <label for="date">Event Date:</label>
            <input type="text" name="date" id="date"><br>
        </div>
```

The HTML for the Count Down application (part 2)

```
        <div>
            <label>&nbsp;</label>
            <input type="button" name="countdown"
                   id="countdown" value="Countdown!">
        </div>
        <div><label id="message"></label></div>
    </main>

    <script
        src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
    </script>
    <script src="count_down.js"></script>
</body>
</html>
```

The CSS style rule for the label element

```
#message {
    color: red;
    font-weight: bold;
}
```

The JavaScript for the Count Down app (part 1)

```
"use strict";

$( document ).ready( () => {

    $("#countdown").click( () => {
        const eventName = $("#event").val();
        const eventDate = $("#date").val();
        const messageLbl = $("#message");

        // make sure user enters an event name and and date
        if (eventName.length == 0 || eventDate.length == 0) {
            messageLbl.text(
                "Please enter both a name and a date." );
            return;
        }

        // make sure event date string has two slashes
        const dateParts = eventDate.split("/");
        if (dateParts.length != 3) {
            messageLbl.text(
                "Please enter the date in MM/DD/YYYY format." );
            return;
        }
```

The JavaScript for the Count Down app (part 2)

```
// make sure event date string has a 4-digit year
const year = eventDate.substring(eventDate.length - 4);
if (isNaN(year)) {
    messageLbl.text(
        "Please enter the date in MM/DD/YYYY format." );
    return;
}

// convert event date string to Date object
// and check for validity
let date = new Date(eventDate);
if (date == "Invalid Date") {
    messageLbl.text(
        "Please enter the date in MM/DD/YYYY format." );
    return;
}
```

The JavaScript for the Count Down app (part 3)

```
// capitalize each word of event name
let formattedName = "";
const words = eventName.split(" ");
for (const i in words) {
    const firstLetter =
        words[i].substring(0,1).toUpperCase();
    const word = firstLetter +
                 words[i].substring(1).toLowerCase();
    formattedName += word.padEnd(word.length + 1);
}
formattedName = formattedName.trimEnd();

// calculate days
const today = new Date();
const msFromToday = date.getTime() - today.getTime();
const msForOneDay = 24 * 60 * 60 * 1000;
let daysToDate = Math.ceil( msFromToday / msForOneDay );
```

The JavaScript for the Count Down app (part 4)

```
        // create and display message
        let msg = "";
        date = date.toDateString();
        if (daysToDate == 0) {
            msg = `Hooray! Today is ${formattedName}! (${date})`;
        }
        else if (daysToDate > 0) {
            msg = `${daysToDate} day(s) until ${formattedName}!
                  (${date})`;
        }
        else if (daysToDate < 0) {
            daysToDate = Math.abs(daysToDate);
            msg = `${formattedName} happened ${daysToDate} day(s)
                  ago. (${date})`;
        }
        messageLbl.text(msg);    });
    });

    $("#event").focus();
});
```

Two of the constructors of the Intl object

```
NumberFormat(locale[, options])
DateTimeFormat(locale[, options])
```

A method the the NumberFormat and DateTimeFormat objects

```
format(object)
```

How to use NumberFormat objects to format numbers

```
const us = new Intl.NumberFormat( "en-US" );
const de = new Intl.NumberFormat( "de-DE" );

const result_1a = us.format(1234567.89);
// result_1a is 1,234,567.89

const result_1b = de.format(1234567.89);
// result_1b is 1.234.567,89
```

How to use NumberFormat objects to format currency

```
const us = new Intl.NumberFormat(
    "en-US", {style:"currency", currency:"USD"} );
const gb = new Intl.NumberFormat(
    "en-GB", {style:"currency", currency:"GBP"} );
const de = new Intl.NumberFormat(
    "de-DE", {style:"currency", currency:"EUR"} );

const result_2a = us.format(100200300.40);
// result_2a is $100,200,300.40

const result_2b = gb.format(100200300.40);
// result_2b is £100,200,300.40

const result_2c = de.format(100200300.40);
// result_2c is 100.200.300,40 €
```

How to use DateTimeFormat objects to format dates

```
const dt = new Date("7/4/2021");

const us = new Intl.DateTimeFormat( "en-US" );
const de = new Intl.DateTimeFormat( "de-DE" );

const result_3a = us.format(dt); // result_3a is 7/4/2021
const result_3b = de.format(dt); // result_3b is 4.7.2021
```

A URL with more information about the Internationalization API

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl

Chapter 13

How to work with control structures, exceptions, and regular expressions

Objectives (part 1)

Applied

1. Use the identity operators in your control structures.
2. Use break and continue statements in your while, do-while, and for loops.
3. Use switch statements, including those that use fall through and default cases.
4. Use the conditional operator for simple logic requirements.
5. Use the AND and OR operators for selections.
6. Use try-catch statements to catch errors.
7. Create and throw Error objects.
8. Create regular expressions and use them to match patterns in strings.

Objectives (part 2)

Knowledge

1. Describe type coercion and distinguish between the equality and identity operators.
2. Describe the use of break and continue statements in loops.
3. Describe the use of a switch statement.
4. Describe the use of a conditional operator as a replacement for an if statement.
5. Describe how to use non-Boolean values in conditions.
6. Explain how the short-circuit evaluations of the AND and OR operators can be used in selections.
7. Describe the use of the try, catch, and finally blocks in a try-catch statement.

Objectives (part 3)

8. Describe the how to create and throw Error objects.
9. Describe the use of regular expressions for matching patterns with strings.

The equality operators

Operator	Description	Example
`==`	Equal	`lastName == "Hopper"`
`!=`	Not equal	`months != 0`

The identity operators

Operator	Description	Example
`===`	Equal	`lastName === "Hopper"`
`!==`	Not equal	`months !== 0`

Example 1: The break statement in a while loop

```
let number = 0;
while (true) {
    number = parseInt( prompt(
                 "Enter a number from 1 to 10.") );
    if ( isNaN(number) || number < 1 || number > 10 ) {
        alert("Invalid entry. Try again.");
    } else {
        break;
    }
}
alert(number);
```

Example 2: The continue statement in a while loop

```
let sum = 0;
let number = 0;
while ( number <= 40 ) {
    number++;
    if ( number % 5 !== 0 ) {
        continue;     // if number isn't divisible by 5
    }
    sum += number;
}
alert(sum);
// displays sum of 5, 10, 15, 20, 25, 30, 35, 40
```

A switch statement with a default case

```
switch ( letterGrade ) {
    case "A":
        message = "well above average";
        break;
    case "B":
        message = "above average";
        break;
    case "C":
        message = "average";
        break;
    case "D":
        message = "below average";
        break;
    case "F":
        message = "failing";
        break;
    default:
        message = "invalid grade";
        break;
}
```

A switch statement with fall through

```
switch ( letterGrade ) {
    case "A":
    case "B":
        message = "Scholarship approved.";
        break;
    case "C":
        message = "Application requires review.";
        break;
    case "D":
    case "F":
        message = "Scholarship not approved.";
        break;
}
```

A function that uses a switch statement to return a value

```
const getTimeOfDay = ampm => {
    switch( ampm.toUpperCase() ) {
        case "AM":
            return "morning";
        case "PM":
            return "evening";
        default:
            return "invalid";
    }
};
```

Syntax of the conditional operator

```
( conditional_expression ) ? value_if_true :
    value_if_false
```

Examples of using the conditional operator

Example 1: Setting a string based on a comparison

```
const message = ( age >= 18 ) ? "Can vote" : "Cannot vote";
```

Example 2: Calculating overtime pay

```
const overtime = ( hours > 40 ) ? ( hours - 40 ) * rate * 1.5 : 0;
```

Example 3: Selecting a singular or plural ending based on a value

```
const ending = ( errorCount == 1 ) ? "" : "s".
const message = "Found " + error_count + " error" + ending + ".";
```

Example 4: Setting a value to 1 if it's at a maximum value

```
let value = ( value == maxValue ) ? 1 : value + 1;
```

Example 5: Returning one of two values based on a comparison

```
return ( number > highest ) ? highest : number;
```

How conditional operators can be rewritten with if statements

Example 1 rewritten with an if statement

```
let message;
if ( age >= 18 ) {
    message = "Can vote";
} else {
    message = "Cannot vote";
}
```

Example 4 rewritten with an if statement

```
if ( value == maxValue ) {
    value = 1;
}
else {
    value = value + 1;
}
```

Non-Boolean values evaluated as false

`0`

"" (Empty string)

`null`

`undefined`

an empty array

Non-Boolean values evaluated as true

any number other than zero

any string that isn’t empty

any array that isn’t empty

any object

any symbol

Example 1: An if statement that checks if a variable is initialized

```
let val;
if (!val) {
    alert("Variable 'val' is not initialized");
}
```

Example 2: An if-else statement that checks for an empty string

```
const name = prompt("Please enter a name");
if (name) {
    alert(name);
} else {
    alert("You did not enter a name");
}
```

Example 3: An if statement that checks for the existence of a property

```
if (window.navigator.geolocation) {
     // code that uses the geolocation property
     // of the window object's Navigator object
}
```

Statements that store a true or false value

Example 1: Using the OR operator

```
const selected = state === "CA" || state === "NC";
```

Example 2: Using the AND operator

```
const canVote = age >= 18 && citizen;
```

Statements that store a value and provide a default value

Example 3: Using the OR operator

```
const name = prompt("Please enter a name") || "N/A";
```

Example 4: Using the nullish coalescing operator

```
const name = prompt("Please enter a name") ?? "N/A";
```

Statements that check for the existence of a property

Example 5: Using the AND operator

```
if (window && window.navigator &&
    window.navigator.geolocation) { ... }
```

Example 6: Using the optional chaining operator

```
if (window?.navigator?.geolocation) { ... }
```

Statements that store the value of a property or a default value if the property doesn’t exist

Example 7: Using the AND and OR operators

```
const initial = book && book.author &&
    book.author.middleInitial || "No Middle Initial";
```

Example 8: Using the nullish coalescing and optional chaining operators

```
const initial = book?.author?.middleInitial ??
    "No Middle Initial";
```

Terms related to control structures

equality operators

type coercion

identity operators

switch statement

switch expression

fall through

conditional expression

truthy value

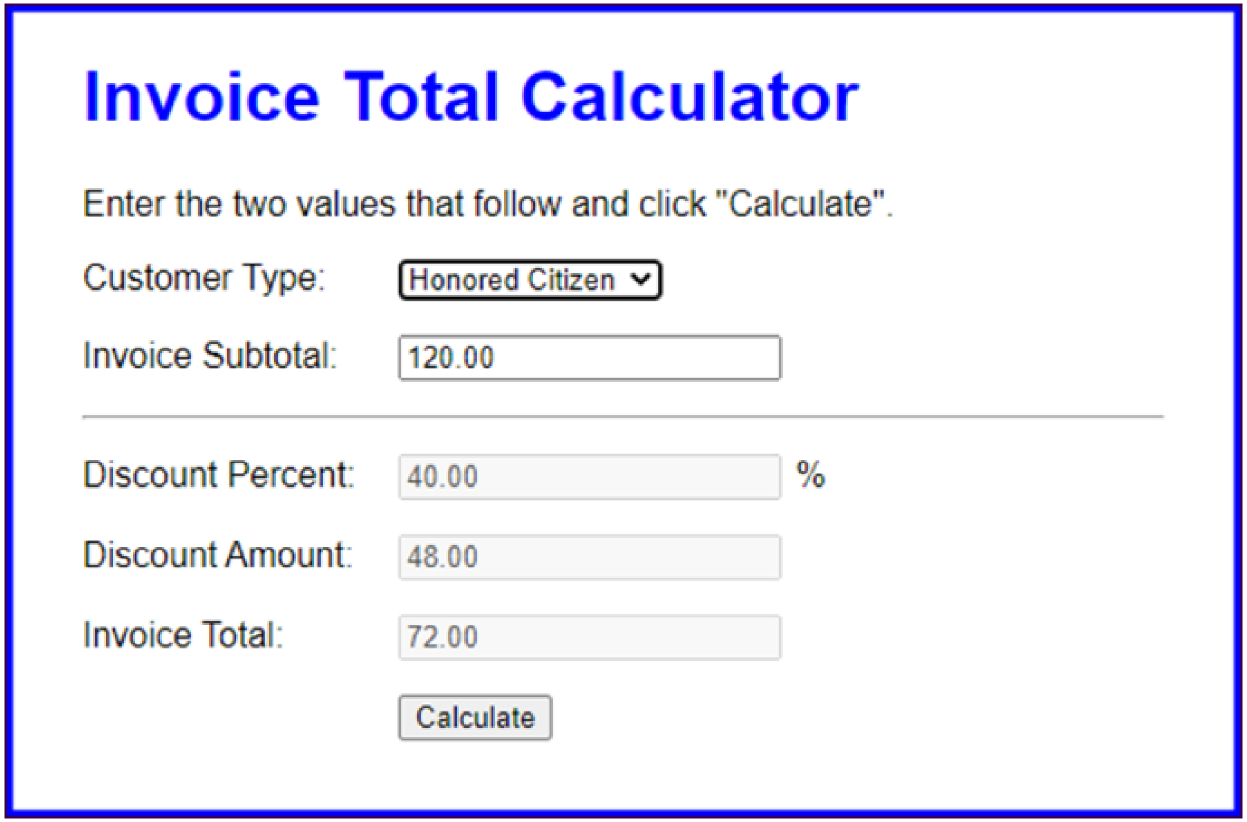
falsey value

short-circuit evaluation

nullish coalescing operator

optional chaining operator

The user interface for the Invoice application



Invoice Total Calculator

Enter the two values that follow and click "Calculate".

Customer Type:	Honored Citizen	
Invoice Subtotal:	120.00	
Discount Percent:	40.00	%
Discount Amount:	48.00	
Invoice Total:	72.00	

The HTML for the Invoice application (part 1)

```
<main>
    <h1>Invoice Total Calculator</h1>
    <p>Enter the two values that follow and click "Calculate".
    </p>
    <div>
        <label for="type">Customer Type:</label>
        <select id="type">
            <option value="reg">Regular</option>
            <option value="loyal">Loyalty Program</option>
            <option value="honored">Honored Citizen</option>
        </select>
    </div>
    <div>
        <label for="subtotal">Invoice Subtotal:</label>
        <input type="text" id="subtotal">
    </div>
    <hr>
    <div>
        <label for="percent">Discount Percent:</label>
        <input type="text" id="percent" disabled>%
    </div>
```

The HTML for the Invoice application (part 2)

```
        <div>
            <label for="discount">Discount Amount:</label>
            <input type="text" id="discount" disabled>
        </div>
        <div>
            <label for="total">Invoice Total:</label>
            <input type="text" id="total" disabled><br>
        </div>
        <div>
            <label>&nbsp;</label>
            <input type="button" id="calculate" value="Calculate">
        </div>
    </main>
```

The JavaScript for the Invoice application (part 1)

```
const calculateDiscount = (customer, subtotal) => {
    switch(customer) {
        case "reg":
            if (subtotal >= 100 && subtotal < 250) {
                return .1;
            } else if (subtotal >= 250 && subtotal < 500) {
                return  .25;
            } else if (subtotal >= 500) {
                return .3;
            } else {
                return 0;
            }
        case "loyal":
            return .3;
        case "honored":
            return (subtotal < 500) ? .4 : .5;
    }
};
```

The JavaScript for the Invoice application (part 2)

```
$( document ).ready( () => {

    $("#calculate").click( () => {
        const customerType = $("#type").val();
        let subtotal = $("#subtotal").val() || 0;  // default
        subtotal = parseFloat(subtotal);

        const discountPercent =
            calculateDiscount(customerType, subtotal);
        const discountAmount = subtotal * discountPercent;
        const invoiceTotal = subtotal - discountAmount;

        $("#subtotal").val( subtotal.toFixed(2) );
        $("#percent").val( (discountPercent * 100).toFixed(2) );
        $("#discount").val( discountAmount.toFixed(2) );
        $("#total").val(  invoiceTotal.toFixed(2) );

        // set focus on type drop-down when done
        $("#type").focus();
    });

    // set focus on type drop-down on initial load
    $("#type").focus();
});
```

The syntax for a try-catch statement

```
try { statements }
catch([errorName]) { statements }
[ finally { statements } ]
```

Two properties of Error objects

```
name

message
```

A try-catch statement for a calculateFV() function

```
const calculateFV = (investment, rate, years) => {
    try {
        let futureValue = investment;
        for (let i = 1; i <= years; i++ ) {
            futureValue += futureValue * rate / 100;
        }
        return futureValue.toFixed(2);
    }
    catch(error) {
        alert (error.name + ": " + error.message)
    }
};
```

A catch block that displays a custom message

```
catch(error) {
    alert("The calculateFV function has thrown an error." );
};
```

A catch block with no error parameter

```
catch {
    alert("The calculateFV function has thrown an error." );
};
```

The syntax for creating a new Error object

```
new Error(message)
```

The syntax for the throw statement

```
throw errorObject;
```

A calculateFV() method that throws a new Error object

```
const calculateFV = ( investment, rate, years ) => {
    if ( isNaN(investment) || investment <= 0 ) {
        throw new Error(
            "calculateFV requires investment greater than 0.");
    }
    if ( isNaN(rate) || rate <= 0 ) {
        throw new Error(
            "calculateFV requires annual rate greater than 0.");
    }
    let futureValue = investment;
    for (let i = 1; i <= years; i++ ) {
        futureValue += futureValue * rate / 100;
    }
    return futureValue.toFixed(2);
};
```

A try-catch statement that catches the Error object that has been thrown

```
try {
    $("future_value").text =
        calculateFV(investment, rate, years);
}
catch(error) {
    alert (error.name + ": " + error.message);
}
finally {
    $("investment").focus();
}
```

Some of the error types in the Error hierarchy

Type	Thrown when
`RangeError`	A numeric value has exceeded the allowable range
`ReferenceError`	A variable is read that hasn’t been defined
`SyntaxError`	A runtime syntax error is encountered
`TypeError`	The type of a value is different from what was expected

A statement that throws a RangeError object

```
throw new RangeError("Annual rate is invalid.");
```

Three reasons for using throw statements

To test the operation of a try-catch statement

To throw an error from a function that lets the calling code know that one or more of the arguments that were passed are invalid

To perform some processing after catching an error and then throw the error again

Terms related to exception handling

try-catch statement

exception

exception handling

try block

catch block

finally block

throw an exception

throw statement

Two ways to create a regular expression object

By using the RegExp() constructor

```
const constantName = new RegExp("expression"[, "flags"]);
```

By coding a regular expression literal

```
const constantName = /expression/[flags];
```

Two statements that create a regular expression that will find “Babbage”

By using the RegExp() constructor

```
const pattern = new RegExp("Babbage");
```

By coding a regular expression literal

```
const pattern = /Babbage/;
```

One method of a regular expression

```
test(string)
```

How to use the test() method

Two strings to test

```
const inventor = "Charles Babbage";
const programmer = "Ada Lovelace";
```

How to use the test() method to search for the pattern

```
alert( pattern.test(inventor) );     // displays true
alert( pattern.test(programmer) );   // displays false
```

How to create a case-insensitive regular expression

When using the RegExp() constructor

```
const pattern = new RegExp("lovelace", "i");
```

When coding a regular expression literal

```
const pattern = /lovelace/i;
```

How to use a case-insensitive regular expression

```
alert( pattern.test(programmer) );   // displays true
```

Special characters in regular expressions

Pattern	Matches
`\\`	Backslash character
`\/`	Forward slash
`\t`	Tab
`\n`	Newline
`\r`	Carriage return
`\f`	Form feed
`\v`	Vertical tab
`[\b]`	Backspace (the only special character that must be inside brackets)
`\u`*dddd*	The Unicode character whose value is the four hexadecimal digits.
`\x`*dd*	The Latin-1 character whose value is the two hexadecimal digits. Equivalent to `\u00`*dd*.

How to match special characters

```
const string =
    "©2020 MMA Inc.\nAll rights reserved (8/2020).";

alert( /\//.test(string) );
// matches / and displays true

alert( /\xA9/.test(string) );
// matches © and displays true
const pattern = new RegExp("\\\\");  // same as /\\/
alert( pattern.test(string) );
// displays false since there's no \
```

Types of characters in regular expressions

Pattern	Matches
`.`	Any character except a newline (use `\.` to match a period)
`[ ]`	Any character in the brackets (use `\[` or `\]` to match a bracket)
`[^ ]`	Any character not in the brackets
`[a-z]`	Any character in the range of characters when used inside brackets
`\w`	Any letter, number, or the underscore
`\W`	Any character that's not a letter, number, or the underscore
`\d`	Any digit
`\D`	Any character that's not a digit
`\s`	Any whitespace character (space, tab, newline, carriage return, form feed, or vertical tab)
`\S`	Any character that's not whitespace

How to match types of characters

```
const string = "The product code is MBT-3461.";

alert( /MB./.test(string) );             // displays true
alert( /MB[TF]/.test(string) );          // displays true
alert( /MBT-\W/.test(string) );          // displays false
```

String positions in regular expressions

Pattern	Matches
^	The beginning of the string (use `\^` to match a caret)
$	The end of the string (use `\$` to match a dollar sign)
\b	Word characters that aren't followed or preceded by a word character
\B	Word characters that are followed or preceded by a word character

How to match string positions

```
const inventor = "Charles Babbage";
alert( /^Charles/.test(inventor) );      // displays true
alert( /Babbage$/.test(inventor) );      // displays true
alert( /^Babbage/.test(inventor) );      // displays false

const programmer = "Ada Lovelace";
alert( /Ad/.test(programmer) );          // displays true
alert( /Ad\b/.test(programmer) );        // displays false
```

How to group and match subpatterns

Pattern	Matches
`(subpattern)`	Creates a subpattern (use `\(` and `\)` to match a parenthesis)
`|`	Matches either the left or right subpattern (use `\\|` to match a vertical bar)
`\n`	Matches the subpattern in the specified position

Examples

```
const name = "Rob Robertson";
alert( /^(Rob)|(Bob)\b/.test(name) );  // displays true
alert( /^(\w\w\w) \1/.test(name) );    // displays true
```

Repeating patterns in regular expressions

Pattern	Matches
`{`*`n`*`}`	Pattern must repeat exactly *n* times (use **`\{`** and **`\}`** to match a brace)
`{`*`n`*`,}`	Pattern must repeat *n* or more times
`{`*`n`*`,`*`m`*`}`	Subpattern must repeat from *n* to *m* times
`?`	Zero or one of the previous subpattern (same as {0,1})
`+`	One or more of the previous subpattern (same as {1,})
`*`	Zero or more of the previous subpattern (same as {0,})

How to match a repeating pattern

```
const phone = "559-555-6627";
const fax   = "(559) 555-6635";
alert( /^\d{3}-\d{3}-\d{4}$/.test(phone) );
                                        // displays true
alert( /^\(\d{3}\) ?\d{3}-\d{4}$/.test(fax) );
                                        // displays true

const phonePattern =
    /^(\d{3}-)|(\(\d{3}\) ?)\d{3}-\d{4}$/;
alert( phonePattern.test(phone) );      // displays true
alert( phonePattern.test(fax) );        // displays true
```

Regular expressions for testing validity

A pattern for testing phone numbers in this format: 999-999-9999

```
/^\d{3}-\d{3}-\d{4}$/
```

A pattern for testing credit card numbers in this format: 9999-9999-9999-9999

```
/^\d{4}-\d{4}-\d{4}-\d{4}$/
```

A pattern for testing zip codes in either of these formats: 99999 or 99999-9999

```
/^\d{5}(-\d{4})?$/
```

A pattern for testing emails in this format: username@mailserver.domain

```
/^[\w\.\-]+@[\w\.\-]+\.[a-zA-Z]+$/
```

A pattern for testing dates in this format: mm/dd/yyyy

```
/^[01]?\d\/[0-3]\d\/\d{4}$/
```

Examples that use these expressions

Testing a phone number for validity

```
const phone = "559-555-6624";           // valid number

const phonePattern = /^\d{3}-\d{3}-\d{4}$/;
if ( !phonePattern.test(phone) ) {
    alert("Invalid phone number");      // not displayed
}
```

Testing a date for a valid format, but not for a valid month, day, and year

```
const startDate = "8/10/220";           // invalid date
const datePattern = /^[01]?\d\/[0-3]\d\/\d{4}$/;
// this will match dates like 19/21/2020 and 9/39/2021
if ( !datePattern.test(startDate) ) {
    alert("Invalid start date");        // displayed
}
```

A function that does more complete validation of an email address

```
const isEmail = email => {
    if (email.length === 0) { return false; }
    const parts = email.split("@");
    if (parts.length !== 2) { return false; }
    if (parts[0].length > 64) { return false; }
    if (parts[1].length > 255) { return false; }

    const address =
        "(^[\\w!#$%&'*+/=?^`{|}~-]+(\\.[\\w!#$%&'*+/=?^`{|}~-]+)*$)";
    const quotedText = "(^\"(([^\\\\\"])|(\\\\[\\\\\"]))+\"$)";
    const localPart = new RegExp( address + "|" + quotedText );
    if ( !localPart.test(parts[0]) ) { return false; }

    const hostnames =
        "(([a-zA-Z0-9]\\.)|([a-zA-Z0-9][-a-zA-Z0-9]{0,62}[a-zA-Z0-9]\\.))+";
    const tld = "[a-zA-Z0-9]{2,6}";
    const domainPart = new RegExp("^" + hostnames + tld + "$");
    if ( !domainPart.test(parts[1]) ) { return false; }

    return true;
};
alert( isEmail("grace@yahoo.com") );       // displays true
alert( isEmail("grace@yahoocom") );        // displays false
```

Terms related to regular expressions

regular expression

pattern

regular expression object

regular expression literal

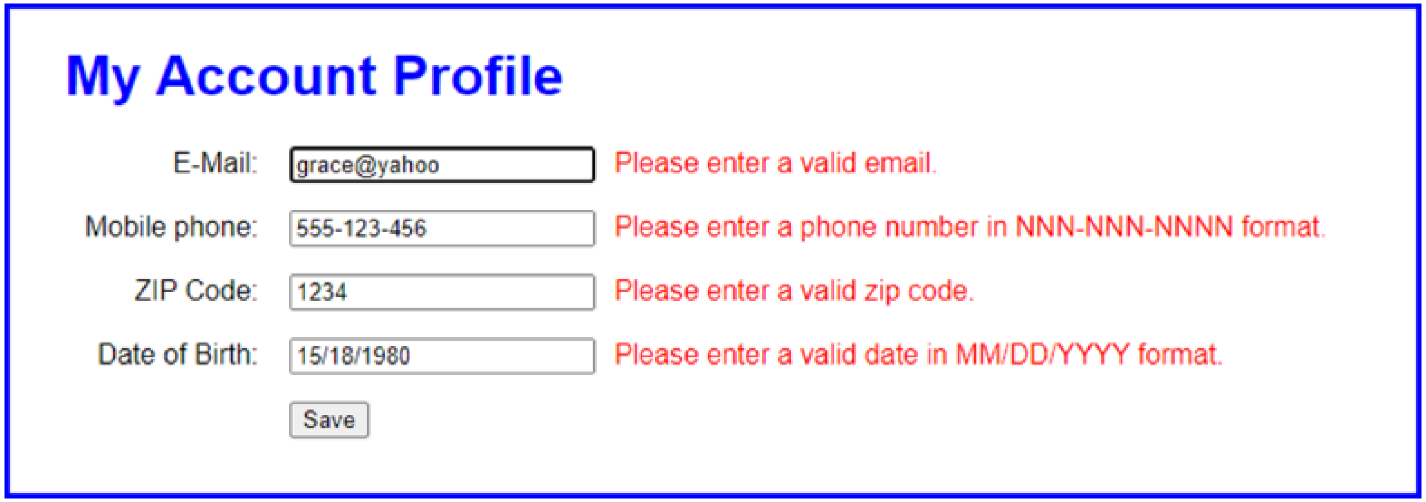
flag

escape character

subpattern

quantifier

The Account Profile app with error messages



My Account Profile

E-Mail:	grace@yahoo	Please enter a valid email.
Mobile phone:	555-123-456	Please enter a phone number in NNN-NNN-NNNN format.
ZIP Code:	1234	Please enter a valid zip code.
Date of Birth:	15/18/1980	Please enter a valid date in MM/DD/YYYY format.

The HTML for the Account Profile app (part 1)

```
<main>
    <h1>My Account Profile</h1>

    <div>
        <label for="email">E-Mail:</label>
        <input type="text" name="email" id="email">
        <span></span>
    </div>

    <div>
        <label for="phone">Mobile phone:</label>
        <input type="text" name="phone" id="phone">
        <span></span>
    </div>

    <div>
        <label for="zip">ZIP Code:</label>
        <input type="text" name="zip" id="zip">
        <span></span>
    </div>
```

The HTML for the Account Profile app (part 2)

```
        <div>
            <label for="dob">Date of Birth:</label>
            <input type="text" name="dob" id="dob">
            <span></span>
        </div>

        <div>
            <label></label>
            <input type="button" id="save" value="Save">
        </div>
    </main>
```

The JavaScript for the Account Profile app (part 1)

```
const isDate = (date, datePattern) => {
    if (!datePattern.test(date)) { return false; }

    const dateParts = date.split("/");
    const month = parseInt( dateParts[0] );
    const day = parseInt( dateParts[1] );

    if ( month < 1 || month > 12 ) { return false; }
    if ( day > 31 ) { return false; }
    return true;
};

$( document ).ready( () => {
    $( "#save" ).click( () => {
        $("span").text("");

        // get values entered by user
        const email = $("#email").val();
        const phone = $("#phone").val();
        const zip = $("#zip").val();
        const dob = $("#dob").val();
```

The JavaScript for the Account Profile app (part 2)

```
// regular expressions for validity testing
const emailPattern = /^[\w\.\-]+@[\w\.\-]+\.[a-zA-Z]+$/;
const phonePattern = /^\d{3}-\d{3}-\d{4}$/;
const zipPattern = /^\d{5}(-\d{4})?$/;
const datePattern = /^[01]?\d\/[0-3]\d\/\d{4}$/;

// check user entries for validity
let isValid = true;
if ( email === "" || !emailPattern.test(email) ) {
    isValid = false;
    $("#email").next().text(
        "Please enter a valid email.");
}
if ( phone === "" || !phonePattern.test(phone) ) {
    isValid = false;
    $("#phone").next().text(
        "Please enter a phone number " +
        "in NNN-NNN-NNNN format.");
}
if ( zip === "" || !zipPattern.test(zip) ) {
    isValid = false;
    $("#zip").next().text(
        "Please enter a valid zip code.");
}
```

The JavaScript for the Account Profile app (part 3)

```
        if ( dob === "" || !isDate(dob, datePattern) ) {
            isValid = false;
            $("#dob").next().text(
                "Please enter a valid date " +
                "in MM/DD/YYYY format.");
        }
        if ( isValid ) {
            // code that saves profile info goes here
        }
        $("#email").focus();
    });

    // set focus on initial load
    $("#email").focus();
});
```

Chapter 14

How to work with browser objects, cookies, and web storage

Objectives (part 1)

Applied

1. Use the properties and methods of the location and history objects in your applications.
2. Use cookies in your applications.
3. Use web storage in your applications.
4. Use Chrome's developer tools to work with cookies and web storage.

Knowledge

1. Describe these properties and methods of the location object: href, reload(), and replace().
2. Describe these properties and methods of the history object: length, back(), forward(), and go().
3. Describe the use of cookies.

Objectives (part 2)

4. Distinguish between session and persistent cookies.
5. Describe the use of web storage.
6. Distinguish between session storage and local storage.

A URL with search parameters

```
http://www.murach.com:8181/javascript/
location.html?first=G&last=Hopper#result
```

Properties of the location object

Property	Value in the URL above
`href`	Complete URL
`protocol`	`http:`
`hostname`	`www.murach.com`
`port`	`8181`
`host`	`www.murach.com:8181`
`path`	`/javascript/location.html`
`search`	`?first=G&last=Hopper`
`hash`	`#result`

Methods of the location object

Method	Description
reload(*force*)	Reloads the current webpage.
replace(*url*)	Loads a new page in the browser and replaces the current page in the history list.

How to load a new web page

```
location.href = "https://www.murach.com";
location = "https://www.murach.com";
```

How to reload a web page

```
location.reload();
// reloads the current page from the cache

location.reload(true);
// reloads the current page from the server
```

How to load a new page and overwrite the current history page

```
location.replace("https://www.murach.com");
```

One property of the history object

Property	Description
`length`	The number of URLs in the history object

Methods of the history object

Method	Description
`back()`	Goes back one step in the URL history
`forward()`	Goes forward one step in the URL history
`go(`***`position`***`)`	Goes forward or back the specified number of steps in the URL history
`go(`***`substring`***`)`	Goes to the most recent URL in the history that contains the substring

How to use the back() method

```
history.back();
```

How to use the forward() method

```
history.forward();
```

How to use the go() method

Go forward two URLs

```
history.go(2);
```

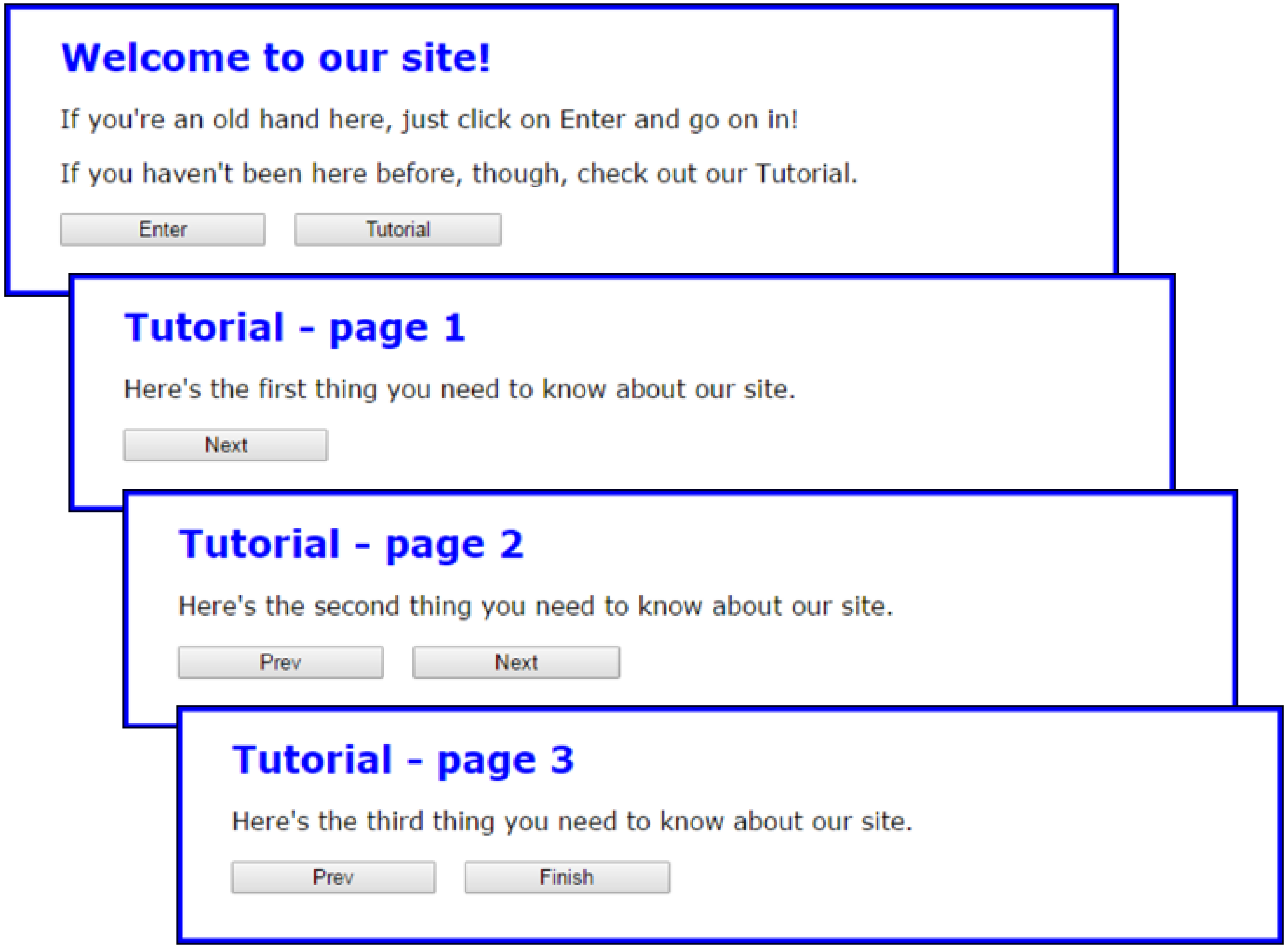
Go back three URLs

```
history.go(-3);
```

Go to the most recent URL that contains "google"

```
history.go("google");
```

The four pages of the Tutorial application



Welcome to our site!

If you're an old hand here, just click on Enter and go on in!

If you haven't been here before, though, check out our Tutorial.

Tutorial - page 1

Here's the first thing you need to know about our site.

Tutorial - page 2

Here's the second thing you need to know about our site.

Tutorial - page 3

Here's the third thing you need to know about our site.

The HTML in the main element of the index.html file

```
<h1>Welcome to our site!</h1>
<p>If you're an old hand here, just click on Enter and go
   on in!</p>
<p>If you haven't been here before, though, check out our
   Tutorial.</p>
<input type="button" id="enter" value="Enter">
<input type="button" id="tutorial" value="Tutorial">
```

The HTML in the main element of the tutorial files

tutorial1.html

```
<h1>Tutorial - page 1</h1>
<p>Here's the first thing you need to know about our
   site.</p>
<input type="button" id="next" value="Next">
```

tutorial2.html

```
<h1>Tutorial - page 2</h1>
<p>Here's the second thing you need to know about our
   site.</p>
<input type="button" id="prev" value="Prev">
<input type="button" id="next" value="Next">
```

tutorial3.html

```
<h1>Tutorial - page 3</h1>
<p>Here's the third thing you need to know about our
   site.</p>
<input type="button" id="prev" value="Prev">
<input type="button" id="next" value="Finish">
```

The JavaScript for each HTML file (part 1)

index.html

```
<script>
    $("#enter").click( () => location = "main.html");
    $("#tutorial").click( () =>
        location = "tutorial1.html");
</script>
```

tutorial1.html

```
<script>
    $("#next").click( () =>
        location.replace("tutorial2.html"));
</script>
```

The JavaScript for each HTML file (part 2)

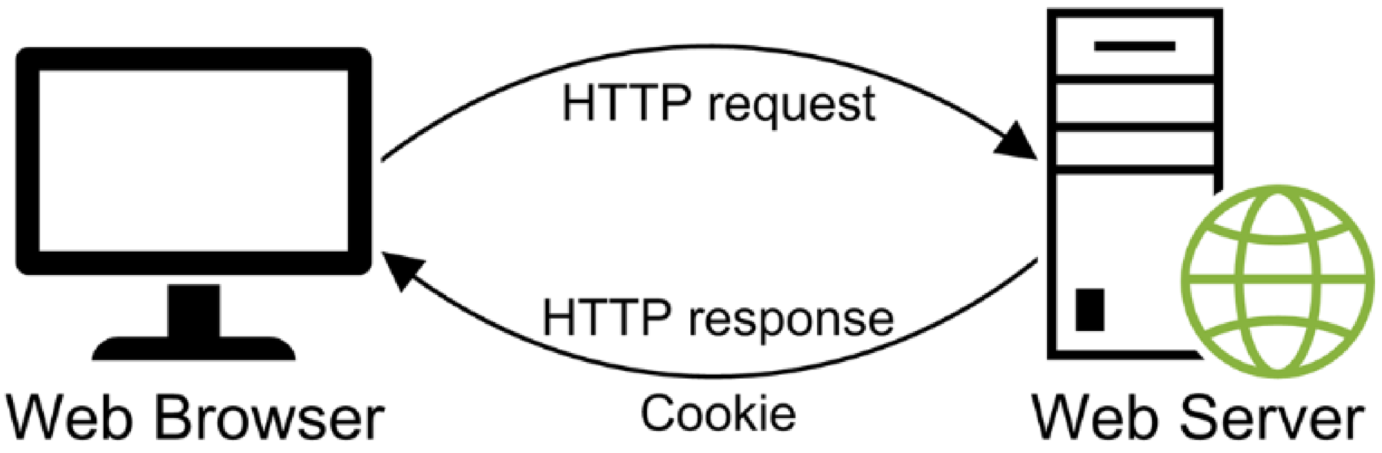
tutorial2.html

```
<script>
    $("#prev").click( () =>
        location.replace("tutorial1.html"));
    $("#next").click( () =>
        location.replace("tutorial3.html"));
</script>
```

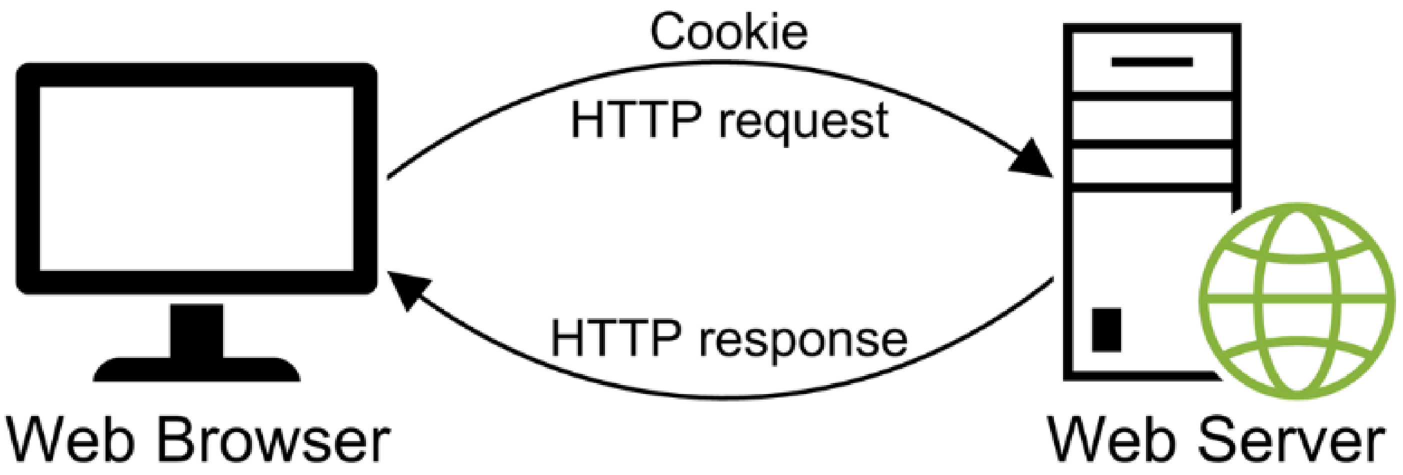
tutorial3.html

```
<script>
    $("#prev").click( () =>
        location.replace("tutorial2.html"));
    $("#next").click( () =>
        location.replace("main.html"));
</script>
```

A browser usually gets a cookie as part of an HTTP response



A browser sends the cookie back to the server with each HTTP request



Attributes of a cookie

Attribute	Description
`max-age`	The lifetime of the cookie in seconds
`path`	The path on the web server that can see the cookie
`domain`	The domain names that can see the cookie
`secure`	If present, the cookie must be encrypted when it is transmitted, and it can only be transmitted when the browser and server are connected by HTTPS or another secure protocol.

Cookie examples

```
email=grace@yahoo.com; path=/
username=ghopper; max-age=1814400; path=/
```

Terms related to cookies

cookie

session cookie

persistent cookie

Two functions for working with cookies

```
encodeURIComponent(value)
decodeURIComponent(value)
```

How to create a session cookie

```
// create the cookie name
let cookie = "tasks=";

// encode and add the data
cookie +=
    encodeURIComponent("Feed dog\nWater plants");

// add the path
cookie += "; path=/";

// store the cookie
document.cookie = cookie;
```

How to create a persistent cookie

```
// create the cookie name
let cookie = "tasks=";

// encode and add the data
cookie +=
    encodeURIComponent("Feed dog\nWater plants");
// add the max-age attribute
cookie += "; max-age=" + 21 * 24 * 60 * 60;

// add the path
cookie += "; path=/";

// store the cookie
document.cookie = cookie;
```

How to add multiple cookies to the document.cookie object

```
document.cookie = "email=john@doe.com; path=/"
document.cookie =
    "username=ghopper; max-age=1814400; path=/"
document.cookie = "email=grace@yahoo.com; path=/"
```

The cookies that are added to the document.cookie object

```
email=grace@yahoo.com; username=ghopper
```

A setCookie() function that creates a session or persistent cookie

```
const setCookie = (name, value, days) => {

    // concatenate cookie name and encoded value
    let cookie = name + "=" + encodeURIComponent(value);

    // if there's a value for days, add max-age to cookie
    if (days) {
        cookie += "; max-age=" + days * 24 * 60 * 60;
    }
    // add the path and then store the cookie
    cookie += "; path=/";
    document.cookie = cookie;
};
```

How to use the setCookie() function to create a persistent cookie

```
setCookie("tasks", "Water plants", 21);
```

Three cookies in the document.cookie object

```
username=ghopper; status=active; tasks=Water%20plants
```

A getCookieByName() function that gets a cookie by name (part 1)

```
const getCookieByName = name => {
    const cookies = document.cookie;

    // get the starting index of the cookie name
    // followed by an equals sign
    let start = cookies.indexOf(name + "=");

    if (start === -1) {      // no cookie with that name
        return "";           // return empty string
    }
    else {                   // get cookie value
        // adjust so the name and equals sign
        // aren't included in the result
        start = start + (name.length + 1);
```

A getCookieByName() function that gets a cookie by name (part 2)

```
            // get the index of the semi-colon at the end
            // of the cookie value
            let end = cookies.indexOf(";", start);
            if (end === -1) {        // last cookie
                end = cookies.length;
            }

            // use the start and end indexes to get
            // the cookie value
            const cookieValue =
                cookies.substring(start, end);

            // return the decoded cookie value
            return decodeURIComponent(cookieValue);
        }
    };
```

How to use the getCookieByName() function to read a cookie

```
const tasks = getCookieByName("tasks");
    // tasks = "Water plants"
```

The cookie to delete

```
tasks=Feed dog; max-age=1814400; path=/
```

How to delete a cookie

```
let cookie = "tasks=";                // set the name and data
cookie += "; max-age=" + 0;           // set max-age to 0
cookie += "; path=/";                 // set the path
document.cookie = cookie;             // delete the cookie
```

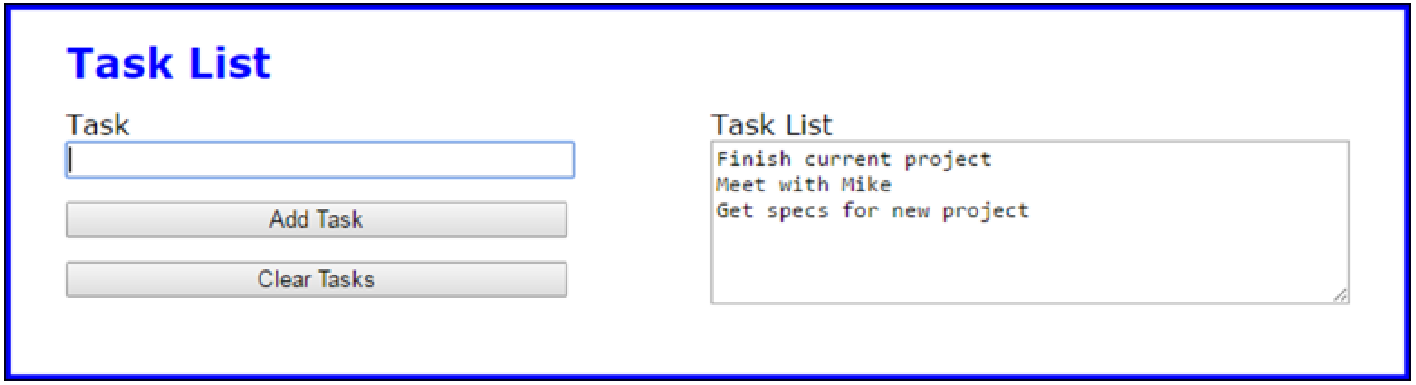
A deleteCookie() function that deletes a cookie

```
const deleteCookie = name => {
    document.cookie = name + "=''; max-age=0; path=/";
};
```

How to delete a cookie

```
deleteCookie("tasks");
```

The Task List application



```
Finish current project
Meet with Mike
Get specs for new project
```

The HTML for the Task List application (part 1)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, initial-scale=1">
  <title>Task List</title>
  <link rel="stylesheet" href="task_list.css">
</head>
<body>
  <main>
    <h1>Task List</h1>
    <div id="tasks">
      <label for="task_list">Task List</label><br>
      <textarea id="task_list" rows="6" cols="50">
      </textarea>
    </div>
    <div>
      <label for="task">Task</label><br>
      <input type="text" name="task" id="task">
    </div>
```

The HTML for the Task List application (part 2)

```
        <div>
          <input type="button" name="add_task"
                 id="add_task" value="Add Task"><br>
          <input type="button" name="clear_tasks"
                 id="clear_tasks" value="Clear Tasks">
        </div>
    </main>
    <script
        src="https://code.jquery.com/jquery-3.4.1.min.js">
    </script>
    <script src="task_list.js"></script>
</body>
</html>
```

The CSS for the div element with “tasks” as its id

```
#tasks {
    margin-top: 0;
    float: right;
}
```

The JavaScript for the Task List app (part 1)

```
const setCookie = (name, value, days) => {
    let cookie = name + "=" + encodeURIComponent(value);
    if (days) {
        cookie += "; max-age=" + days * 24 * 60 * 60;
    }
    cookie += "; path=/";
    document.cookie = cookie;
};

const getCookieByName = name => {
    const cookies = document.cookie;
    let start = cookies.indexOf(name + "=");
    if (start === -1) {
        return "";
    } else {
        start = start + (name.length + 1);
        let end = cookies.indexOf(";", start);
        if (end === -1) {
            end = cookies.length;
        }
        const cookieValue = cookies.substring(start, end);
        return decodeURIComponent(cookieValue);
    }
};
```

The JavaScript for the Task List app (part 2)

```
const deleteCookie = name =>
    document.cookie = name + "=''; max-age=0; path=/";

$(document).ready( () => {

    $("#add_task").click( () => {
        const textbox = $("#task");
        const task = textbox.val();
        if (task === "") {
            alert("Please enter a task.");
            textbox.focus();
        } else {
            let tasks = getCookieByName("tasks");
            tasks = tasks.concat(task, "\n");
            setCookie("tasks", tasks, 21);

            textbox.val("");
            $("#task_list").val(getCookieByName("tasks"));
            textbox.focus();
        }
    });
```

The JavaScript for the Task List app (part 3)

```
        $("#clear_tasks").click( () => {
            deleteCookie("tasks");
            $("#task_list").val("");
            $("#task").focus();
        });

        // display tasks on initial load
        $("#task_list").val(getCookieByName("tasks"));
        $("#task").focus();
    });
```

The syntax for working with local storage

```
// saves the data in the item
localStorage.setItem("key", "value")

// gets the data in the item
localStorage.getItem("key")

// removes the item
localStorage.removeItem("key")

// removes all items
localStorage.clear()
```

The syntax for working with session storage

```
// saves the data in the item
sessionStorage.setItem("key", "value")

// gets the data in the item
sessionStorage.getItem("key")

// removes the item
sessionStorage.removeItem("key")

// removes all items
sessionStorage.clear()
```

The shortcut syntax for getting or saving an item

```
// save or get the data in the local storage item
localStorage.key

// save or get the data in the session storage item
sessionStorage.key
```

JavaScript that uses local and session storage for hit counters

```
$(document).ready( () => {

    if (localStorage.hits) {
        localStorage.hits =
            parseInt(localStorage.hits) + 1;
    } else { localStorage.hits = 1; }

    if (sessionStorage.hits) {
        sessionStorage.hits =
            parseInt(sessionStorage.hits) + 1;
    } else { sessionStorage.hits = 1; }

    console.log("Hits for this browser: " +
        localStorage.hits + "\n" +
        "Hits for this session: " +
        sessionStorage.hits)
});
```

A console message that shows the current values of both hits items

```
Hits for this browser: 5
Hits for this session: 2
```

Terms related to web storage

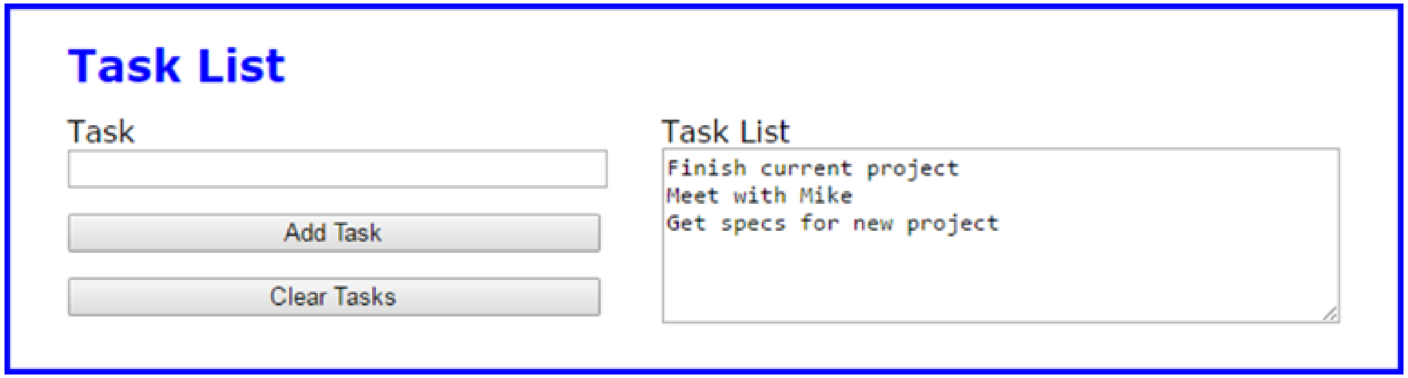
web storage

key/value pair

local storage

session storage

The Task List application with web storage



```
Finish current project
Meet with Mike
Get specs for new project
```

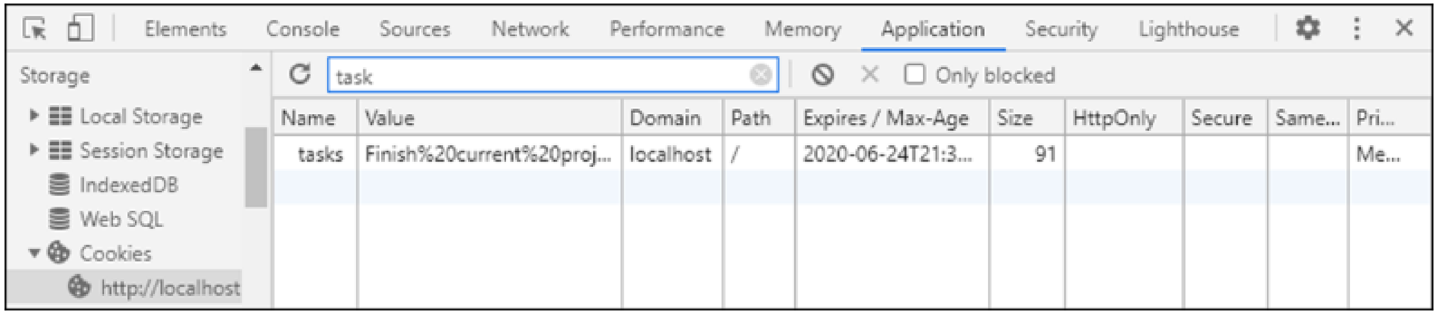
The JavaScript for the Task List application with web storage (part 1)

```
$(document).ready( () => {

    $("#add_task").click( () => {
        const textbox = $("#task");
        const task = textbox.val();
        if (task === "") {
            alert("Please enter a task.");
            textbox.focus();
        } else {
            let tasks = localStorage.myTasks || "";
            localStorage.myTasks =
                tasks.concat(task, "\n");

            textbox.val("");
            $("#task_list").val(localStorage.myTasks);
            textbox.focus();
        }
    });
```

The JavaScript for the Task List application with web storage (part 2)

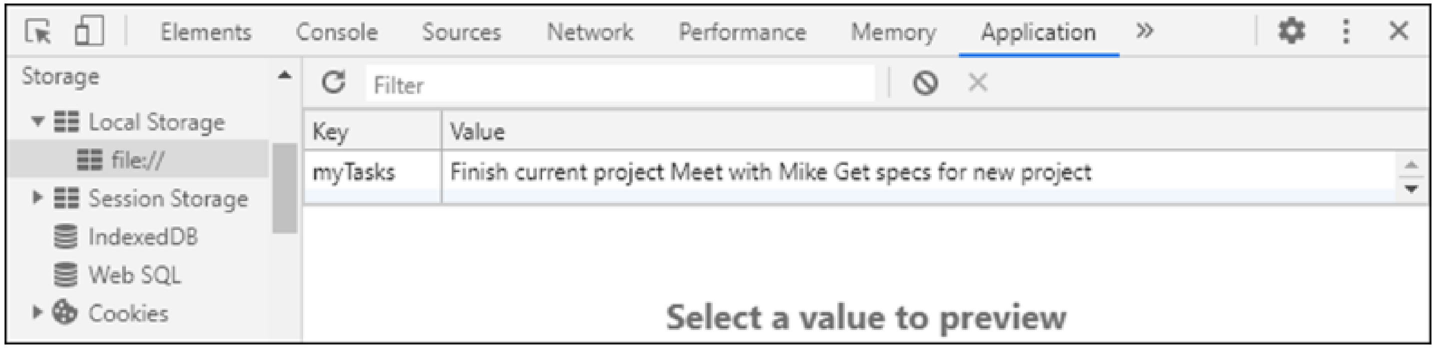
```
    $("#clear_tasks").click( () => {
        localStorage.removeItem("myTasks");
        $("#task_list").val("");
        $("#task").focus();
    });

    // display tasks on initial load
    $("#task_list").val(localStorage.myTasks);
    $("#task").focus();
});
```

The cookies for an application in the Application panel



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	Same...	Pri...
tasks	Finish%20current%20proj...	localhost	/	2020-06-24T21:3...	91				Me...

The local storage items for an application in the Application panel



Key	Value
myTasks	Finish current project Meet with Mike Get specs for new project

Select a value to preview

How to view cookies and stored items

1. Press F12 to open the developer tools.
2. Click on the Application tab at the top of the tools window, and then look for the Storage section in the pane on the left-hand side.
3. Expand an appropriate storage type such as Cookies, Local Storage, or Session Storage.
4. Click on the appropriate URL to display the stored values in a grid. For cookie storage, you may want to filter the cookies by entering the name of the cookie you want to view.

How to delete or edit cookies or stored items

To delete an item, right-click the item and select Delete.

To edit an item, right-click the item and select Edit Value. Then, edit the text for the value.

To apply the edit or deletion to the application, make sure to refresh the page.

Chapter 15

How to work with arrays, sets, and maps

Objectives (part 1)

Applied

1. Use arrays in your applications.
2. Use associative arrays and arrays of arrays in your applications.
3. Use sets and maps in your applications.

Knowledge

1. Describe the creation and use of an array.
2. Describe the use of indexes and the length property for working with an array.
3. Distinguish between the use of for, for-of, and for-in loops for working with an array.
4. Describe how destructuring can assign the elements in an array to individual constants or variables.

Objectives (part 2)

5. Describe these methods of an Array object: push(), pop(), unshift(), shift(), splice(), slice(), indexOf(), lastIndexOf(), includes(), entries(), values(), and keys().
6. Describe these methods of an array that accept callback function as parameters: find(), findIndex(), filter(), every(), some(), forEach(), sort(), reduce(), map(), and flatMap().
7. Describe the static Array.isArray() method.
8. Describe how to split a string into an array.
9. Describe at least two ways to copy and array.
10. Distinguish between an array, an associative array, and an array of arrays.
11. Describe JSON.
12. Describe how you can convert an array to JSON and back.
13. Distinguish between an array, a set, and a map.

The syntax for creating an array

Using the *new* keyword with the Array() constructor

```
const arrayName = new Array(length);
```

Using the static Array.of() method

```
const arrayName = Array.of();
```

Using an array literal

```
const arrayName = [];
```

The syntax for creating an array and assigning values in one statement

Using the *new* keyword with the Array() constructor

```
const arrayName = new Array(arrayList);
```

Using the static Array.of() method

```
const arrayName = Array.of(arrayList);
```

Using an array literal

```
const arrayName = [arrayList];
```

Examples that create an array and assign values in one statement

Using the Array() constructor

```
const rates = new Array(14.95, 12.95, 11.95, 9.95);
```

Using the Array.of() method

```
const dice = Array.of(1, 2, 3, 4, 5, 6);
```

Using an array literal

```
const names = ["Grace", "Charles", "Ada"];
```

The syntax for referring to an element of an array

```
arrayName[index]
```

Code that refers to the elements in an array

```
rates[2]    // Refers to third element in rates array
names[1]    // Refers to second element in names array
```

How to assign values to an array by accessing each element

How to assign numbers to an array that starts with four undefined elements

```
const rates = new Array(4);
rates[0] = 14.95;
rates[1] = 12.95;
rates[2] = 11.95;
rates[3] = 9.95;
```

How to assign strings to an array that starts with no elements

```
const names = [];
names[0] = "Grace";
names[1] = "Charles";
names[2] = "Ada";
```

A property of an array

```
length
```

An operator of an array

```
delete
```

How to add an element to the end of an array

```
const numbers = [1, 2, 3];    // array is 1, 2, 3
numbers[numbers.length] = 4;  // array is 1, 2, 3, 4
```

How to add an element at a specific index

```
const numbers = [1, 2, 3];    // array is 1, 2, 3
numbers[5] = 6;
     // array is 1, 2, 3, undefined, undefined, 6
```

How to delete a number at a specific index

```
const numbers = [1, 2, 3];    // array is 1, 2, 3
delete numbers[1];            // array is 1, undefined, 3
```

How to remove all elements

```
const numbers = [1, 2, 3];  // array contains 3 elements
numbers.length = 0;         // removes all elements
```

A sparse array that contains 999 undefined elements

```
const numbers = [1];      // array contains 1

numbers[1000] = 1001;     // array contains 1 and 1001
                          // with 999 undefined elements
                          // in between
```

An array that's used by the following loops

```
const names = ["Grace", "Charles", "Ada"];
names[5] = "Alan";
// adds two undefined elements between "Ada" and "Alan"
```

How to use a for loop with an array

The syntax for processing all the elements

```
for (let index = 0; index < arrayName.length; index++) {
    // statements that use the index to access the array
    // element value
}
```

Code that uses a for loop

```
let displayString = "";
for (let i = 0; i < names.length; i++) {
    displayString += names[i] + " ";
}
// displayString is Grace Charles Ada undefined undefined Alan
```

How to use a for-in loop with an array

The syntax

```
for (const|let index in arrayName) {
    // statements that use the index to access the array
    // element value
}
```

Code that uses a for-in loop

```
let displayString = "";
for (const i in names) {
    displayString += names[i] + " ";
}
// displayString is Grace Charles Ada Alan
```

How to use a for-of loop with an array

The syntax

```
for (const|let value of arrayName) {
    // statements that use the array element value
}
```

Code that uses a for-of loop

```
let displayString = "";
for (const val of names) {
    displayString += val + " ";
}
// displayString is Grace Charles Ada undefined undefined Alan
```

The syntax for destructuring an array

```
const|let [identifier1, identifier2, ...] = arrayName;
```

Two arrays used by the following examples

```
const totals = [141.95, 212.25, 411, 135.75];
const fullName = ["Grace", "M", "Hopper"];
```

Example 1: Assign the first three elements in an array

```
const [total1, total2, total3] = totals;
// total1 is 141.95, total2 is 212.25, total3 is 411
```

Example 2: Skip an array element

```
const [firstName, , lastName] = fullName;
// firstName is "Grace", lastName is "Hopper"
```

Example 3: Use a default value

```
const [first, middle, last, suffix = "none"] = fullName;
// first is "Grace", middle is "M", last is "Hopper",
// suffix is "none"
```

Example 4: Use the rest operator to assign some elements to a new array

```
const[total1, total2, ...remainingTotals] = totals;
// total1 is 141.95, total2 is 212.25,
// remainingTotals is [411, 135.75]
```

Example 5: Destructure a string

```
const [first, second, third] = "USA";
// first is "U", second is "S", third is "A"
```

Terms related to arrays

array

element

length

array literal

index

sparse array

destructure an array

rest operator

Methods of the Array type that add, modify, remove, and copy elements

```
push(element_list)
pop()
unshift(element_list)
shift()
splice(start,number)
splice(start,number, element_list)
slice([start][,end])
```

A names array that’s by the following examples

```
const names = ["Grace", "Charles", "Ada"];
```

Example 1: Add elements to and remove an element from the end of the array

```
names.push("Alan", "Linus");
// names is ["Grace", "Charles", "Ada", "Alan", "Linus"]

let removedName = names.pop();   // removedName is Linus
// names is ["Grace", "Charles", "Ada", "Alan"]
```

Example 2: Add and remove an element from the beginning of the array

```
names.unshift("Linus");
// names is ["Linus", "Grace", "Charles", "Ada", "Alan"]

removedName = names.shift();     // removedName is Linus
// names is ["Grace", "Charles", "Ada", "Alan"]
```

Example 3: Replace elements from a specific index

```
names.splice(1, 2, "Mary", "Linus");
// names is ["Grace", "Mary", "Linus", "Alan"]
```

Example 4: Copy some of the elements of the array to a new array

```
const partialCopy = names.slice(1, 3);
// partialCopy is ["Mary", "Linus"]
// names array is unchanged
```

Example 5: Copy all the elements of the array to a new array

```
const fullCopy = names.slice();
// fullCopy is ["Grace", "Mary", "Linus", "Alan"]
// names is unchanged
```

Methods of the Array type that inspect an array or its elements

```
indexOf(value[,start])
lastIndexOf(value[,start])
includes(value[,start])
find(function)
findIndex(function)
filter(function)
every(function)
some(function)
entries()
values()
keys()
isArray(object)
```

A numbers array used by the following examples

```
const numbers =
    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
```

Example 1: Two ways to check if a specific value is in the array

```
if(numbers.indexOf(5) != -1) { ... }
if(numbers.includes(5)) { ... }
```

Example 2: Filter the numbers in the array to get all the values less than 10

```
const lessThanTen =
    numbers.filter( value => value < 10 );
// lessThanTen is [1, 2, 3, 4, 5, 6, 7, 8, 9]
// numbers array is unchanged
```

Example 3: Check if all the numbers in the array are positive

```
const isAllPos = numbers.every( value => value > 0 );
// isAllPos is true
```

Example 4: Use a for-of loop with the entries() method

```
for(let [key, val] of numbers.entries()) {
    console.log(`key: ${key}, val: ${val}`);
}
// displays "key: 0, val: 1", "key: 1, val: 2", etc.
```

Methods of the Array type that transform elements

```
forEach(function)
join([separator])
toString()
sort([comparison_function])
reverse()
map(function)
reduce(function[,init])
flat([depth])
flatMap(function)
```

Example 1: Convert the elements in an array to a single string

```
const names = ["Grace", "Charles", "Ada", "Alan"];

let str = names.join();
// str is "Grace,Charles,Ada,Alan"

str = names.join(", ");
// str is "Grace, Charles, Ada, Alan"

str = names.toString();

// str is "Grace,Charles,Ada,Alan"
```

Example 2: Sort numeric values in ascending sequence

```
const numbers = [5, 12, 8, 6, 9, 2];
numbers.sort( (x, y) => x - y );
// numbers is [2, 5, 6, 8, 9, 12]
```

Example 3: Create a new array with each element multiplied by 2

```
const doubled = numbers.map( value => value * 2 );
// doubled is [4, 10, 12, 16, 18, 24]
// numbers is unchanged
```

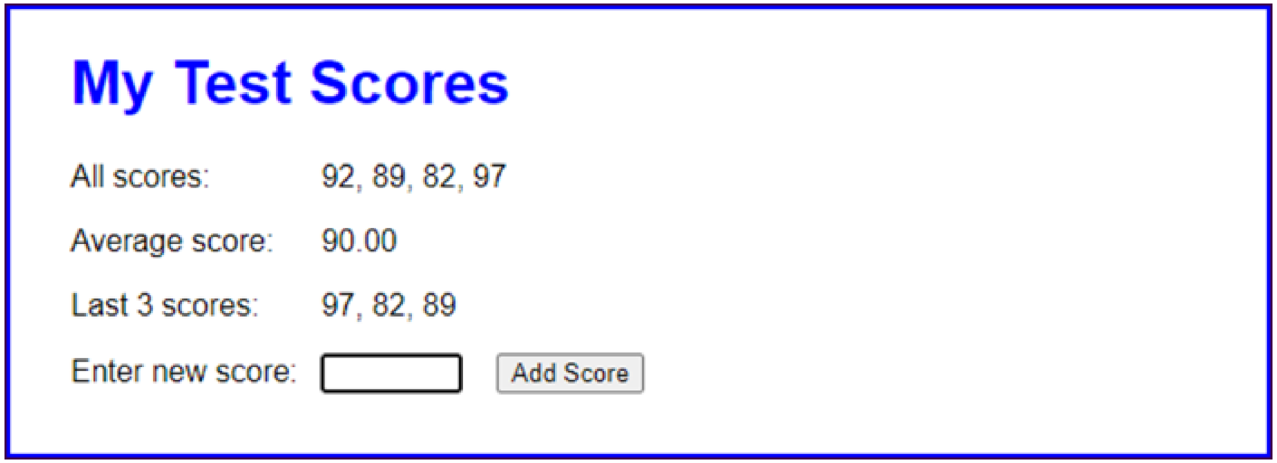
Example 4: Transform the elements and then convert to a single string

```
const names = ["Grace", "Charles", "Ada", "Alan"];
let str = names.reduce( (prev, current) =>
    prev + " " + current.toLowerCase(), "Names:" );
// str is "Names: grace charles ada alan"
```

Example 5: Flatten an array that's nested two levels deep

```
const nested = [5, 12, 8, 6, 9, [4, [0, 7], 1], 2];
const flattened = nested.flat(2);
// flattened is [5, 12, 8, 6, 9, 4, 0, 7, 1, 2]
```

The Test Scores application



My Test Scores

All scores: 92, 89, 82, 97

Average score: 90.00

Last 3 scores: 97, 82, 89

Enter new score:

The HTML for the Test Scores application (part 1)

```
<body>
    <main>
        <h1>My Test Scores</h1>
        <div>
            <label>All scores:</label>
            <label id="all"></label>
        </div>
        <div>
            <label>Average score:</label>
            <label id="avg"></label>
        </div>
        <div>
            <label>Last 3 scores:</label>
            <label id="last"></label>
        </div>
```

The HTML for the Test Scores application (part 2)

```
        <div>
            <label for="score">Enter new score:</label>
            <input type="text" id="score">
            <input type="button" id="add_score"
                   value="Add Score">
            <span></span>
        </div>
    </main>

    <script
        src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
    </script>
    <script src="test_scores.js"></script>
</body>
```

The JavaScript for the Test Scores app (part 1)

```
"use strict";

$(document).ready( () => {

    const scores = [];

    $("#add_score").click( () => {

        const score = parseFloat($("#score").val());

        if (isNaN(score) || score < 0 || score > 100) {
            $("#add_score").next().text(
                "Score must be from 0 to 100.");
        }
        else {
            $("#add_score").next().text("");

            // add score to scores array
            scores.push(score);

            // display all scores
            $("#all").text(scores.join(", "));
```

The JavaScript for the Test Scores app (part 2)

```
            // calculate and display average score
            const total =
                scores.reduce( (tot, val) => tot + val, 0 );
            const avg = total/scores.length;
            $("#avg").text(avg.toFixed(2));

            // display last 3 scores
            const len = scores.length;
            const lastScores = (len <= 3) ? scores.slice() :
                scores.slice(len - 3, len); // copy last three
            lastScores.reverse();
            $("#last").text(lastScores.join(", "));
        }

        // get text box ready for next entry
        $("#score").val("");
        $("#score").focus();
    });

    // set focus on initial load
    $("#score").focus();
});
```

A String method that creates an array

```
split(separator[,limit])
```

How to split a string that's separated by spaces into an array

```
const fullName = "Grace M Hopper";
const nameParts = fullName.split(" "); // creates array

console.log(nameParts.length);         // displays 3
console.log(nameParts.toString());
                             // displays Grace,M,Hopper

const lastName = nameParts[nameParts.length - 1];
console.log(lastName);       // displays Hopper
```

How to split a string that’s separated by hyphens into an array

```
const date = "1-2-2021";
const dateParts = date.split("-");   // creates an array

console.log(dateParts.length);       // displays 3
console.log(dateParts.join("/"));    // displays 1/2/2021
```

How to split a string into an array of characters

```
const fullName = "Grace Hopper";
const nameCharacters = fullName.split("");

console.log(nameCharacters.length);  // displays 12
console.log(nameCharacters.join());
                // displays G,r,a,c,e, ,H,o,p,p,e,r
```

How it works if the string doesn't contain the separator

```
const date = "1-2-2021";
const dateParts = date.split("/");

console.log(dateParts.length);       // displays 1
console.log(dateParts.join());       // displays 1-2-2021
```

How to get just one element from a string

```
const fullName = "Grace M Hopper";
const firstName = fullName.split(" ", 1);

console.log(firstName.length);          // displays 1
console.log(firstName[0]);              // displays Grace
```

A static method of the Array type

```
from(array)
```

Four ways to make a copy of an existing array

Call the slice() method of the array.

Pass the array to the static from() method of the Array type.

Destructure the array with a rest operator.

Use an array literal with a spread operator and the array.

An array that’s copied by the following examples

```
const names = ["Grace", "Charles", "Ada"];
```

How to copy the array

Using the slice() method

```
const namesCopy = names.slice();
```

Using the Array.from() method

```
const namesCopy = Array.from(names);
```

Using destructuring and the rest operator

```
const [...namesCopy] = names;
```

Using an array literal and the spread operator

```
const namesCopy = [...names];
```

How to create an associative array with four elements

```
const item = [];
item["code"] = 123;
item["name"] = "HTML5";
item["cost"] = 54.5;
item["quantity"] = 5;

console.log(item.length);                      // Displays 0
console.log(Object.keys(item).length);         // Displays 4
```

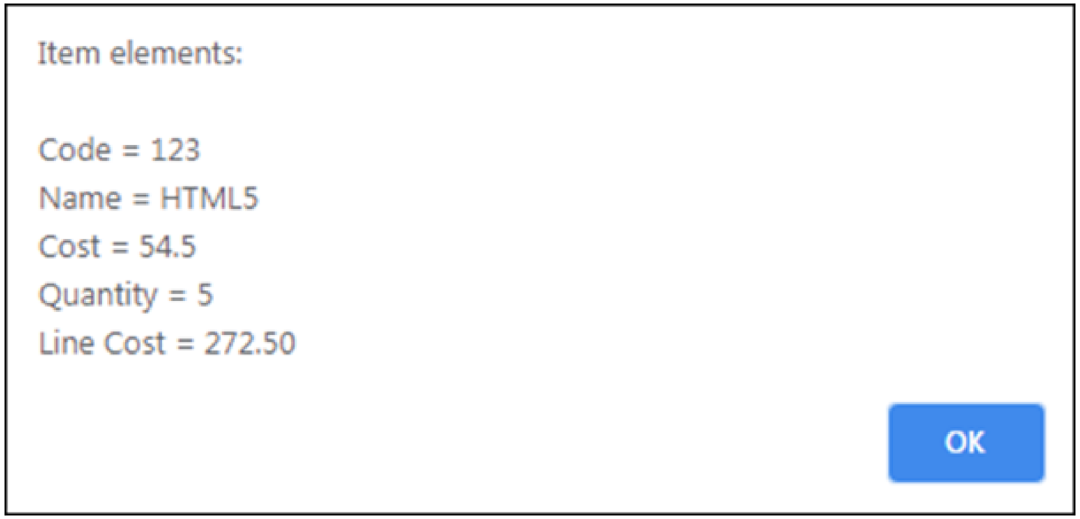
How to add an element to the associative array

```
item["lineCost"] =
    (item["cost"] * item["quantity"]).toFixed(2);
```

How to retrieve and display the elements in the associative array

```
alert("Item elements:\n" +
      "\nCode = " + item["code"] +
      "\nName = " + item["name"] +
      "\nCost = " + item["cost"] +
      "\nQuantity = " + item["quantity"] +
      "\nLine Cost = " + item["lineCost"]);
```

The message displayed by the alert statement



```
Item elements:

Code = 123
Name = HTML5
Cost = 54.5
Quantity = 5
Line Cost = 272.50
```

How to use a for-in loop with the associative array

```
let result = "";
for (let i in item) {
    result += i + "=" + item[i] + " ";
}
// result is "code=123 name=HTML5 cost=54.5 quantity=5
// lineCost=272.50 "
```

How to create and use an array of arrays

Code that creates an array of arrays

```
const students = [];
students[0] = [80, 82, 90, 87, 85];
students[1] = [79, 80, 74];
students[2] = [93, 95, 89, 100];
students[3] = [60, 72, 65, 71];
```

Code that refers to elements in the array of arrays

```
console.log(students[0][1]);       // displays 82
console.log(students[2][3]);       // displays 100
```

How to create and use an array of associative arrays (part 1)

Code that creates an array

```
const invoice = [];    // create an empty invoice array
```

Code that adds an associative array to the invoice array directly

```
invoice[0] = [];
invoice[0]["code"] = 123;
invoice[0]["name"] = "HTML5";
invoice[0]["cost"] = 54.5;
invoice[0]["quantity"] = 5;
```

How to create and use an array of associative arrays (part 2)

Code that creates an associative array and then adds it to the invoice array

```
const item = [];
item["code"] = 456;
item["name"] = "jQuery";
item["cost"] = 52.5;
item["quantity"] = 2;
invoice.push(item);
```

Code that refers to the elements in the array of associative arrays

```
console.log(invoice[0]["code"]);     // displays 123
console.log(invoice[1]["name"]);     // displays jQuery
```

More terms related to arrays

spread operator

associative array

arrays

Two static methods of the JSON type

Method	Description
`stringify(`*`object`*`)`	Returns a JSON string for the specified object or array.
`parse(`*`json`*`)`	Returns an object or array that contains the data in the specified JSON string.

How to convert an Array object to JSON

```
// create an array of arrays that stores 2 tasks
const tasks = [];
tasks.push(["Finish current project",
            new Date("11/20/2020")]);
tasks.push(["Get specs for next project",
            new Date("12/01/2020")]);

// convert array to JSON
const json = JSON.stringify(tasks);

// save JSON to web storage
localStorage.tasks = json;
```

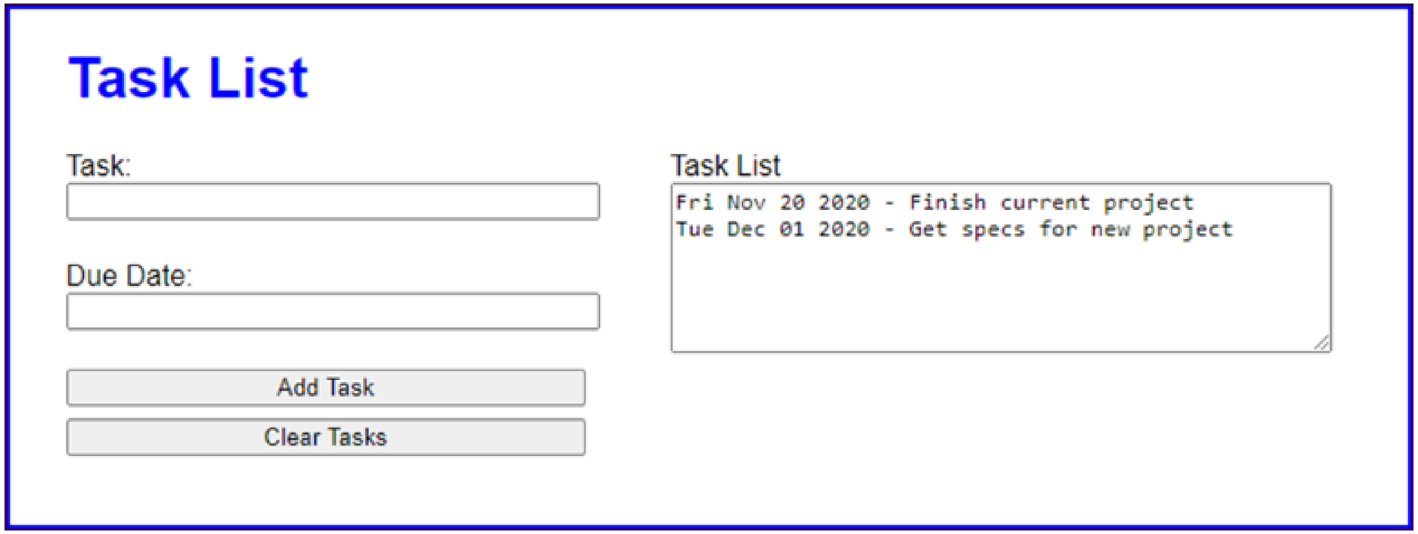
The JSON for the Array object

```
[["Finish current project","2020-11-20T08:00:00.000Z"],
["Get specs for next project","2020-12-
01T08:00:00.000Z"]]
```

How to convert JSON to an Array object

```
// get JSON from web storage
const json = localStorage.tasks;

// convert JSON to array
const tasks = JSON.parse(json);

console.log(tasks[0][0]);
              // displays "Finish current project"
console.log(tasks[1][1]);
              // displays "2020-12-01T08:00:00.000Z"

// convert JSON string to Date object
const date = new Date(tasks[1][1]);
console.log(date.toDateString());
              // displays "Tue Dec 1 2020"
```

The Task List application



```
Fri Nov 20 2020 - Finish current project
Tue Dec 01 2020 - Get specs for new project
```

The HTML for the Task List application (part 1)

```
<body>
    <main>
        <h1>Task List</h1>
        <div id="tasks">
            <label for="task_list">Task List</label><br>
            <textarea id="task_list" rows="6" cols="50">
            </textarea>
        </div>
        <div>
            <label for="task">Task:</label><br>
            <input type="text" name="task" id="task">
        </div>
        <div>
            <label for="due_date">Due Date:</label><br>
            <input type="text" name="due_date"
                   id="due_date">
        </div>
```

The HTML for the Task List application (part 2)

```
            <div>
                <input type="button" id="add_task"
                       value="Add Task"><br>
                <input type="button" id="clear_tasks"
                       value="Clear Tasks">
            </div>
        </main>

        <script
            src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
        </script>
        <script src="task_list.js"></script>
    </body>
```

The JavaScript for the Task List app (part 1)

```
const displayTaskList = tasks => {
    let taskString = "";
    if (tasks.length > 0) {
        // convert stored date string to Date object
        tasks = tasks.map( task =>
            [task[0], new Date(task[1])] );

        tasks.sort( (task1, task2) => {   // sort by date
            const date1 = task1[1]; // get Date object from task1
            const date2 = task2[1]; // get Date object from task2
            if (date1 < date2) { return -1; }
            else if (date1 > date2) { return 1; }
            else { return 0; }
        });

        taskString = tasks.reduce( (prev, curr) => {
            return prev + curr[1].toDateString() + " - " +
                curr[0] + "\n";
        }, ""); // pass initial value for prev parameter
    }

    $("#task_list").val(taskString);
    $("#task").focus();
};
```

The JavaScript for the Task List app (part 2)

```
$(document).ready( () => {
    const taskString = localStorage.tasks;
    const tasks = (taskString) ? JSON.parse(taskString) : [];

    $("#add_task").click( () => {
        const task = $("#task").val();
        const dateString = $("#due_date").val();
        const dueDate = new Date(dateString);

        if (task && dateString && dueDate != "Invalid Date") {
            // store dateString
            const newTask = [task, dateString];

            tasks.push(newTask);
            localStorage.tasks = JSON.stringify(tasks);

            $("#task").val("");
            $("#due_date").val("");
            displayTaskList(tasks);
        } else {
            alert("Please enter a task and valid due date.");
            $("#task").select();
        }
    });
```

The JavaScript for the Task List app (part 3)

```
        $("#clear_tasks").click( () => {
            tasks.length = 0;
            localStorage.removeItem("tasks");
            $("#task_list").val("");
            $("#task").focus();
        });

        displayTaskList(tasks);
    });
```

The syntax for creating a set

```
const setName = new Set([array]);
```

A statement that creates an empty set

```
const emptySet = new Set();
```

A statement that creates a set from an array

```
const names = new Set(["Grace", "Charles", "Ada"]);
```

A property of the Set type

```
size
```

Some of the methods of the Set type

```
add(value)
delete(value)
clear()
has(value)
forEach(function)
entries()
values()
keys()
```

How to check for values in a set

```
let hasName = names.has("Grace");  // hasName is true
hasName = names.has("Linus");      // hasName is false
```

How to add values to a set

```
let size = names.size;   // size is 3
names.add("Linus");      // adds new value to set
size = names.size;       // size is 4

names.add("Grace");      // already in set so not added
size = names.size;       // size is 4
```

How to use a set to remove duplicate values from an array

```
const dupeArray = [4, 6, 2, 3, 2, 3, 4, 7, 6, 8, 9, 2, 8, 2];
const set = new Set(dupeArray);
const noDupes = Array.from(set);
    // noDupes is [4, 6, 2, 3, 7, 8, 9]
```

The syntax for creating a map

```
const mapName = new Map([arrayOfArrays]);
```

A statement that creates an empty map

```
const map = new Map();
```

A statement that creates a map from an array of arrays

```
const names = new Map(
    [ ["Grace", "Hopper"], ["Ada", "Lovelace"] ]);
```

A property of the Map type

```
size
```

Some of the methods of the Map type

```
set(key,value)
get(key)
delete(key)
clear()
has(key)
forEach(function)
entries()
values()
keys()
```

How to check for a key in the map and retrieve its associated value

```
if (names.has("Grace")) {
    console.log(names.get("Grace"));
    // displays "Hopper"
}
```

How to add key/value pairs to the map

```
let size = names.size;                 // size is 2

// add new key/value pair to map
names.set("Charles", "Babbage");
size = names.size;                     // size is 3

// replace value associated with key
names.set("Grace", "Slick");
size = names.size;                     // size is 3
```

How to get arrays of the keys and key/value pairs of the map

```
const full = Array.from(names);
// full is [["Grace", "Slick"], ["Ada", "Lovelace"],
// ["Charles", "Babbage"]]

const firstNames = Array.from(names.keys());
// firstNames is ["Grace", "Ada", "Charles"]
```

Chapter 16

How to work with objects

Objectives (part 1)

Applied

1. Use object literals to create objects that have properties and methods.
2. Use classes to create objects that have properties and methods.
3. Use the properties and methods of the objects you create.
4. Use JavaScript libraries in your applications.

Knowledge

1. Distinguish between creating objects with object literals and creating objects with classes.
2. Distinguish between the concise method syntax for coding methods and the traditional syntax.
3. Explain how two variables can refer to the same object.
4. Distinguish between data properties and accessor properties.

Objectives (part 2)

5. Describe how getters and setters work with read-only and write-only properties.
6. Describe the creation, use, and benefits of JavaScript libraries.
7. Describe how you can use a class to define objects that have properties and methods.
8. Describe the inheritance hierarchy for these JavaScript object types: Object, String, Number, Boolean, Date, Array, and Function.
9. Distinguish between inheritance and object composition and describe when you would use each technique.
10. Distinguish between a class-based language and a prototypal language.
11. Describe how to use a symbol and a generator function to create an iterator for an object.

Objectives (part 3)

12. Describe the creation and use of cascading methods.
13. Describe how you can use destructuring to assign the values stored in an object to individual variables or constants.

How to initialize a new object with an object literal

```
const invoice = {};
```

How to initialize a new object with properties and methods

```
const invoice = {
    taxRate: 0.0875,                // property
    getTotal(subtotal) {            // method
        const salesTax = subtotal * this.taxRate;
                                    // this = the object
        return subtotal + salesTax;
    }
};
```

How to use dot notation to refer to an object’s properties and methods

```
console.log(invoice.taxRate);        // displays 0.0875
const total = invoice.getTotal(100); // total is 108.75
```

How to nest objects

```
const invoice = {
    terms: {
        dueDays: 30,
        description: "Net due 30 days"
    }
};
```

How to use dot notation to refer to nested objects

```
console.log(invoice.terms.dueDays);
// displays 30

console.log(invoice.terms.description);
// displays 'Net due 30 days'
```

Terms related to object literals

object literal

property

method

nested object

Two ways to code a method

Using traditional syntax

```
const invoice = {
    getTotal: function(subtotal, taxRate) {
        return subtotal + (subtotal * taxRate);
    }
};
```

Using concise method syntax

```
const invoice = {
    getTotal(subtotal, taxRate) {
        return subtotal + (subtotal * taxRate);
    }
};
```

Two ways to initialize a property to a variable or constant

Two variables

```
let name = "Grace";
let year = 2021;
```

Using traditional syntax

```
const person = {
    name: name,
    year: year
};
```

Using shorthand properties

```
const person = {
    name,
    year
};
```

How to add properties and methods to an object

```
// create an object
const invoice = {};

// add a property
invoice.taxRate = 0.0875;

// add a method
invoice.getSalesTax(subtotal) {
    return (subtotal * this.taxRate);
};
```

How to modify the value of a property

```
invoice.taxRate = 0.095;
```

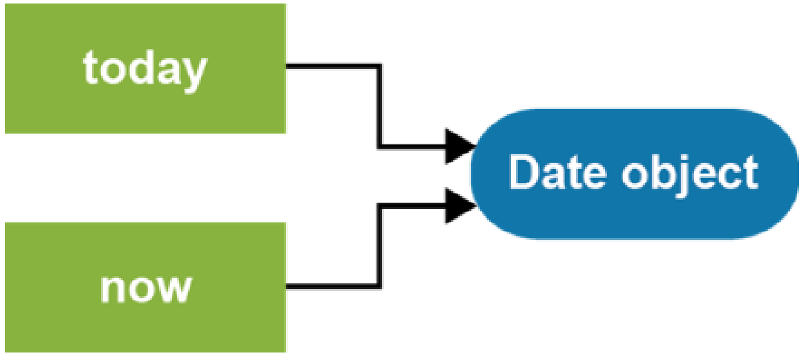
How to remove a property from an object

```
delete invoice.taxRate;
console.log(invoice.taxRate);    // displays undefined
```

Two constants that refer to the same object

```
const today = new Date();
const now = today;

today.setFullYear(1968);
console.log(now.getFullYear());     // displays 1968
```

A diagram that illustrates these references



An object with an accessor property named fullName

```
const person = {
    firstName: "Grace",
    lastName: "Hopper",
    get fullName() {
        return `${this.firstName} ${this.lastName}`;
    },
    set fullName(val) {
        const names = val.split(" ");
        if(names && names.length === 2) {
            this.firstName = names[0];
            this.lastName = names[1];
        }
        else {
            throw new TypeError(
                "fullName must include first and " +
                "last name.");
        }
    }
};
```

Code that uses the accessor property

```
console.log(person.fullName);  // displays "Grace Hopper"

person.fullName = "Ada Lovelace";

console.log(person.firstName); // displays "Ada"
console.log(person.lastName);  // displays "Lovelace"
console.log(person.fullName);  // displays "Ada Lovelace"
```

Terms related to methods and properties

concise method syntax

shorthand property notation

delete operator

reference to an object

data property

accessor property

getter

setter

read-only property

write-only property

The benefits of JavaScript libraries

They let you group similar functionality in a single file.

They make your code easier to understand, maintain, and reuse.

They encourage the separation of concerns.

The lib_mpg.js file

```
const mpg = {
    miles: 0,
    gallons: 0,
    calculate() {
        return this.miles / this.gallons;
    }
};
```

How to include and use JavaScript libraries in your application

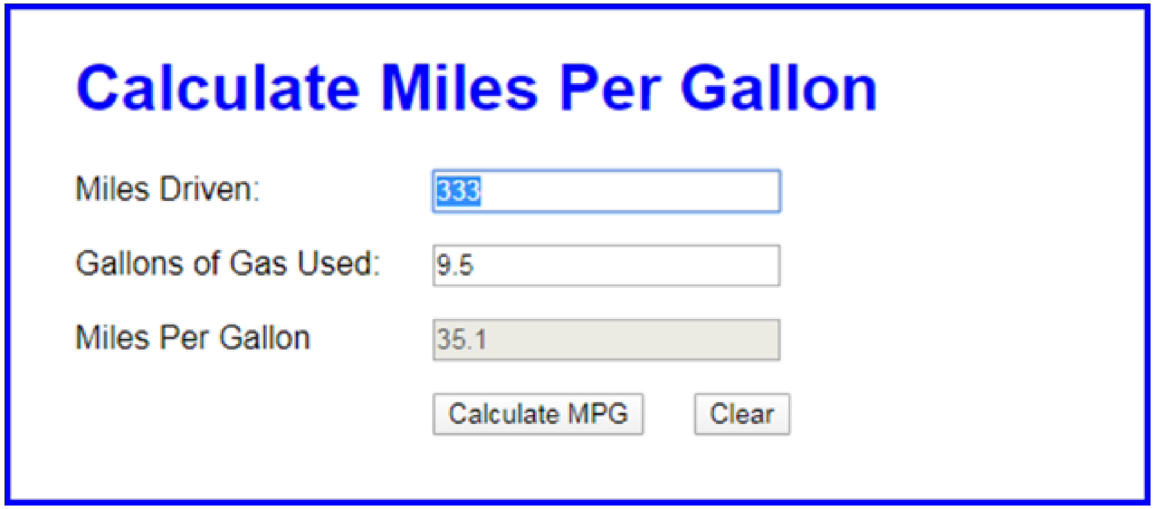
In the index.html file

```
<body>
    ...
    <script
      src="https://code.jquery.com/jquery-3.4.1.min.js">
    </script>
    <script src="lib_mpg.js"></script>
    <script src="main.js"></script>
</body>
```

In the main.js file

```
$(document).ready( () => {
    $("#calculate").click( () => {
        mpg.miles = parseFloat($("#miles").val());
        mpg.gallons = parseFloat($("#gallons").val());
        $("#mpg").val(mpg.calculate().toFixed(1));
    });
});
```

The Miles Per Gallon application



Calculate Miles Per Gallon

Miles Driven: 333

Gallons of Gas Used: 9.5

Miles Per Gallon 35.1

The HTML for the MPG application (part 1)

```
<body>
    <main>
        <h1>Calculate Miles Per Gallon</h1>
        <div>
            <label for="miles">Miles Driven:</label>
            <input type="text" id="miles">
        </div>
        <div>
            <label for="gallons">Gallons of Gas Used:
            </label>
            <input type="text" id="gallons">
        </div>
        <div>
            <label for="mpg">Miles Per Gallon</label>
            <input type="text" id="mpg" disabled>
        </div>
```

The HTML for the MPG application (part 2)

```
        <div>
            <label></label>
            <input type="button" id="calculate"
                   value="Calculate MPG">
            <input type="button" id="clear"
                   value="Clear">
        </div>
    </main>

    <script
        src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
    </script>
    <script src="lib_mpg.js"></script>
    <script src="mpg.js"></script>
</body>
</html>
```

The lib_mpg.js file for the MPG application

```
"use strict";

const mpg = {
    miles: 0,
    gallons: 0,
    get isValid() {
        if (isNaN(this.miles) || isNaN(this.gallons)) {
            return false;
        }
        else if (this.miles <= 0 || this.gallons <= 0) {
            return false;
        }
        else {
            return true;
        }
    },
    calculate() {
        return this.miles / this.gallons;
    }
};
```

The mpg.js file for the MPG application (part 1)

```
"use strict";

$(document).ready( () => {
    $("#calculate").click( () => {
        mpg.miles = parseFloat($("#miles").val());
        mpg.gallons = parseFloat($("#gallons").val());

        if (mpg.isValid) {
            $("#mpg").val(mpg.calculate().toFixed(1));
            $("#miles").select();
        } else {
            alert("Both entries must be numeric " +
                  "and greater than zero.");
            $("#miles").focus();
        }
    });
```

The mpg.js file for the MPG application (part 2)

```
        $("#clear").click( () => {
            $("#miles").val("");
            $("#gallons").val("");
            $("#mpg").val("");

            $("#miles").focus();
        });

        $("#miles").focus();
    });
```

How to use a class to define an object type

The Invoice class

```
class Invoice {
    constructor() {
        this.subtotal = null;
        this.taxRate = null;
    }
    getTotal() {
        const salesTax = this.subtotal * this.taxRate
        return this.subtotal + salesTax;
    }
}
```

How to create and use an Invoice object

```
const invoice = new Invoice();
invoice.subtotal = 100;
invoice.taxRate = 0.0875;
total = invoice.getTotal();       // total is 108.75
```

Code that attempts to create an Invoice object without the *new* keyword

```
const invoice = Invoice();
                     // throws a TypeError exception
```

How to add parameters to the constructor for the Invoice type

```
class Invoice {
    constructor(subtotal, taxRate) {
        this.subtotal = subtotal;
        this.taxRate = taxRate;
    }
    getTotal() { /* same as before */ }
}
```

How to pass arguments to the constructor

```
const invoice = new Invoice(100, 0.0875);
total = invoice.getTotal();            // total is 108.75
```

How to create two Invoice objects that hold different data

```
const invoice1 = new Invoice(100, 0.0875);
const invoice2 = new Invoice(1000, 0.07);

const total1 = invoice1.getTotal();   // total1 is 108.75
const total2 = invoice2.getTotal();   // total2 is 1070
```

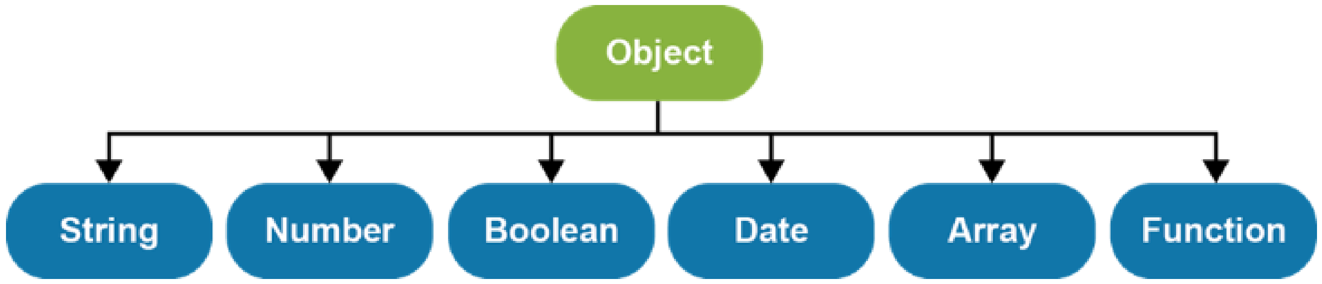
Terms related to classes

class

constructor

instance of a class

The JavaScript object hierarchy



The Person class

```
class Person {
    constructor(fname, lname) {
        this.firstName = fname;
        this.lastName = lname;
    }
    get fullName() {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

How to create and use a Person object

```
// create Person object
const p = new Person("Grace", "Hopper");

console.log(p.fullName);       // displays "Grace Hopper"
console.log(p instanceof Object);        // displays true
console.log(p instanceof Person);        // displays true
```

An Employee class that inherits the Person class

```
class Employee extends Person {
    constructor(fname, lname, hireDate) {
        super(fname, lname);
        this.hireDate = hireDate;
    }
}
```

How to create and use an Employee object

```
const emp = new Employee(
    "Bjarne", "Stroustrup", new Date("1/1/1979"));
console.log(emp.fullName);
                         // displays "Bjarne Stroustrup"
console.log(emp.hireDate.toDateString());
                         // displays "Mon Jan 01 1979"
console.log(emp instanceof Person);     // displays true
console.log(emp instanceof Employee);   // displays true
```

Terms related to inheritance

native object types

inheritance

inherit properties and methods

subclass

superclass

instanceof operator

A NumberArray class that uses inheritance

```
class NumberArray extends Array {
    constructor() {
        super();
        // defines a new property
        this.lastNumber = null;
    }
    // overrides an existing method
    push(num) {
        if(typeof num === "number") {
            super.push(num);
            this.lastNumber = num;
        } else {
            throw new TypeError(
                "NumberArray can only store numbers");
        }
    }
}
```

How to create and use a NumberArray object that uses inheritance

```
const arr = new NumberArray();
arr.push(1.07);
arr.push(2.21);
arr.push(3.14);
// arr.push("Grace");           // Would throw TypeError

console.log(arr.lastNumber);    // displays 3.14
console.log(arr.length);        // displays 3
console.log(arr.toString());    // displays 1.07,2.21,3.14

arr.unshift("Grace");    // PROBLEM! stores invalid value
```

A problem with the NumberArray class

Not all inherited methods that add elements to the array check to make sure the element is a number. As a result, it’s possible to add invalid data to the NumberArray object.

It makes sense to use inheritance when…

One object *is* a type of another object.

Both classes are part of the same logical domain.

The subclass primarily adds features to the superclass.

A NumberArray class that uses object composition

```
class NumberArray {
    constructor() {
        this._numbers = [];                // 'private' property
        this.lastNumber = null;
    }
    get length() {                         // read-only property
        return this._numbers.length;
    }
    push(num) {                            // method
        if(typeof num === "number") {
            this._numbers.push(num);
            this.lastNumber = num;
        } else {
            throw new TypeError(
                "NumberArray can only store numbers");
        }
    }
    toString() {                           // method
        return this._numbers.toString();
    }
}
```

How to create and use a NumberArray object that uses object composition

```
const arr = new NumberArray();
arr.push(1.07);
arr.push(2.21);
arr.push(3.14);
// arr.push("Grace");             // Would throw TypeError

console.log(arr.lastNumber);    // displays 3.14
console.log(arr.length);        // displays 3
console.log(arr.toString());    // displays
1.07,2.21,3.14

// arr.unshift("Grace");          // Would throw TypeError
                                // because method isn't
                                // defined
```

It makes sense to use object composition when…

One object *has* a type of another object.

The subclass restricts access to features of the superclass.

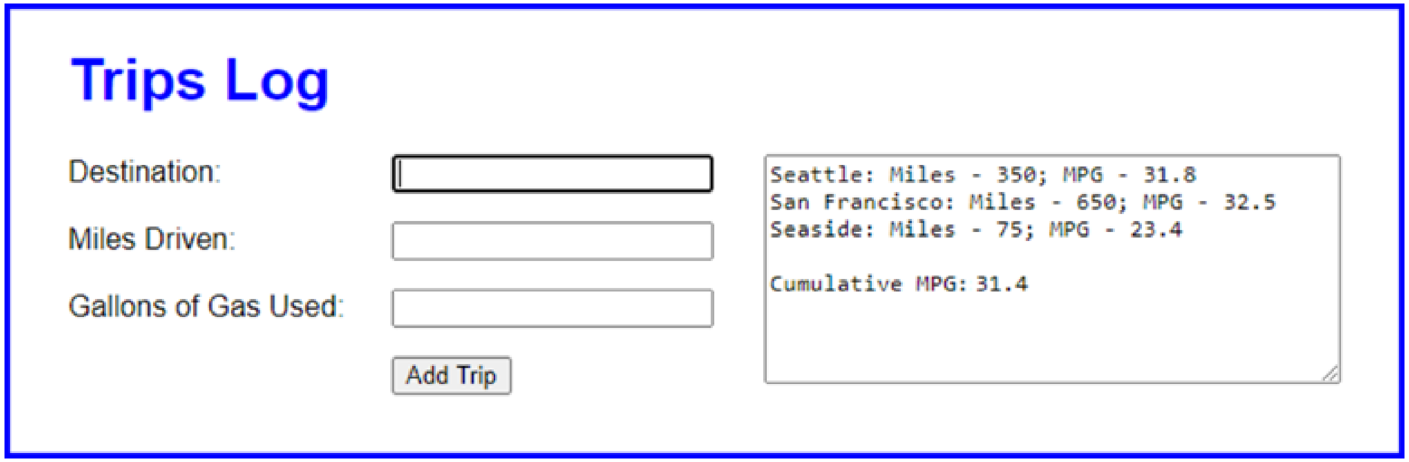
Terms related to object composition

object composition

encapsulation

data hiding

The Trips application



```
Seattle: Miles - 350; MPG - 31.8
San Francisco: Miles - 650; MPG - 32.5
Seaside: Miles - 75; MPG - 23.4

Cumulative MPG: 31.4
```

The HTML for the Trips application (part 1)

```
<body>
    <main>
        <h1>Trips Log</h1>
        <div id="trips">
            <textarea id="trip_list" rows="8" cols="42">
            </textarea>
        </div>
        <div>
            <label for="destination">Destination:</label>
            <input type="text" id="destination">
        </div>
        <div>
            <label for="miles">Miles Driven:</label>
            <input type="text" id="miles">
        </div>
        <div>
            <label for="gallons">Gallons of Gas Used:
            </label>
            <input type="text" id="gallons">
        </div>
```

The HTML for the Trips application (part 2)

```
            <div>
                <label></label>
                <input type="button" id="add_trip"
                       value="Add Trip">
            </div>
        </main>
        <script
            src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
        </script>
        <script src="lib_trips.js"></script>
        <script src="trips.js"></script>
    </body>
```

Some of the CSS for the Trips application

```
#trips {
    float: right;
}
```

The lib_trips.js file for the Trips app (part 1)

```
"use strict";

class Trip {
    constructor(destination, miles, gallons) {
        this.destination = destination;
        this.miles = parseFloat(miles);
        this.gallons = parseFloat(gallons);
    }

    get isValid() {              // a read-only property
        if (this.destination == "" || isNaN(this.miles) ||
                                     isNaN(this.gallons)) {
            return false;
        } else if (this.miles <= 0 || this.gallons <= 0){
            return false;
        } else {
            return true;
        }
    }

    get mpg() {                  // a read-only property
        return this.miles / this.gallons;
    }
```

The lib_trips.js file for the Trips app (part 2)

```
     toString() {
        const mpg = this.mpg.toFixed(1);
        return `${this.destination}: Miles - ${this.miles}; MPG - ${mpg}`;
    }
}

class Trips {
    constructor() {
        this._trips = [];
    }

    push(trip) {
        // only allow Trip objects to be added to array
        if (trip instanceof Trip) {
            this._trips.push(trip);
        }
    }

    get totalMpg() {            // a read-only property
        let totalMiles = 0;
        let totalGallons = 0;
        for (let trip of this._trips) {
            totalMiles += trip.miles;
            totalGallons += trip.gallons;
        }
        return totalMiles / totalGallons;
    }
```

The lib_trips.js file for the Trips app (part 3)

```
        toString() {
            let str = "";
            for (let trip of this._trips) {
                str += trip.toString() + "\n";
            }
            str += "\nCumulative MPG: " + this.totalMpg.toFixed(1);
            return str;
        }
    }
```

The trips.js file for the Trips application (part 1)

```
"use strict";

$(document).ready( () => {
    const trips = new Trips();

    $("#add_trip").click( () => {
        const trip = new Trip(
          $("#destination").val(), $("#miles").val(),
          $("#gallons").val());

        if (trip.isValid) {
            trips.push(trip);
            $("#trip_list").val(trips.toString());

            $("#destination").val("");
            $("#miles").val("");
            $("#gallons").val("");

            $("#destination").focus();
        }
```

The trips.js file for the Trips application (part 2)

```
        else {
            alert("Please complete all fields.\n" +
                  "Miles and gallons must be numeric " +
                  "and greater than zero.");
            $("#destination").select();
        }
    });

    $("#destination").focus();
});
```

Code that creates two instances of the Date type

```
const taxDay = new Date("4/15/2021");
const xmas = new Date("12/25/2021");

console.log(taxDay.toDateString());  // displays Thu Apr 15 2021
console.log(xmas.toDateString());    // displays Sat Dec 25 2021
```

Code that adds a method to the prototype object of the Date type

```
Date.prototype.toNumericDateString = function() {
    const m = this.getMonth() + 1;         // month is zero based
    const d = this.getDate();
    const y = this.getFullYear();
    return m + "/" + d + "/" + y;
};

console.log(taxDay.toNumericDateString()); // displays 4/15/2021
console.log(xmas.toNumericDateString());   // displays 12/25/2021
```

Code that adds an own property to one instance of the Date type

```
taxDay.hasExtension = true;

console.log(taxDay.hasExtension);   // displays true
console.log(xmas.hasExtension);     // displays undefined
```

Code that uses an own property to override a prototype property

```
xmas.toDateString = function() {
    return "It's Christmas Day";
};

console.log(taxDay.toDateString());
// displays Thu Apr 15 2021

console.log(xmas.toDateString());
// displays It's Christmas Day
```

Terms related to prototypes

prototype object

prototypal language

clone a prototype object

direct property

own property

override a method

A constructor function that creates Invoice objects

```
const Invoice = function(subtotal, taxRate) {
    this.subtotal = subtotal;
    this.taxRate = taxRate;
};
```

Code that adds a method to the Invoice object prototype

```
Invoice.prototype.getTotal = function() {
    const salesTax = this.subtotal * this.taxRate
    return this.subtotal + salesTax;
};
```

Code that uses the constructor function to create two Invoice objects

```
const invoice1 = new Invoice(100, 0.0875);
const invoice2 = new Invoice(1000, 0.07);
const total1 = invoice1.getTotal();    // total1 is 108.75
const total2 = invoice2.getTotal();    // total2 is 1070
```

A factory function that creates Invoice objects

```
const getInvoice = function(subtotal, taxRate) {
    const invoicePrototype = {   // define prototype methods
        getTotal: function() {
            const salesTax = this.subtotal * this.taxRate
            return this.subtotal + salesTax;
        }
    };
    const invoice = Object.create(invoicePrototype);
    invoice.subtotal = subtotal;
    invoice.taxRate = taxRate;
    return invoice;
};
```

Code that uses a factory function to create two Object types

```
const invoice1 = getInvoice(100, 0.0875);
const invoice2 = getInvoice(1000, 0.07);
const total1 = invoice1.getTotal();    // total1 is 108.75
const total2 = invoice2.getTotal();    // total2 is 1070
```

Terms related to legacy coding of objects

constructor function

factory function

How to use brackets to refer to properties and methods of an object

```
const invoice = {
    taxRate: 0.0875,                             // property
    getTotal(subtotal) {                         // method
        return subtotal + subtotal * this.taxRate;
    }
};

console.log(invoice.taxRate);                    // displays 0.0875
console.log(invoice["taxRate"]);                 // displays 0.0875

const total1 = invoice.getTotal(100);            // total1 is 108.75
const total2 = invoice["getTotal"](100);         // total2 is 108.75

// collides with getTotal()
invoice.getTotal = function(subtotal) {
    return subtotal + subtotal * 0.10;
};

const total3 = invoice.getTotal(100);            // total3 is 110.00
console.log(total3);
```

A symbol used as a computed property name

In an object

```
const taxRate = Symbol("taxRate");
const invoice = {
    [taxRate]: 0.0875
};

invoice.taxRate = 0.07;  // doesn't collide with taxRate symbol
console.log(invoice[taxRate]);                    // displays 0.0875
console.log(invoice.taxRate);                     // displays 0.07
```

In a class

```
const trips = Symbol("trips");
class Trips {
    constructor() {
        this[trips] = [];
    }
    // other methods that use the symbol named trips go here
}
```

Two of the well-known symbols

Symbol	Method Name	Description
`Symbol.hasInstance`	`@@hasInstance()`	Defines how an object behaves when used with the instanceof operator.
`Symbol.iterator`	`@@iterator()`	Defines how an object behaves when used with a for-of loop or with a spread operator that’s described in the next chapter.

One method required by an iterator object

Method	Description
`next()`	Executes the specified code and returns an IteratorResult object.

Two properties of the IteratorResult object

Property	Description
`done`	A Boolean value that indicates whether the iteration is done.
`value`	The value of the current item in the sequence. Can be omitted when done is true.

An object literal that's iterable

```
const taskList = {
    tasks: [],
    // methods of the taskList object go here
    [Symbol.iterator]() {     // define the iterator() method
        return {              // return the Iterator object
            tasks: this.tasks,
            index: 0,
            next() {    // required method of the Iterator object
                if (this.index == this.tasks.length) {
                    return {done: true};          // IteratorResult
                } else {
                    let value = this.tasks[this.index];
                    this.index++;
                    return {value, done: false};  // IteratorResult
                }
            }
        };
    }
};
```

A for-of loop that consumes the object

```
for (const task of taskList) {
    console.log(task.description)
}
```

A keyword that's used with generator functions

Keyword	Description
`yield`	Returns an IteratorResult object and pauses execution.

An iterable object literal with a generator function

```
const taskList = {
    tasks: [],
    // methods of the taskList object go here

    // define the iterator() method
    *[Symbol.iterator]() {
        for (let task of this.tasks) {
            yield task;
        }
    }
};
```

A for-of loop that consumes the object

```
for (const task of taskList) {
    console.log(task.description)
}
```

A Date object that's iterable

```
Date.prototype[Symbol.iterator] = function*() {
    yield this.getFullYear();
    yield this.getMonth();
    yield this.getDate();
    yield this.getHours();
    yield this.getMinutes();
    yield this.getSeconds();
};
```

A for-of loop that consumes an iterable Date object

```
const now = new Date();
for (const part of now) {
    console.log(part);   // displays 6 date parts
}
```

Two methods that modify an object but don’t return the object

Two methods of the taskList object

```
const taskList = {
    load() {
        // load code goes here
    },
    add(task) {
        // add code goes here
    }
};
```

Chaining these method calls doesn’t work

```
taskList.load().add(task);
// TypeError: Cannot read property 'add' of undefined
```

The methods must be called one at a time

```
taskList.load();
taskList.add(task);
```

Two methods that modify an object and then return the object

Two methods of the taskList object

```
const taskList = {
    load() {
        // load code goes here
        return this;
    },
    add(task) {
        // add code goes here
        return this;
    }
};
```

Chaining these method calls works

```
taskList.load().add(task);
```

A person object with three properties that have values

```
const person = {
    firstName: "Grace", lastName: "Hopper", dob: null
};
```

How to assign property values to variables or constants

Using the same names

```
const {firstName, lastName} = person;
```

Using different names

```
const {firstName: fname, lastName: lname} = person;
```

How to provide a default value for an assignment

```
const {firstName, dob, age = "unknown"} = person;

console.log(dob);   // displays null
console.log(age);   // displays "unknown"
```

How to destructure an object in the parameter list of a function

```
const displayGreeting = ({firstName, lastName}) => {
    console.log("Hello, " + firstName + " " + lastName);
};
```

Code that calls the function

```
displayGreeting(person);
// displays "Hello, Grace Hopper"

displayGreeting();
// TypeError: Cannot destructure property
```

An object with a nested object that has two properties

```
const invoice = {
    subtotal: 100,
    terms: {taxRate: 0.0875, dueDays: 30}    // nested object
};
```

How to assign property values of a nested object

```
const {subtotal, terms: {taxRate}} = invoice;
```

Static methods of the Object type (part 1)

Method	Description
`defineProperties(`*`obj`*`,`*`desc`*`)`	Define one or more properties on the object it receives. Sets the property attributes based on a descriptor object.
`keys(`*`obj`*`)`	An array of the names of the enumerable properties of the specified object, not including symbol properties.
`getOwnPropertyNames(`*`obj`*`)`	An array of the names of the enumerable and non-enumerable properties of the specified object, not including symbol properties.

Static methods of the Object type (part 2)

Method	Description
`getOwnPropertySymbols(obj)`	An array of the symbol properties of the specified object.
`getOwnPropertyDescriptors(obj)`	A descriptor object for all the properties in the specified object, including symbol properties.

A statement for working with properties

Name	Description
`for-in`	A statement that loops through the properties of an object, not including non-enumerable and symbol properties.

An operator for working with properties

Name	Description
`in`	An operator that returns a Boolean value that indicates if the specified property is in the object, including non-enumerable and symbol properties.

An example that works with properties

```
const pet = {
    name: "Maisie"         // configurable, writable, and enumerable
};

// add three properties to the object
Object.defineProperties(pet, {
    id: {value: "1234"},   // NOT configurable, writable, or enumerable
    adoptionDate: {writable: true},
    species: {writable: true, enumerable: true}
});

// inspect the object with a for-in loop and the in operator
for (let propName in pet) {
    console.log(propName);  // only displays "name" and "species"
}
console.log("adoptionDate" in pet);            // displays true
console.log("id" in pet);                      // displays true

// get information about the object
Object.keys(pet);                 // ["name","species"]
Object.getOwnPropertyNames(pet);  // ["name","id","adoptionDate","species"]
Object.getOwnPropertySymbols(pet); // []
Object.getOwnPropertyDescriptors(pet);  // descriptor for each property
```

More terms related to objects

symbol

name collision

well-known symbols

iterator

generator function

yield statement

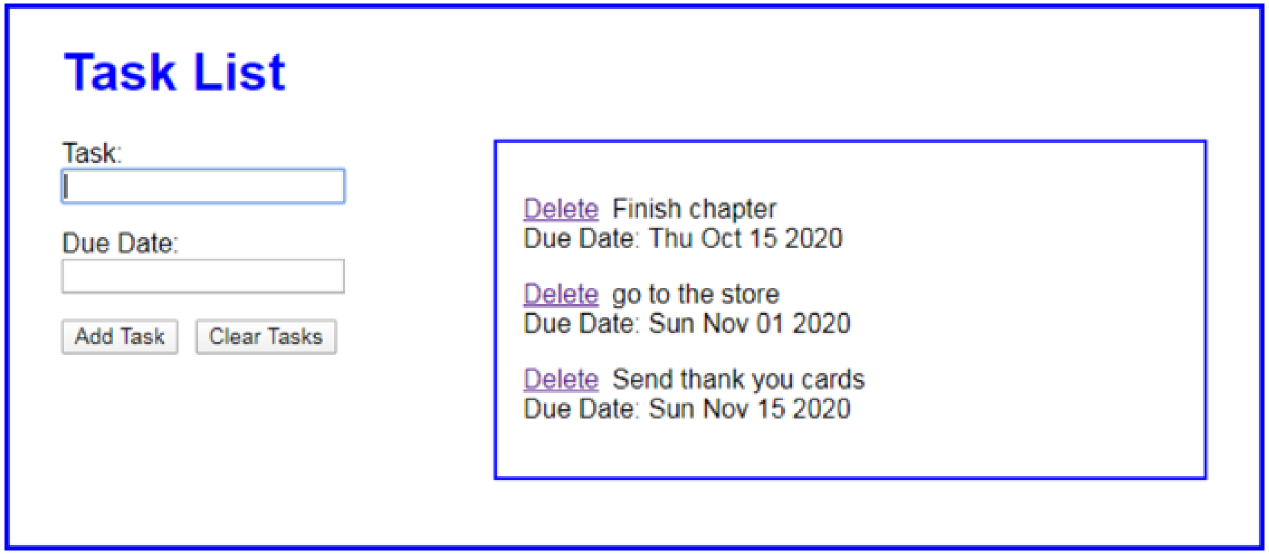
cascading method

fluent code

method chaining

destructuring

The Task List application



Task List

Task:

Due Date:

Delete Finish chapter
Due Date: Thu Oct 15 2020

Delete go to the store
Due Date: Sun Nov 01 2020

Delete Send thank you cards
Due Date: Sun Nov 15 2020

The HTML for the Task List application (part 1)

```
<body>
    <main>
        <h1>Task List</h1>
        <div id="tasks"></div>
        <div>
            <label for="task">Task:</label>
            <input type="text" name="task" id="task">
        </div>
        <div>
            <label for="due_date">Due Date:</label>
            <input type="text" name="due_date" id="due_date">
        </div>
        <div>
            <input type="button" id="add_task" value="Add Task">
            <input type="button" id="clear_tasks"
                   value="Clear Tasks">
        </div>
        <div class="clear"></div>
    </main>
```

The HTML for the Task List application (part 2)

```
        <script
            src="https://code.jquery.com/jquery-3.4.1.slim.min.js">
        </script>
        <script src="lib_task.js"></script>
        <script src="lib_storage.js"></script>
        <script src="lib_task_list.js"></script>
        <script src="task_list.js"></script>
    </body>
```

Some of the CSS for the Task List application

```
#tasks {
    float: right;
    width: 25em;
    margin: 0 0 .5em;
    padding: 1em;
    border: 2px solid black;
    min-height: 5em;
}
#tasks a {
    margin-right: 0.5em;
}
.clear {
    clear: both;
}
```

The lib_task.js file for the Task List app (part 1)

```
"use strict";

class Task {
    constructor({description, dueDate}) {  // uses destructuring
        this.description = description;
        if (dueDate) {
            this.dueDate = new Date(dueDate);
        } else {
            this.dueDate = new Date();
            this.dueDate.setMonth(this.dueDate.getMonth() + 1);
        }
    }

    get isValid() {
        if (this.description === "") {
            return false;
        }
        const today = new Date();
        if (this.dueDate.getTime() <= today.getTime() ) {
            return false;
        }
        return true;
    }
```

The lib_task.js file for the Task List app (part 2)

```
        toString() {
            return `${this.description}<br>
                    Due Date: ${this.dueDate.toDateString()}`;
        }
    }
```

The lib_storage.js file for the Task List app

```
"use strict";

const storage = {
    retrieve() {
        const tasks = [];
        const json = localStorage.tasks;
        if(json) {
            const taskArray = JSON.parse(json);
            for(let obj of taskArray) {
                tasks.push(new Task(obj)); // uses destructuring
            }
        }
        return tasks;
    },
    store(tasks) {
        localStorage.tasks = JSON.stringify(tasks);
    },
    clear() {
        localStorage.tasks = "";
    }
};
```

The lib_task_list.js file for the Task List app (part 1)

```
"use strict";

const tasks = Symbol("tasks");

const taskList = {
    [tasks]: [],
    load() {
        this[tasks] = storage.retrieve();
        return this;
    },
    save() {
        storage.store(this[tasks]);
        return this;
    },
```

The lib_task_list.js file for the Task List app (part 2)

```
sort() {
    this[tasks].sort( (task1, task2) => {
        if (task1.dueDate < task2.dueDate) {
            return -1;
        } else if (task1.dueDate > task2.dueDate) {
            return 1;
        } else {
            return 0;
        }
    });
    return this;
},
add(task) {
    this[tasks].push(task);
    return this;
},
delete(i) {
    this.sort();
    this[tasks].splice(i, 1);
    return this;
},
```

The lib_task_list.js file for the Task List app (part 3)

```
    clear() {
            storage.clear();
            return this;
        },
        *[Symbol.iterator]() {
            for (let task of this[tasks]) {
                yield task;
            }
        }
   };
```

The task_list.js file for the Task List app (part 1)

```
"use strict";

const displayTasks = () => {
    taskList.sort();

    let html = "";
    for (const task of taskList) {
        html += "<p><a href='#'>Delete</a>" +
        task.toString() + "</p>";
    }
    $("#tasks").html(html);

    // add click() event handler to each <a> element
    $("#tasks").find("a").each( (index, a) => {
        $(a).click( evt => {
            taskList.load().delete(index).save();
            displayTasks();
            evt.preventDefault();
            $("input:first").focus();
        });
    });
}
```

The task_list.js file for the Task List app (part 2)

```
$(document).ready( () => {
    $("#add_task").click( () => {
        const taskObj = {                      // object literal
            description: $("#task").val(),
            dueDate: $("#due_date").val()
        };
        const newTask = new Task(taskObj);  // Task object

        if (newTask.isValid) {
            taskList.load().add(newTask).save();
            displayTasks();
            $("#task").val("");
            $("#due_date").val("");
        } else {
            alert("Please enter a task and/or " +
                  "a due date that's in the future.");
        }
        $("#task").select();
    });
```

The task_list.js file for the Task List app (part 3)

```
        $("#clear_tasks").click( () => {
            taskList.clear();
            $("#tasks").html("");
            $("#task").val("");
            $("#due_date").val("");
            $("#task").focus();
        });

        taskList.load()
        displayTasks();
        $("#task").focus();
    });
```