Comp 324/424 - Client-side Web Design

Spring Semester 2024 Week 7

Dr Nick Hayward

JS Core - checking equality - part 1

- JS has four equality operators, including two **not equal**
  - `==` , `===` , `!=` , `!==`
- `==` - checks for value equality, whilst allowing coercion
- `===` - checks for value equality but without coercion

```
var a = 49;
var b = "49";

console.log(a == b); //returns true
console.log(a === b); //returns false
```

- first comparison checks values
  - if necessary, try to coerce one or both values until a match occurs
  - allows JS to perform a simple equality check
  - results in `true`
- second check is simpler
  - coercion is not permitted, and a simple equality check is performed
  - results in `false`

JS Core - checking equality - part 2

- which comparison operator should we use
- useful suggestions for usage of comparison operators
  - use `===` if either side of the comparison could be true or false
  - use `===` if either value could be one of the following specific values,
    - `0` , `""` , `[]`
  - otherwise, it's safe to use `==`
  - simplify code in a JS application due to the implicit coercion.
- **not equal** counterparts, `!` and `!==` work in a similar manner

---

JS Core - checking inequality - part 1

- known as **relational comparison**, we can use the inequality operators,
  - `<` , `>` , `<=` , `>=`
- inequality operators often used to check comparable values like numbers
  - inherent ordinal check
- can be used to compare strings

```
"hello" < "world"
```

- coercion also occurs with inequality operators
  - no concept of **strict inequality**

```
var a = 49;
var b = "59";
var c = "69";

a < b; //returns true
b < c; //returns true
```

JS Core - checking inequality - part 2

- we can encounter an issue when either value cannot be coerced into a number

```
var a = 49;
var b = "nice";

a < b; //returns false
a > b; //returns false
a == b; //returns false
```

- issue for `<` and `>` is string is being coerced into invalid number value, `NaN`
- `==` coerces string to `NaN` and we get comparison between `49 == NaN`

HTML5, CSS, & JS - example - part 14

interaction - add a note - abstract code

- need to create a new function to abstract
  - creation and output of a new note
  - manage the input field for our note app
- moving logic from button click function to separate, abstracted function
- then call this function as needed
  - for a button click or keyboard press
  - then create and render the new note

```
// create a note
// - input = value from input field
// - output = DOM node for output of new note
function createNote(input, output) {
    // create p node
    let p = document.createElement('p');
    // get value from input field for note
    let inputVal = input.value;
    // check input value
    if (inputVal !== '') {
      // create text node
      let noteText = document.createTextNode(inputVal);
      // append text to paragraph
      p.appendChild(noteText);
      // append new paragraph and text to existing note output
      output.appendChild(p);
      // clear input text field
      input.value = '';
    }
}
```

HTML5, CSS, & JS - example - part 15

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  let noteOutput = document.querySelector('.note-output');
  // add note button
  let addNoteBtn = document.getElementById('add-note');
  // input field for add note
  let inputNote = document.getElementById('input-note');

  // add event listener to add note button
  addNoteBtn.addEventListener('click', () => {
      createNote(inputNote, noteOutput);
  });

  // add event listener for keypress in note input field
  inputNote.addEventListener('keypress', (e) => {
    // check key pressed by code - 13 - return
    if (e.keyCode === 13) {
      createNote(inputNote, noteOutput);
    }
  });

}

// load app
travelNotes();
```

interaction - add a note - plain JS

- DEMO - travel notes - series 1

HTML5, CSS, & JS - example - part 16

interaction - add a note - animate

- JavaScript well-known for is its simple ability to animate elements
- many built-in effects available in various JS animation libraries
  - build our own as well
- to `fadeIn` an element, effectively it needs to be hidden first
- we hide our newly created note
- then we can set it to `fadeIn` when ready
  - ...
- DEMO - travel notes - series 1

CSS Basics - complex selector - part 1

- our DOM will often become more complicated and detailed
- depth and complexity will require more complicated selectors as well
- lists and their list items are a good example

```
<ul>
  <li>unordered first</li>
  <li>unordered second</li>
  <li>unordered third</li>
</ul>
<ol>
  <li>ordered first</li>
  <li>ordered second</li>
  <li>ordered third</li>
</ol>
```

- two lists, one unordered and the other ordered
- style each list, and the list items using rulesets

```
ul {
  border: 1px solid green;
}
ol {
  border: 1px solid blue;
}
```

Demo - Complex Selectors - Part 1

- Demo - Complex Selectors Part 1

CSS Basics - complex selector - part 2

- add a ruleset for the list items, `<li>`
- applying the same style properties to both types of lists
- more specific to apply a ruleset to each list item for the different lists

```
ul li {
  color: blue;
}
ol li {
  color: red;
}
```

- also be useful to set the background for specific list items in each list

```
li:first-child {
  background: #bbb;
}
```

- pseudoclass of `nth-child` to specify a style for the second, fourth &c. child in the list

```
li:nth-child(2) {
  background: #ddd;
}
```

Demo - Complex Selectors - Part 2

- Demo - Complex Selectors Part 2

CSS Basics - complex selector - part 3

- style odd and even list items to create a useful alternating pattern

```
li:nth-child(odd) {
  background: #bbb;
}
li:nth-child(even) {
  background: #ddd;
}
```

- select only certain list items, or rows in a table &c.
  - e.g. every fourth list item, starting at the first one

```
li:nth-child(4n+1) {
  background: green;
}
```

- for **even** and **odd** children we're using the above with convenient shorthand
- other examples include
  - `last-child`
  - `nth-last-child()`
  - many others...

Demo - CSS Complex Selectors - Part 3

- Demo - Complex Selectors Part 3

HTML5, CSS, & JS - example - part 17

style and render notes

- we have some new notes in our app
- add some styling to help improve the look and feel of a note
- can set background colours, borders font styles...
- set differentiating colours for each alternate note
- allows us to try some pseudoclasses in the CSS
  - specified paragraphs in the `note-output` section

```
.note-output p:nth-child(even) {
  background-color: #ccc;
}
.note-output p:nth-child(odd) {
  background-color: #eee;
}
```

- DEMO - travel notes - series 1

HTML5, CSS, & JS - final thoughts

- a basic app that records simple notes
- many additional options we can add
- some basic functionality is needed to make it useful
  - autosave - otherwise we lose our data each time we refresh the browser
  - edit a note
  - delete a note
  - add author information
- additional functionality might include
  - save persistent data to DB, name/value pairs…
  - organise and view collections of notes
  - add images and other media
    - local and APIs
  - add contextual information
    - again, local and APIs
  - structure notes, media, into collection
  - define related information
  - search, sort…
  - export options and sharing…
- security, testing, design patterns

Video - Scotoma - Da Vinci Code

Scotoma - The Da Vinci Code - Source: YouTube

Image - HTML5, CSS, & JS - DOM recap

Image - Travel Notes - Series 1 - recap

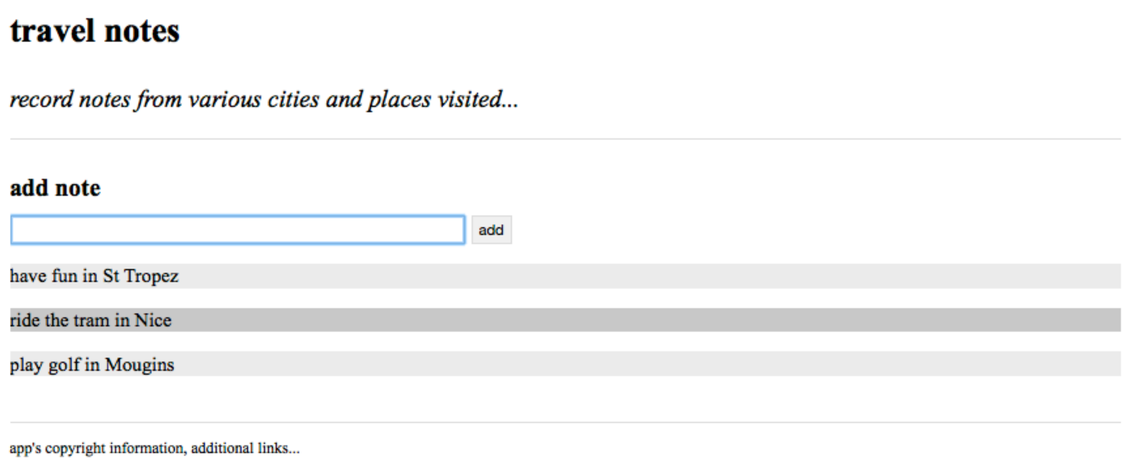
HTML5, CSS, & JS - example - add-ons

new features and add-ons…

- delete all notes
- delete a single note
- new event handlers
- additional styling

```
                            |----------|
                            |  <body>  |
                            |----------|
                                 |
  + -------------------------|--------------------------+
  |                              |                          |
|----------|                |----------|             |----------|
| <header> |                |  <main>  |             | <footer> |
|----------|                |----------|             |----------|
     |                           |                          |
+-----------+           +-------------------+               |
|           |           |                   |               |
|------|  |-----|  |-----------|     |-----------|      |-----|
| <h3> |  | <p> |  | <section> |     | <section> |      | <p> |
|------|  |-----|  |-----------|     |-----------|      |-----|
                         |
                +----------------+
                |                |
            |------|       |---------|
            | <h5> |       | <input> |
            |------|       |---------|
```

Figure 1: Travel Notes - DOM recap



travel notes

record notes from various cities and places visited...

add note

have fun in St Tropez

ride the tram in Nice

play golf in Mougins

app's copyright information, additional links...

Figure 2: Travel Notes - Series 1 - Demo 8 recap

HTML5, CSS, & JS - Series 2 - part 1.2

```
// delete all notes button
let deleteAll = document.getElementById('notes-delete');

// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // loop through notes and remove a single note per iteration...
    for (let note of notes) {
        note.remove();
    }
});
```

delete option - all notes - plain js

- option to **delete all** notes from `.note-output`
- add a new toolbar for note controls and options

```
<section class="note-controls">
  <button id="notes-delete">Delete all</button>
</section>
```

- then add some simple styling for this new toolbar

```
/* note controls */
.note-controls {
  margin: 10px 0 10px 0;
  padding: 2px;
  border-bottom: 1px solid #dedede;
  display: none;
}
/* simplify default button styles for note controls */
.note-controls button {
  padding: 2px;
  margin: 2px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

Video - HTML5, CSS, & JS

display vs visibility CSS - Display versus Visibility - UP TO 1:46

Source - CSS Display and Visibility - YouTube

HTML5, CSS, & JS - example - part 2.2

delete option - all notes - plain js

- note controls toolbar is hidden, by default in the CSS
- need some way to check its visibility as we add our notes

    - no notes, then the toolbar is not required
- use `display` property to check node

```
// check visibility of passed node
function checkVisible(node) {
    // check passed node's current visibility
    if (node.style.display != 'block') {
        // show in DOM to allow fadeIn...
        node.style.display = 'block';
        // call fadeIn for node in DOM
        fadeIn(node);
    }
}
```

- simply checking a passed element to see whether it is hidden
- can update this method later on to check hidden and visible
- call this function as required
- & usage with a defined node

```
// define node to check...
let controls = document.getElementById('controls');
// call function
checkVisible(controls);
```

HTML5, CSS, & JS - example - part 2.3

delete option - all notes - plain js

- use `visibility` property to check node

```
// check visibility of passed node
function checkVisible(node) {
    // check passed node's current visibility
    if (node.style.visibility = 'hidden') {
    // show in DOM to allow fadeIn...
    node.style.display = 'block';
    node.style.visibility = 'visible';
    // call fadeIn type animation &c. for node in DOM
    fadeIn(node);
    }
}
```

JS Core - more conditionals - part 1

- briefly considered conditional statements using the `if` statement,

```
if (a > b) {
console.log("a is the best...");
} else {
console.log("b is the best...");
}
```

- Switch statements effectively follow the same pattern as `if` statements
    - designed to allow us to check for multiple values in a more succinct manner

    - enable us to check and evaluate a given expression
    - then attempt to match a required value against an available `case`
- addition of `break` is important, ensures only matched case is executed
    - then the application breaks from the switch statement
- if no `break` execution after that case will continue
    - commonly known as **fall through**
    - may be an intentional feature of your code design
    - allows a match against multiple possible cases

JS Core - switch conditional - example

```
var a = 4;

switch (a) {
case 3:
  //par 3
  console.log("par 3");
  break;
case 4:
  //par 4
  console.log("par 4");
  break;
case 5:
  //par 5
  console.log("par 5");
  break;
case 59:
  //dream score
  console.log("record");
  break;
default:
  console.log("more practice");
}
```

JS Core - more conditionals - part 2

ternary

- a more concise way to write our conditional statements
- known as the **ternary** or **conditional** operator
- consider this operator a more concise form of standard `if...else` statement

```
var a = 59;
var b = (a > 59) ? "high" : "low";
```

- equivalent to the following standard `if...else` statement

```
var a = 59;

if (a > 59) {
  var b = "high";
} else {
  var b = "low";
}
```

HTML5, CSS, & JS - example - part 4.2

JS code so far - plain JS

- add a note, the `.note-controls` toolbar is shown
  - **delete all** button now becomes available to our users

```
// delete all notes button
let deleteAll = document.getElementById('notes-delete');

// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // hide parent controls node...
    deleteAll.parentNode.style.display = 'none';
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // loop through notes and remove a single note per iteration...
    for (let note of notes) {
        // remove single node
        note.remove();
    }
});
```

- hide parent node for controls…
- DEMO 1 - travel notes - series 2

Video - Fitts' Law

mouse pointers and Fitts' law Mouse Pointers & Fitts's Law

Source - Mouse Pointers & Fitts's Law - Computerphile - YouTube

---

HTML5, CSS, & JS - example - part 5

delete option - all notes

- still making an assumption notes exist in the note-output section
- add an additional function to check element exists in the DOM or not
- use `length` property

```
element.length
```

- new function for checking elements in the DOM

```
//check elements exists
function checkExist(element) {
  if (element.length) {
    return true;
  } else {
    return false;
  }
}
```

HTML5, CSS, & JS - example - part 6.2

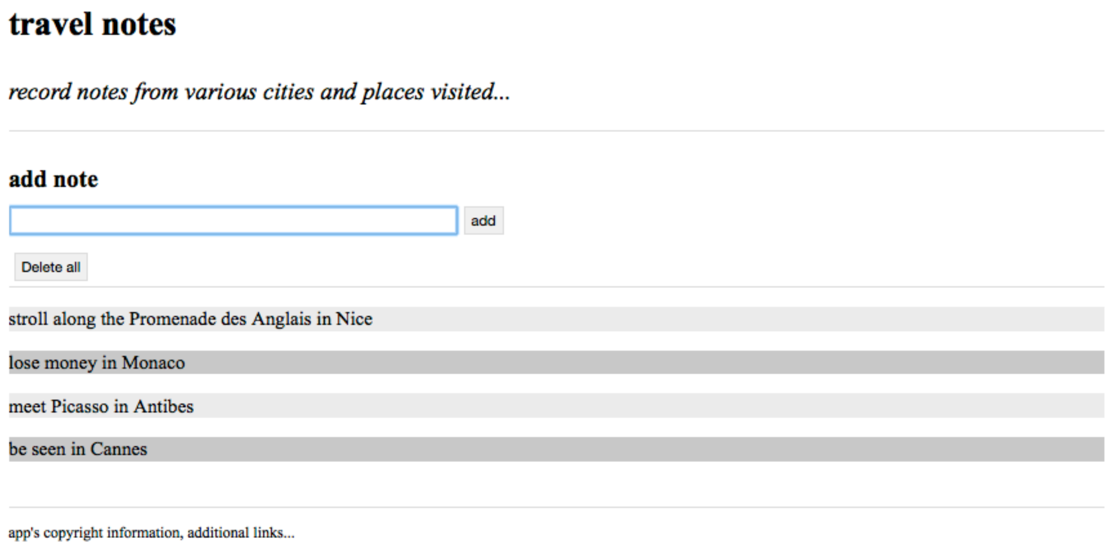
delete option - all notes - plain JS

- updated delete all notes option to include check for notes
- call `checkExist()` function in conditional statement

```
// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // check notes in DOM
    if (checkExist(notes) === true) {
        // hide parent controls node...
        deleteAll.parentNode.style.display = 'none';
        // loop through notes and remove a single note per iteration...
        for (let note of notes) {
            // remove single node
            note.remove();
        }
    }
});
```

- DEMO 2 - travel notes - series 2

Image - Travel Notes - Series 2 - demo 2


travel notes

record notes from various cities and places visited...

add note

stroll along the Promenade des Anglais in Nice

lose money in Monaco

meet Picasso in Antibes

be seen in Cannes

app's copyright information, additional links...

Figure 3: Travel Notes - Series 2 - Demo 2

Video - HTML5, CSS, & JS

white space / negative space - part 1 UI Design - How to use Negative Space in UI Design - UP TO 3:32

Source - UI Design - White space or Negative Space - YouTube

HTML5, CSS, & JS - example - part 7

delete option - per note

- consider adding a single delete option per note
- allowing a user to selectively delete their chosen note
  - regardless of hierarchical position within the `.note-output` section
- design decisions for such an option might include
  - do we offer a selection option, such as checkboxes, to select one or more delete items
  - perhaps a single delete button per note
  - a drag and drop to delete option
  - there are many different ways to present and use this option
- programmatically follow a similar pattern for deletion of the note

---

HTML5, CSS, & JS - example - part 8.2

delete option - per note - plain js

- simply need to delete the selected note
  - use the same `remove()` function for single and all notes
- add option per note to allow user to delete a required note
- add a delete button for each note
  - add programmatically with each new note

```
// create button element - pass class and text
function createButton(btnClass, btnTxt) {
    // create button node
    let btnNode = document.createElement('button');
    // create button text node
    let btnTxtNode = document.createTextNode(btnTxt);
    // set attribute on button node
    btnNode.setAttribute('class', btnClass);
    // append text to button
    btnNode.appendChild(btnTxtNode);
    // return new button node with text and attribute...
    return btnNode;
}
```

- new function allows us to create simple buttons as required
  - a specified class and button text passed as parameters
  - use function to build required delete button in `createNote()` function
- then call as required,

```
// create delete button for note
let delButton = createButton('note-delete', 'delete');
```

HTML5, CSS, & JS - example - part 9.2

delete option - per note - plain js

- append/prepend delete option to note
  - before adding note to the DOM in `createNote` function

```
function createNote(input, output) {
  // get value from input field for note
    let inputVal = input.value;

    // check input value
    if (inputVal !== '') {
    // create p node
    let p = document.createElement('p');
        // create delete button for note
        let delButton = createButton('note-delete', 'delete');
        // prepend button to note
        p.prepend(delButton);
        // create text node
        let noteText = document.createTextNode(inputVal);
        // append text to paragraph
        p.appendChild(noteText);
        // append new paragraph and text to existing note output
        output.appendChild(p);
        // call custom animation for fade in...
        //fadeIn(p);
        // clear input text field
        input.value = '';
    }

    let controls = document.getElementById('app-controls');
    checkVisible(controls);
}
```

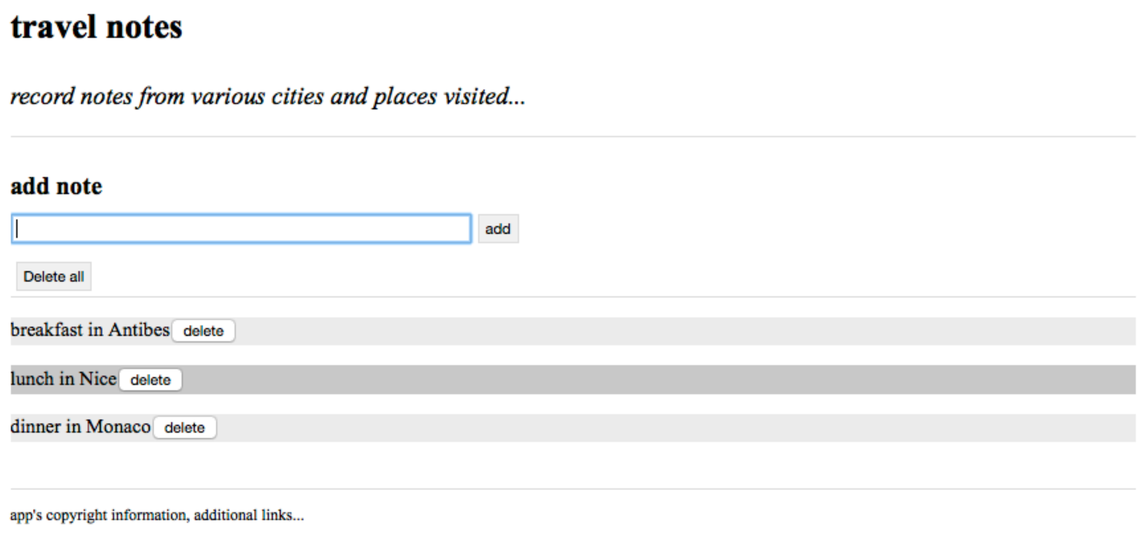
Image - Travel Notes - Series 2 - demo 3

- DEMO 3 - travel notes - series 2

HTML5, CSS, & JS - example - part 11

delete option - per note

- now allow our users to delete a single note
- single note option is awkward at the moment
- simply allow a user to either mouseover or select a note to show additional options
  - showing the available delete button
- enable a user to select their note of choice
  - need to bind a click event to a note
- user selects a note
  - no check for previous other visible delete buttons



travel notes

record notes from various cities and places visited...

add note

breakfast in Antibes delete

lunch in Nice delete

dinner in Monaco delete

app's copyright information, additional links...

Figure 4: Travel Notes - Series 2 - Demo 3

- ensure only delete button for selected note is shown

JS Core - functions and values

- variables acting as groups of code and blocks
- act as one of the primary mechanisms for scope within our JS applications
- also use functions as values
- effectively using them to set values for other variables

```
var a;

function scope() {
  "use strict";
    a = 49;
  return a;
}

b = scope() * 2;
console.log(b);
```

- useful and interesting aspect of the JS language
  - allows us to build values from multiple layers and sources

---

Image - HTML5, CSS, & JS - too many delete buttons

HTML5, CSS, & JS - example - part 12.1

delete option - per note

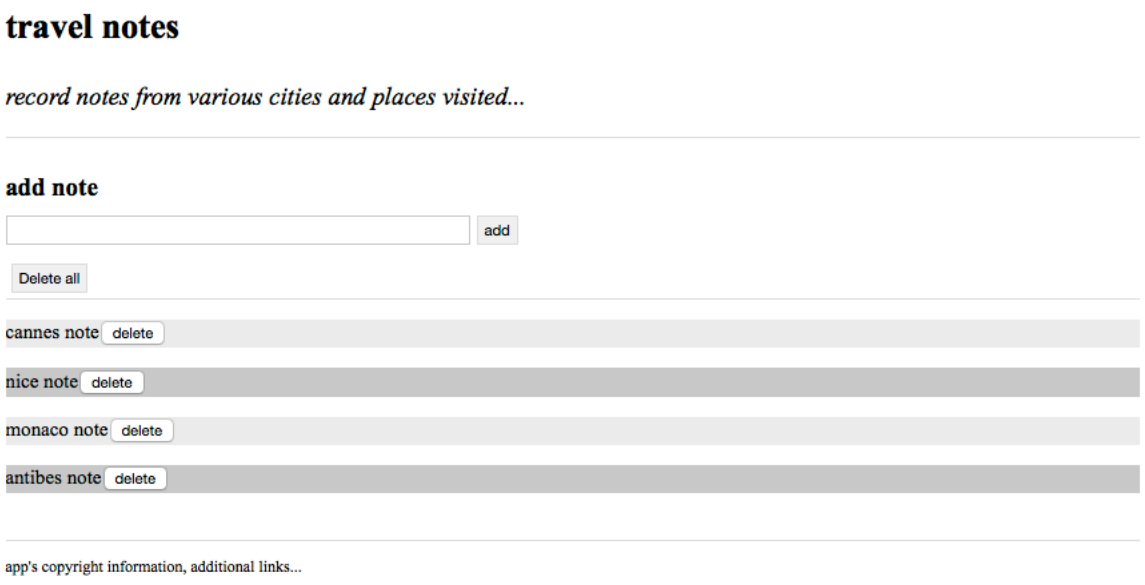

travel notes
record notes from various cities and places visited...
add note
add
Delete all
cannes note
delete
nice note
delete
monaco note
delete
antibes note
delete
app's copyright information, additional links...

Figure 5: Travel Notes - Week 6 - Too many delete buttons

- return to our earlier function, `checkVisible()`
- modify to allow better abstraction and usage
- modify to test for visibility
  - then simply return a boolean value

```
//check element visibility - expects single element relative to display:none
function checkVisible(element) {
  //check if element is hidden or not
  if (element.is(":hidden")) {
    return true;
  } else {
    return false;
  }
}
```

- also need to modify check for the `.note-controls` in `createNote()` function

```
...
//check visibility of note controls
if (checkVisible($(".note-controls")) === true) {
    // animate showing of note controls...
}
...
```

HTML5, CSS, & JS - example - part 12.2

delete option - per note - plain js

- note delete button

```
// add delete button for current note
// use anonymous FN instead of arrow FN
// this binds to clicked DOM node
delButton.addEventListener('click', function () {
    console.log('note delete...', this.parentNode);
    this.parentNode.remove();
});
```

- note delete button with check for notes
  - no notes - hide *delete all* option

```
// add delete button for current note
// use anonymous FN instead of arrow FN
// this binds to clicked DOM node
delButton.addEventListener('click', function () {
    console.log('note delete...', this.parentNode);
    this.parentNode.remove();
    // get notes from DOM
    let notes = output.querySelectorAll('p');
    if (checkExist(notes) === false) {
        controls.style.display = 'none';
    }
});
```

- DEMO 3 - travel notes - series 2 - plain JS

HTML5, CSS, & JS - example - part 13.2

delete option - per note - plain JS

- check for current delete buttons per note
  - hide each delete button
  - then, show delete button for current note…

```
// click listener for note
p.addEventListener('click', function() {
    // get notes delete buttons from DOM
    let delBtns = output.querySelectorAll('.note-delete');
    if (checkExist(delBtns) === true) {
        for (let btn of delBtns) {
            btn.style.display = 'none';
        }
    }
    this.querySelector('.note-delete').style.display = 'inline';
});
```

- bind handler for the user clicking on a note
- check whether other delete buttons are visible on any other notes
  - if visible, we can simply hide these delete buttons
  - then show the delete option for the currently selected note
- later abstract this function to handle other options associated with each note
  - DEMO 4 - travel notes - series 2
  - version 1
  - version 2

HTML5, CSS, & JS - example - part 14

style note(s)

- add some additional styling to our notes
  - start with some changes to the design of each note
  - then considered the overall `.note-output` section
- remove styling for alternating notes, set uniform style per note

```css
/* note paragraph output */
.note-output p {
  margin: 10px;
  padding: 10px;
  border: 1px solid #b1c4b1;
  cursor:pointer;
}
```

- need to add some styling for our delete button, and position it within each note

```css
/* note delete button */
.note-output p button.note-delete {
  display: block;
  padding: 5px;
  margin: 5px 5px 10px 0;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

HTML5, CSS, & JS - example - part 15

style note(s)

- add some styling for the button's hover pseudo-class
  - acts as useful feedback to the user that the button is an active element

```css
.note-output p button.note-delete:hover {
  background-color: #aaa;
  color: #fff;
}
```

- also consider adding some similar feedback to our note
  - a sign of active as the user moves their mouse cursor over each note

```css
/* note paragraph output hover */
.note-output p:hover {
  border: 1px solid #1a3852;
}
```

- DEMO 5 - travel notes - series 2
  - version 1
  - version 2

HTML5, CSS, & JS - example - part 16

style note(s)

- a couple of issues that still need to be fixed in the application
  - first issue is lack of consistency in styling our buttons
- fixed by abstracting our CSS styling for a default button
  - specific button styles can be added later

```css
/* default button style */
button {
  padding: 2px;
  margin: 2px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

- removed the need for a ruleset to style the button for
  - adding a note, delete all notes, and the single delete button per note

---

HTML5, CSS, & JS - example - part 17

style note(s)

- also create a default ruleset for a button hover pseudo-class
  - again reducing our need for repetition in the stylesheet

```css
/* default button hover style */
button:hover {
  background-color: #aaa;
  color: #fff;
}
```

- iterative development is fine
  - continue to abstract styles, overall design, and logic as we develop an application

---

HTML5, CSS, & JS - example - part 19

a few extras to consider...

- alternative layouts
  - grid
  - squares
  - snippet view
  - table
  - lists...
- notifications
- snippets with expansion
- split views
  - note snippet with contextual/media per note...
- drag and drop delete
- filters
- sort options
- tags

- much, much more...

Image - Square notes - a bit of fun

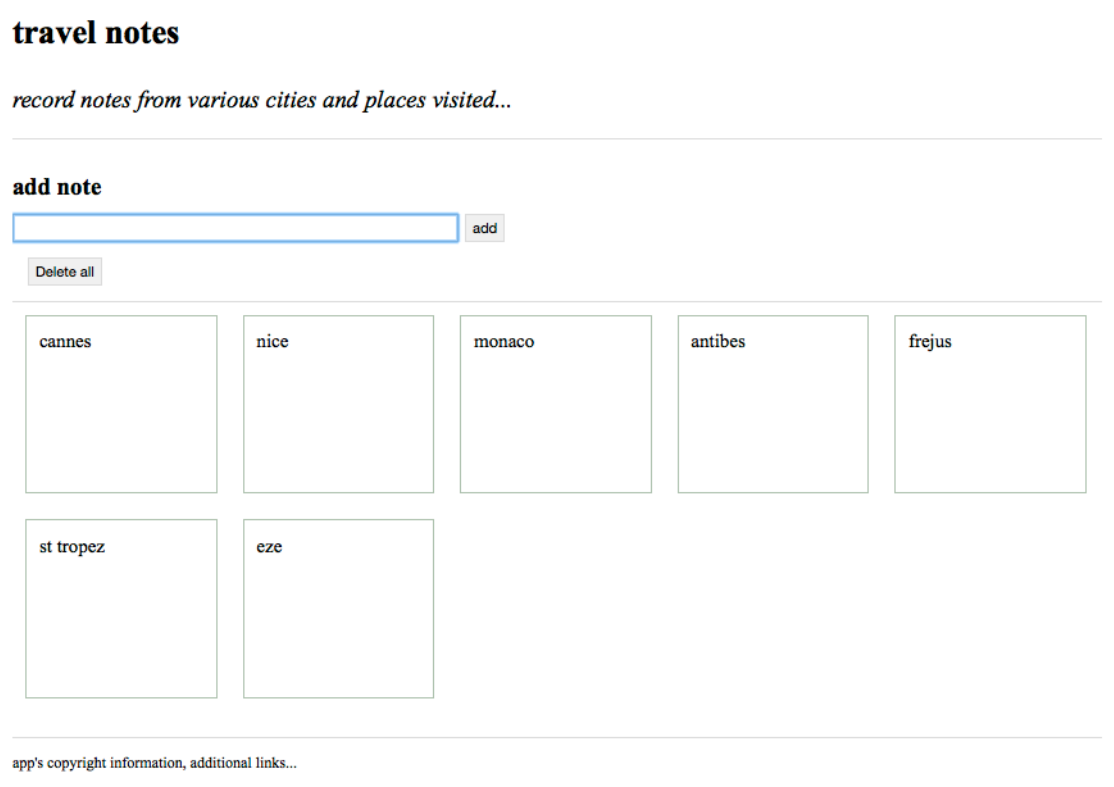

travel notes
record notes from various cities and places visited...
add note
add
Delete all
cannes
nice
monaco
antibes
frejus
st tropez
eze
app's copyright information, additional links...

Figure 6: Travel Notes - Week 6 - Squares

- DEMO - travel notes - squares

Video - HTML5, CSS, & JS

white space / negative space - part 2 UI Design - How to use Negative Space in UI Design - UP TO 5:17

Source - UI Design - White space or Negative Space - YouTube

JS extras - best practices - part 1

a few best practices...

variables

- limit use of global variables in JavaScript
  - easy to override

    - can lead to unexpected errors and issues
    - should be replaced with appropriate local variables, closures
- local variables should always be declared with keyword `var`
    - avoids automatic global variable issue

declarations

- add all required declarations at the top of the appropriate script or file
    - provides cleaner, more legible code
    - helps to avoid unnecessary global variables
    - avoid unwanted re-declarations

types and objects

- avoid declaring numbers, strings, or booleans as objects
- treat more correctly as primitive values
    - helps increase the performance of our code
    - decrease the possibility for issues and bugs

JS extras - best practices - part 2

type conversions and coercion

- weakly typed nature of JS
    - important to avoid accidentally converting one type to another
    - converting a number to a string or mixing types to create a NaN (Not a Number)
- often get a returned value set to NaN instead of generating an error
    - try to subtract one string from another may result in `NaN`

comparison

- better to try and work with `===` instead of `==`
    - `==` tries to coerce a matching type before comparison
    - `===` forces comparison of values and type

defaults

- when parameters are required by a function
    - function call with a missing argument can lead to it being set as **undefined**
    - good coding practice to assign default values to arguments
    - helps prevent issues and bugs

switches

- consider a `default` for the switch conditional statement
- ensure you always set a `default` to end a switch statement

JS extras - performance - part 1

loops

- try to limit the number of calculations, executions, statements performed per loop iteration
- check loop statements for assignments and statements
    - those checked or executed once
    - rather than each time a loop iterates
- `for` loop is a standard example of this type of quick optimisation

```
// bad
for (i = 0; i < arr.length; i++) {
...
}
// good
l = arr.length;
for (i = 0; i < l; i++) {
...
}
```

- source - W3

JS extras - performance - part 2

DOM access

- repetitive DOM access can be slow, and resource intensive
- try to limit the number of times code needs to access the DOM
- simply access once and then use as a local variable

```
var testDiv = document.getElementById('test');
testDiv.innerHTML = "test...";
```

JavaScript loading

- not always necessary to place JS files in the `<head>` element
  - check context, in particular for recent mobile and desktop frameworks
    - Cordova, Electron…
- adding JS scripts to end of the page's body
  - allows browser to load the page first
- HTTP specification defines browsers should not download more than two components in parallel

---

JS Core - objects - part 1

Objects

- **object** type includes a compound value
  - JS can use to set properties, or named locations
- each of these properties holds its own value
  - can be defined as any type

```
var objectA = {
  a: 49,
  b: 59,
  c: "Philae"
};
```

- access these values using either **dot** or **bracket** notation

```
//dot notation
objectA.a;
//bracket notation
objectA["a"];
```

JS Core - objects - example

```
// create object
var object = {
  archive: 'waldzell',
  access: 'castalia',
  purpose: 'gaming'
};

// log output with dot notation
console.log(`archive is ${object.archive}`);

// log output with bracket notation - returns undefined
console.log(`access is restricted to ${object[1]}`);

// log output with bracket notation
console.log(`purpose is ${object['purpose']}`);
```

Image - JS Object

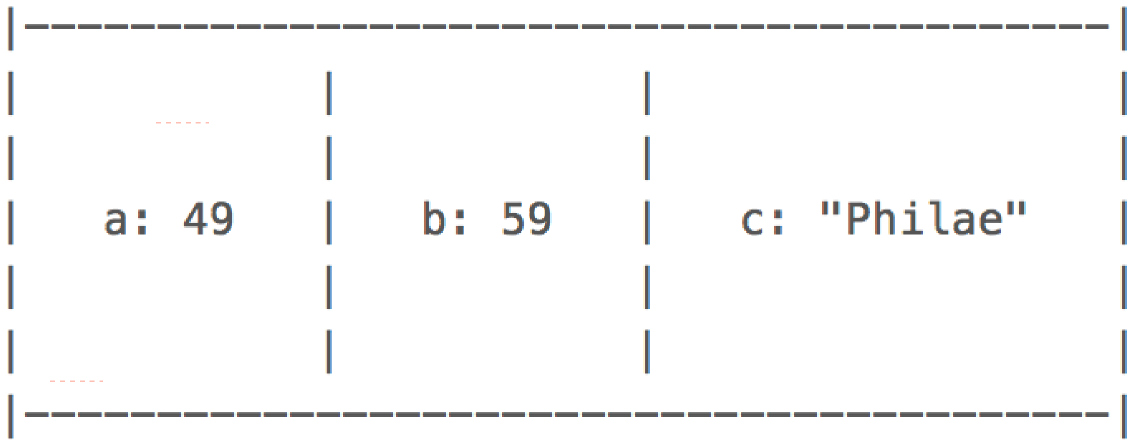


Figure 7: JS Object

ES6 - template literals

```
// create object
var object = {
  archive: 'waldzell',
  access: 'castalia',
  purpose: 'gaming'
};

// log output with template literals
```

```
console.log(`archive is ${object.archive}`);

// log output
console.log('archive is ' + object.archive);

// log output all object properties with template literals
console.log(`archive = ${object.archive}, access = ${object.access},  purpose = ${object.purpose}`);

// log output all object properties
console.log('archive = ' + object.archive + ', access = ' + object.access + ' purpose = ' + object.purp
```

JS Core - objects - part 2

Arrays

- JS array an object that contains values, of any type, in numerically indexed positions
  - store a number, a string...
  - array will start at index position `0`
  - increments by *1* for each new value
- arrays can also have properties
  - e.g. automatically updated **length** property

```
var arrayA = [
  49,
  59,
  "Philae"
];
arrayA.length; //returns 3
```

- each value can be retrieved from its applicable index position

```
arrayA[2]; //returns the string "Philae"
```

Image - JS Array

JS Core - objects - Arrays

examples

- Random Greeting Generator - Basic

CSS Basics - box model - part 1

- consideration of the CSS box model
- a document's attempt to represent each element as a rectangular box
- boxes and properties determined by browser rendering engine
- browser calculates size, properties, and position of these required boxes
- properties can include, for example,
  - colour, background features, borders, width, height...
- box model designed to describe an element's required space and content

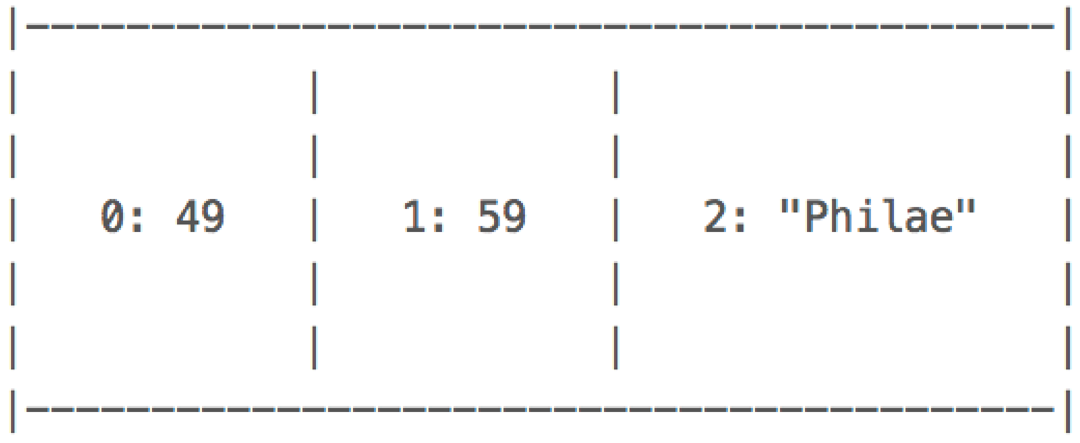


Figure 8: JS Array

- each box has a series of edges,
  - **margin** edge
  - **border** edge
  - **padding** edge
  - **content** edge

CSS Basics - box model - part 2

Content

- box's **content area** describes element's actual content
- properties can include `color` , `background` , `img` ...
  - apply inside the **content** edge
- dimensions include **content width** and **content-height**
- content size properties (assuming that the `box-sizing` property remains default) include,
  - `width` , `min-width` , `max-width` , `height` , `min-height` , `max-height`

Demo - CSS Box Model

- Demo - CSS Box Model

CSS Basics - box model - part 3

Padding

- box's **padding area** includes the extent of the padding to the surrounding border
- background, colour etc properties for a content area extend into the padding
  - we often consider the padding as extending the content
- padding itself is located in the box's **padding edge**
- dimensions are the width and height of the **padding-box**.

- control space between padding and content edge using the following properties,
  - `padding-top` , `padding-right` , `padding-bottom` , `padding-left`
  - `padding` (sizes calculated clock-wise)

Demo - CSS Box Model - Padding

- JSFiddle - CSS Box Model

CSS Basics - box model - part 4

Border

- **border area** extends **padding area** to area containing the borders
- it becomes the area inside the **border edge**
- define its dimensions as the width and height of the **border-box**
- calculated area depends upon the width of the `border` we set in the CSS
- set size of our border using the following properties in CSS,
  - `border-width`
  - `border`

Demo - CSS Box Model - Border

- JSFiddle - CSS Box Model

CSS Basics - box model - part 5

Margin

- **margin area** can extend this border area with an empty area
  - useful to create a defined separation of one element from its neighbours
- dimensions of area defined as width and height of the **margin-box**
- control size of our margin area using the following properties,
  - `margin-top` , `margin-right` , `margin-bottom` , `margin-left`
  - `margin` (sizes calculated clock-wise)

Demo - CSS Box Model - Margin

- JSFiddle - CSS Box Model

Demo - CSS Box Model

- Demo - CSS Box Model

```
margin edge
|----------------------------------------------------------------------------|
|                                                                            |
|    border edge                                                             |
|    |------------------------------------------------------------------|    |
|    |   padding edge                                                   |    |
|    |   |----------------------------------------------------------|   |    |
|    |   |                                                          |   |    |
|    |   |        content edge                                      |   |    |
|    |   |        |----------------------------------------|        |   |    |
|    |   |        |                             ^          |        |   |    |
|    |   |        |                             | height   |        |   |    |
|    |   |        | <---------------------------|--------> |        |   |    |
|    |   |        |     width                   v          |        |   |    |
|    |   |        |----------------------------------------|        |   |    |
|    |   |                                                          |   |    |
|    |   |                                                          |   |    |
|    |   |----------------------------------------------------------|   |    |
|    |                                                                  |    |
|    |------------------------------------------------------------------|    |
|                                                                            |
|                                                                            |
|----------------------------------------------------------------------------|
```

Figure 9: CSS Box Model

Image - CSS Box Model

Source - MDN - CSS Box Model

Demo - CSS Box Model - Interactive

- interactive Box Model

Demos

CSS

- CSS - Complex Selectors Part 1
- CSS - Complex Selectors Part 2
- CSS - Complex Selectors Part 3

Travel Notes - series 1

- travel notes - demo 6
- travel notes - demo 7
- travel notes - demo 8

Travel notes app - series 2 - option 1

- travel notes - demo 1
- travel notes - demo 2
- travel notes - demo 3
- travel notes - demo 4
- travel notes - demo 5
- travel notes - demo 6

Travel notes app - series 2 - options 2 - plain JS

- travel notes - plain JS - demo 3
- travel notes - plain JS - demo 4
- travel notes - plain JS - demo 5

Resources

- CSS
  - CSS Box Model
  - MDN - CSS Box Model
  - CSS Selectors
- JS
  - MDN - JS
  - JS Info - DOM Nodes
    - MDN - JS Objects
    - W3 Schools - JS