

Comp 324/424 - Client-side Web Design

Spring Semester 2020 - Week 7

Dr Nick Hayward

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*

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";  
}
```

JS Core - closures - part 1

- important and useful aspect of JavaScript
- dealing with variables and scope
 - *continued, broader access to ongoing variables via a function's scope*
- closures as a useful construct to allow us to access a function's scope
 - *even after it has finished executing*
- can give us something similar to a private variable
 - *then access through another variable using relative scopes of outer and inner*
- inherent benefit is that we are able to repeatedly access internal variables
 - *normally cease to exist once a function had executed*

JS Core - closures - example - 1

```
//value in global scope
var outerVal = "test1";

//declare function in global scope
function outerFn() {
  //check & output result...
  console.log(outerVal === "test1" ? "test is visible..." : "test not visible...");
}

//execute function
outerFn();
```

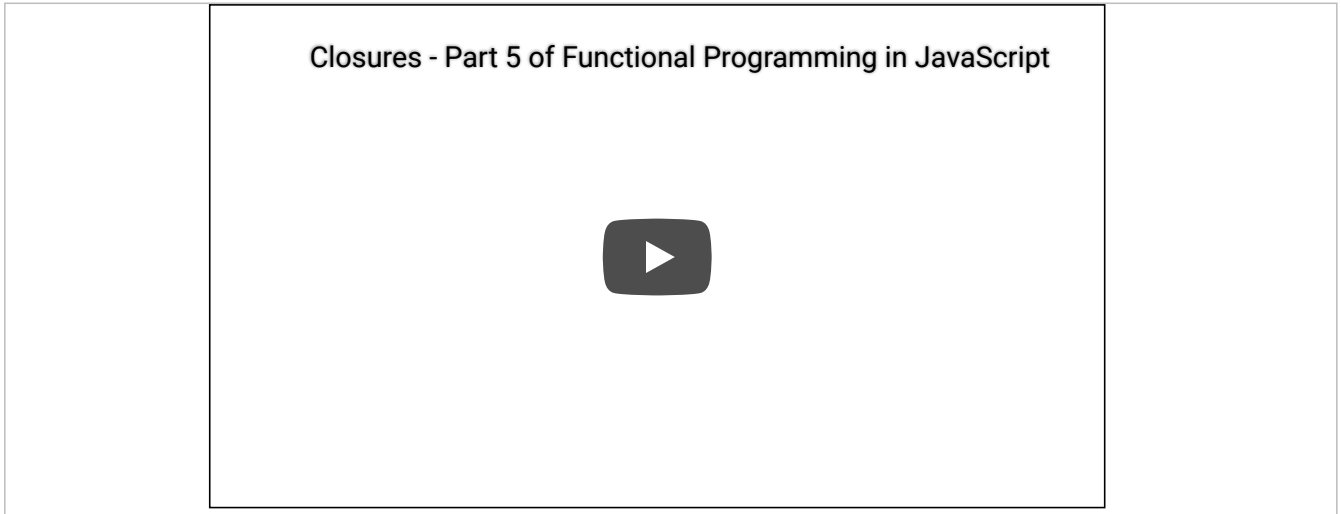
Image - JS Core - closures - global scope

```
test is visible...  
test.js (13,2)
```

JS Core - Closures - global scope

Video - JS Core

closures - part 1



Closures in JavaScript - UP TO 3:17

Source - JavaScript Closures - YouTube

JS Core - closures - example - 2

```
"use strict";

function addTitle(a) {
  var title = "hello ";
  function updateTitle() {
    var newTitle = title+a;
    return newTitle;
  }
  return updateTitle;
}

var buildTitle = addTitle("world");
console.log(buildTitle());
```

JS Core - closures - part 2

Why use closures?

- use closures a lot in JavaScript
 - *real driving force behind Node.js, jQuery, animations...*
- closures help reduce amount, complexity of code necessary for advanced features
- closures help us add otherwise impossible features, e.g.
 - *any task using callbacks - event handlers...*
 - *private object variables...*
- closure allows us to work with a function that has been defined within another scope
 - *still has access to all variables within the defined outer scope*
 - *helps create basic encapsulated data*
 - *store data in a separate scope - then share it where needed*

JS Core - closures - part 3

```
function count(a) {  
  return function(b) {  
    return a + b;  
  }  
}  
  
var add1 = count(1);  
var add5 = count(5);  
var add10 = count(10);  
  
console.log(add1(8));  
console.log(add5(8));  
console.log(add10(8));
```

- using one function to create multiple other functions, add1, add5, add10, and so on.

Video - JS Core

closures - part 2

Closures - Part 5 of Functional Programming in JavaScript



Closures in JavaScript - UP TO 5:21

Source - JavaScript Closures - YouTube

JS Core - closures - example - 3

```
// variables in global scope
var outerVal = "test2";
var laterVal;

function outerFn() {
  // inner scope variable declared with value - scope limited to function
  var innerVal = "test2inner";
  // inner function - can access scope from parent function & variable innerVal
  function innerFn() {
    console.log(outerVal === "test2" ? "test2 is visible" : "test2 not visible");
    console.log(innerVal === "test2inner" ? "test2inner is visible" : "test2inner is not visible");
  }
  // inner function now added to global scope - now able to access elsewhere & call later
  laterVal = innerFn;
}
// invokes outerFn, innerFn is created, and its reference assigned to laterVal
outerFn();
// THEN - innerFn is invoked using laterVal - can't access innerFn directly...
laterVal();
```

Image - JS Core - closures - inner scope

```
test2 is visible  
test.js (15,5)  
test2inner is visible  
test.js (16,5)
```

JS Core - Closures - inner scope

JS Core - closures - part 4

- how is the `innerVal` variable available when we execute the inner function?
 - *this is why closures are such an important and useful concept in JavaScript*
 - *use of closures creates a sense of persistence in the scope*
- closures help create
 - *scope persistence*
 - *delayed access to functions and variables*
- closure creates a safe wrapper around
 - *the function*
 - *variables that are in scope as a function is defined*
- closure ensures function has everything necessary for correct execution
- closure wrapper persists whilst function exists

n.b. closure usage is not memory free - there is an impact on app memory and usage...

Video - JS Core

closures - part 3

Closures - Part 5 of Functional Programming in JavaScript



Closures in JavaScript - UP TO 6:20

Source - JavaScript Closures - YouTube

JS core - this

- this keyword - correct and appropriate usage
 - *commonly misunderstood feature of JS*
- value of this is not inherently linked with the function itself
- value of this determined in response to how the function is called
- value itself can be dynamic, simply based upon how the function is called
- if a function contains this, its reference will usually point to an object

JS core - this - part 1

global, window object

- when we call a function, we can bind the `this` value to the window object
- resultant object refers to the root, in essence the global scope

```
function test1() {  
  console.log(this);  
}  
  
test1();
```

- NB: the above will return a value of undefined in strict mode.
- also check for the value of `this` relative to the global object,

```
var a = 49;  
  
function test1() {  
  console.log(this.a);  
}  
  
test1();
```

- JSFiddle - this - window
- JSFiddle - this - global

JS core - this - part 2

object literals

- within an object literal, the value of `this`, thankfully, will always refer to its own object

```
var object1 = {  
  method: test1  
};  
  
function test1() {  
  console.log(this);  
}  
  
object1.method();
```

- return value for `this` will be the object itself
- we get the returned object with a property and value for the defined function
- other object properties and values will be returned and available as well
- JSFiddle - this - literal
- JSFiddle - this - literal 2

JS core - this - part 3

object literals

```
var sites = {};  
sites.name = "philae";  
  
sites.titleOutput = function() {  
  console.log("Egyptian temples...");  
};  
  
sites.objectOutput = function() {  
  console.log(this);  
};  
  
console.log(sites.name);  
sites.objectOutput();  
sites.titleOutput();
```

Image - Object literals console output

```
philae  
test.js (22,1)  
▸ [object Object]      {name: "philae"}  
  test.js (19,3)  
Egyptian temples...  
test.js (15,3)
```

JS - this - object literals output

JS core - this - part 4

events

- for events, value of `this` points to the owner of the bound event

```
<div id="test">click to test...</div>
```

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

function output() {
  console.log(this);
};

testDiv.addEventListener('click', output, false);
```

- element is clicked, value of `this` becomes the clicked element
- also change the context of `this` using built-in JS functions
 - *such as* `.apply()`, `.bind()`, and `.call()`
- JSFiddle - this - events

ES6 JS - Arrow functions

basic

```
/**  
  js-plain - definitions and arguments  
  - basic example for arrow function  
**/  
  
// define array for planets  
planets = ['mars', 'jupiter', 'venus'];  
// use for each loop with array, and create arrow function for output to console  
planets.forEach(planet => console.log(planet));
```

■ Demo

ES6 JS - Arrow functions

function context

```
/**
  js-plain - definitions and arguments
  - example of arrow function with function context
  **/

// button constructor
function Button() {
  this.clicked = false;
  // arrow function in function context
  this.click = () => {
    this.clicked = true;
    var message = `button clicked - ${this.clicked}`;
    console.log(message);
    document.getElementById("output").append(message);
  };
}

// create button object
var button = new Button();
var element = document.getElementById("test");
element.addEventListener("click", button.click);
```

■ Demo

ES6 JS - Arrow functions

example

- Random Greeting Generator - A bit better - v0.2

JS - Closures - *private* object property

A brief demo of getters and setters with private object property.

- FN: constructor function
 - *'private variable' - not directly accessible*
 - *define properties on object*
 - add *getter* and *setter* methods
- Use:
 - *instantiate object using constructor*
 - *log output of check against getter method for value of 'private' variable*
 - *use 'setter' method to update value of 'private' variable*
 - *log output for check of value update of 'private' variable*

JS - closures - *private* object property - example

```
// define constructor
function Archive() {
  // private variable - accessible through function closures
  let _catalogue = 'glass bead';
  // define catalogue property access
  Object.defineProperty(this, 'catalogue', {
    get: () => {
      console.log(`catalogue requested...`);
      return _catalogue;
    },
    set: value => {
      console.log(`catalogue updated`);
      _catalogue = value;
    }
  });
}

// instantiate object from Archive constructor
const archiveCheck = new Archive();

// check access to constructor variable - returns 'undefined' without getter method
console.log(`direct access against private variable = ${archiveCheck._catalogue}`);
// check access using getter method - returns variable value
console.log(`getter access against private variable = ${archiveCheck.catalogue}`);

// update catalogue value - uses 'setter' method
archiveCheck.catalogue = 'history';

// check update catalogue variable
console.log(`updated catalogue = ${archiveCheck.catalogue}`);
```

■ Demo - private object property

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 - initial usage

fun exercise

Choose one of the following app examples,

- sports website for latest scores and updates
 - *e.g. scores for current matches, statistics, team data, player info &c.*
- shopping website
 - *product listings and adverts, cart, reviews, user account page &c.*
- restaurant website
 - *introductory info, menus, sample food images, user reviews &c.*

Then, consider the following

- where do you need JavaScript in the app?
 - *why?*

JS extras - JSON - part 1

- JSON is a lightweight format and wrapper for storing and transporting data
- inherently language agnostic, easy to read and understand
- growing rapidly in popularity
 - *many online APIs have updated XML to JSON for data exchange*
- syntax of JSON is itself derived from JS object notation
 - *text-only format*
- allows us to easily write, describe, and manipulate JSON in practically any programming language
- JSON syntax follows a few basic rules,
 - *data is recorded as name/value pairs*
 - *data is separated by commas*
 - *objects are defined by a start and end curly brace*
 - {}
 - *arrays are defined by a start and end square bracket*
 - []

JS extras - JSON - part 2

- underlying construct for JSON is a pairing of name and value

```
"city": "Marseille"
```

JSON Objects

- contained within curly braces
- objects can contain multiple name/value pairs

```
{  
  "country": "France",  
  "city": "Marseille"  
}
```

JS extras - JSON - part 3

JSON Arrays

- contained within square brackets
 - *arrays can also contain objects*

```
{  
  "cities": [  
    {  
      "name": "Marseille",  
      "region": "Provence-Alpes-Côte d'Azur"  
    },  
    {  
      "name": "Paris",  
      "region": "Île-de-France"  
    }  
  ]  
}
```

- use this with JavaScript, and parse the JSON object.
 - *JSFiddle - Parse JSON*

HTML5, CSS, & JS - example - part 1

Structure

- combine HTML5, CSS, and JavaScript, to create an example application
- outline of our project's basic directory structure

```
.
|- assets
|  |- images //logos, site/app banners - useful images for site's design
|  |- scripts //js files
|  |- styles //css files
|- docs
|  |- json //any .json files
|  |- txt //any .txt files
|  |- xml //any .xml files
|- media
|  |- audio //local audio files for embedding & streaming
|  |- images //site images, photos
|  |- video //local video files for embedding & streaming
|- index.html
```

- each of the above directories can, of course, contain many additional sub-directories
 - */- images may contain sub-directories for albums, galleries...*
 - */- xml may contain sub-directories for further categorisation..*
 - *and so on...*

HTML5, CSS, & JS - example - part 2

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>travel notes - v0.1</title>
    <meta name="description" content="information on travel destinations">
    <meta name="author" content="ancientlives">
    <!-- css styles... -->
    <link rel="stylesheet" type="text/css" href="assets/styles/style.css">
  </head>
  <body>
    ...
    <!-- js scripts... -->
    <script type="text/javascript" src="assets/scripts/jquery.min.js"></script>
    <script type="text/javascript" src="assets/scripts/travel.js"></script>
  </body>
</html>
```

- JS files at foot of body
 - *hierarchical rendering of page by browser - top to bottom*
 - *JS will now be one of the last things to load*
 - *JS files often large, slow to load*
 - *helps page load faster...*

HTML5, CSS, & JS - example - part 3

index.html - body

```
<body>
  <!-- document header -->
  <header>
    <h3>travel notes</h3>
    <p>record notes from various cities and placed visited...</p>
  </header>
  <!-- document main -->
  <main>
    <!-- note input -->
    <section class="note-input">
    </section>
    <!-- note output -->
    <section class="note-output">
    </section>
  </main>
  <!-- document footer -->
  <footer>
    <p>app's copyright information, additional links...</p>
  </footer>
  <!-- js scripts... -->
  <script type="text/javascript" src="assets/scripts/jquery.min.js"></script>
  <script type="text/javascript" src="assets/scripts/travel.js"></script>
</body>
```

HTML5, CSS, & JS - example - part 4

style.css

```
body {  
    width: 850px;  
    margin: auto;  
    background: #fff;  
    font-size: 16px;  
    font-family: "Times New Roman", Georgia, Serif;  
}  
h3 {  
    font-size: 1.75em;  
}  
header {  
    border-bottom: 1px solid #dedede;  
}  
header p {  
    font-size: 1.25em;  
    font-style: italic;  
}  
footer p {  
    font-size: 0.8em;  
}
```


HTML5, CSS, & JS - example - part 5.1

travel.js

```
//overall app logic and loader...  
function travelNotes() {  
    "use strict";  
  
    $(".note-output").html("<p>first travel note for Marseille...</p>");  
  
};  
  
$(document).ready(travelNotes);
```

- a simple JS function to hold the basic logic for our app
- call this function any reasonable, logical name
- in initial function, we set the strict pragma
- add an example call to the jQuery function, `html()`
 - *sets some initial note content*
- function `travelNotes()` loaded using the jQuery function `ready()`
 - *many different ways to achieve this basic loading of app logic*

HTML5, CSS, & JS - example - part 5.2

travel.js - plain JS

```
function travelNotes() {  
  "use strict";  
  
  // get a reference to `.note_output` in the DOM  
  // n.b. these can be combined as well...  
  let noteOutput = document.querySelector('.note-output');  
  noteOutput.innerHTML = '<p>first travel note for Marseille...</p>';  
  
}  
  
// Load app  
travelNotes();
```

- DEMO 1 - travel notes - series 1

HTML5, CSS, & JS - example - part 6

add a note

- app's structure includes three clear semantic divisions of content
 - `<header>`, `<main>`, and `<footer>`
- `<main>` content category - create and add our notes for our application
- allow a user to create a new note
 - *enter some brief text, and then set it as a note*
- output will simply resemble a heading or brief description for our note
- add HTML element `<input>` to allow a user to enter note text
 - *new attributes in HTML5 such as `autocomplete`, `autofocus`, `required`, `width`...*
 - *set accompanying*

```
<h5>add note</h5>  
<input>
```

```
<input type="text" value="add a note...">
```

HTML5, CSS, & JS - example - part 7

tidy up styling

- additional styles to create correct, logical separation of visual elements and content
- add a border to the top of our footer
 - *perhaps matching the header in style*
- update the box model for the <main> element
- add some styling for <h5> heading

```
h5 {  
  font-size: 1.25em;  
  margin: 10px 0 10px 0;  
}  
main {  
  overflow: auto;  
  padding: 15px 0 15px 0;  
}  
footer {  
  margin-top: 5px;  
  border-top: 1px solid #dedede;  
}
```

HTML5, CSS, & JS - example - part 8

input update

```
<input><button>add</button>
```

```
.note-input input {  
  width: 40%;  
}  
.note-input button {  
  padding: 2px;  
  margin-left: 5px;  
  border-radius: 0;  
  border: 1px solid #dedede;  
  cursor: pointer;  
}
```

- also update css for input and button
- remove button's rounded borders to match style of input
- match border for button to basic design aesthetics
- set cursor appropriate for a link style...
- DEMO 2 - travel notes - series 1

HTML5, CSS, & JS - example - part 9.1

interaction - add a note

- added and styled our input and button for adding a note
- use jQuery to handle click event on button
- update `travel.js` file for event handler

```
//handle user event for `add` button click
$(".note-input button").on("click", function(e) {
  console.log("add button clicked...");
});
```

HTML5, CSS, & JS - example - part 9.2

interaction - add a note - plain JS

```
let addNoteBtn = document.getElementById('add-note');
addNoteBtn.addEventListener('click', () => {
  console.log('add button clicked...');
});
```

HTML5, CSS, & JS - example - part 10.1

interaction - add a note - output

- update this jQuery code to better handle and output the text from the input field
- what is this handler actually doing?
 - *jQuery code has attached an event listener to an element in the DOM*
 - *referenced in the selector option at the start of the function*
 - *uses standard CSS selectors to find the required element*
- jQuery can select and target DOM elements using standard CSS selectors
 - *then manipulate them, as required, using JavaScript*

```
//handle user event for `add` button click
$(".note-input button").on("click", function(e) {
    $(".note-output").append("<p>sample note text...</p>");
});
```

- output some static text to note-output

HTML5, CSS, & JS - example - part 10.2

interaction - add a note - output - plain JS

```
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');  
  
  // add event listener to add note button  
  addNoteBtn.addEventListener('click', () => {  
    // create p node  
    let p = document.createElement('p');  
    // create text node  
    let noteText = document.createTextNode('sample note text...');  
    // append text to paragraph  
    p.appendChild(noteText);  
    // append new paragraph and text to existing note output  
    noteOutput.appendChild(p);  
  });  
}
```

- DEMO 3 - travel notes - series 1

HTML5, CSS, & JS - example - part 11.1

interaction - add a note - output

```
//overall app Logic and loader...
function travelNotes() {
    "use strict";

    //handle user event for `add` button click
    $(".note-input button").on("click", function(e) {
        //object for wrapper html for note
        var $note = $("

");
        //get value from input field
        var note_text = $(".note-input input").val();
        //set content for note
        $note.html(note_text);
        //append note text to note-output
        $(".note-output").append($note);
    });
};

$(document).ready(travelNotes);


```

HTML5, CSS, & JS - example - part 11.2

interaction - add a note - output - plain JS

```
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');

  addNoteBtn.addEventListener('click', () => {
    // create p node
    let p = document.createElement('p');
    // get value from input field for note
    let inputVal = inputNote.value;
    // create text node
    let noteText = document.createTextNode(inputVal);
    // append text to paragraph
    p.appendChild(noteText);
    // append new paragraph and text to existing note output
    noteOutput.appendChild(p);
  });
}
```

- DEMO 4 - travel notes - series 1

HTML5, CSS, & JS - example - part 12.1

interaction - add a note - clear input

```
//overall app logic and loader...
function travelNotes() {
    "use strict";

    //handle user event for `add` button click
    $(".note-input button").on("click", function(e) {
        //object for wrapper html for note
        var $note = $("
```

HTML5, CSS, & JS - example - part 12.2

interaction - add a note - clear input - plain JS

```
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', () => {  
    // create p node  
    let p = document.createElement('p');  
    // get value from input field for note  
    let inputVal = inputNote.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  
      noteOutput.appendChild(p);  
      // clear input text field  
      inputNote.value = '';  
    }  
  });  
}
```

■ DEMO 5 - travel notes - series 1

HTML5, CSS, & JS - example - part 13.1

interaction - add a note - keyboard listener

- need to consider how to handle keyboard events
- listening and responding to a user hitting the return key in the input field
- similar pattern to user click on button

```
$(".note-input input").on("keypress", function (e) {  
  if (e.keyCode === 13) {  
    ...do something...  
  }  
});
```

- need to abstract handling both button click and keyboard press
- need to be selective with regard to keys pressed
- add a conditional check to our listener for a specific key
- use local variable from the event itself, eg: e, to get value of key pressed
- compare value of e against key value required

HTML5, CSS, & JS - example - part 13.2

interaction - add a note - keyboard listener - plain JS

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

- example recording keypresses
 - *Demo Editor*

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*

```
//manage input field and new note output
function createNote() {
  //object for wrapper html for note
  var $note = $("<p>");
  //define input field
  var $note_text = $(".note-input input");
  //conditional check for input field
  if ($note_text.val() !== "") {
    //set content for note
    $note.html($note_text.val());
    //append note text to note-output
    $(".note-output").append($note);
    $note_text.val("");
  }
}
```


HTML5, CSS, & JS - example - part 15.1

interaction - add a note - travel.js

```
//overall app logic and loader...
function travelNotes() {
  "use strict";

  //manage input field and new note output
  function createNote() {
    //object for wrapper html for note
    var $note = $("

");
    //define input field
    var $note_text = $(".note-input input");
    //conditional check for input field
    if ($note_text.val() !== "") {
      //set content for note
      $note.html($note_text.val());
      //append note text to note-output
      $(".note-output").append($note);
      $note_text.val("");
    }
  }

  //handle user event for `add` button click
  $(".note-input button").on("click", function(e) {
    createNote();
  });

  //handle user event for keyboard press
  $(".note-input input").on("keypress", function(e){
    if (e.keyCode === 13) {
      createNote();
    }
  });
};

$(document).ready(travelNotes);


```

HTML5, CSS, & JS - example - part 15.2

interaction - add a note - plain JS

```
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);
    }
  });
}

// 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 = '';  
    }  
}  
  
// Load app  
travelNotes();
```

- DEMO 6 - travel notes - series 1

HTML5, CSS, & JS - example - part 16

interaction - add a note - animate

- jQuery well-known for its simple ability to animate elements
- many built-in effects available in jQuery
 - *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
- many additional parameters for jQuery's fadeIn function
 - *customise a callback*
 - *change the speed of the animation*
 - *and so on...*
- jQuery API - fadeIn

HTML5, CSS, & JS - example - part 17

interaction - add a note - animate js

```
//manage input field and new note output
function createNote() {
  //object for wrapper html for note
  var $note = $("<p>");
  //define input field
  var $note_text = $(".note-input input");
  //conditional check for input field
  if ($note_text.val() !== "") {
    //set content for note
    $note.html($note_text.val());
    //hide new note to setup fadeIn...
    $note.hide();
    //append note text to note-output
    $(".note-output").append($note);
    //fadeIn hidden new note
    $note.fadeIn("slow");
    $note_text.val("");
  }
}
```

- DEMO 7 - travel notes - series 1

HTML5, CSS, & JS - example - part 18

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 8 - 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

Demos

- ES6 (ES2015)
- let usage - Random Greeting Generator v0.2
- JS Arrays
 - *Random Greeting Generator - v0.1*
- JSFiddle
 - *Basic logic - functions*
 - *Basic logic - scope*
 - *this - events*
 - *this - global*
 - *this - literal*
 - *this - literal 2*
 - *this - window*
 - *Parse JSON*
- Travel notes app - series 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 - demo 7*
 - *travel notes - demo 8*

Resources

- JavaScript Closures - YouTube
- jQuery
 - *jQuery*
 - *jQuery API*
 - *jQuery - deferred*
 - *jQuery - .getJSON()*
 - *jQuery - JSONP*
 - *jQuery - promise*
- MDN
 - *MDN - JS*
 - *MDN - JS Const*
 - *MDN - JS - Iterators and Generators*
 - *MDN - JS Objects*