Comp 324/424 - Client-side Web Design

Fall Semester 2019 - Week 11

Dr Nick Hayward

JavaScript extras - API examples

further API examples

- Google Maps & Places

- Google Distance Matrix

- Google Maps markers & resizing

- Twitter
  - *user queries &c.*
  - *OAuth based login and authentication*

- Yelp
  - *custom server and remote API query*
  - *sample handling of local API for queries*

- YouTube
  - *custom query*
  - *custom with account authentication*
  - *search and playback*

- ...

ES6 Generators & Promises - intro

- generators and promises are new to plain JavaScript
  - *introduced with ES6 (ES2015)*

- **Generators** are a special type of function
  - *produce multiple values per request*
  - *suspend execution between these requests*

- *generators* are useful to help simplify convoluted loops

- suspend and resume code execution, &c.
  - *helps write simple, elegant async code*

- **Promises** are a new, built-in object
  - *help development of async code*

- promise becomes a placeholder for a value not currently available
  - *but one that will be available later*

ES6 Generators & Promises - async code and execution

- JS relies on a single-threaded execution model

- query a remote server using standard code execution
  - *block the UI until a response is received and various operations completed*

- we may modify our code to use callbacks
  - *invoked as a task completes*
  - *should help resolve blocking the UI*

- callbacks can quickly create a *spaghetti* mess of code, error handling, logic...

- *Generators* and *Promises*
  - *elegant solution to this mess and proliferation of code*

ES6 Generators & Promises - promises - intro

- a *promise* is similar to a placeholder for a value we currently do not have
  - *but we would like later...*

- it's a guarantee of sorts
  - *eventually receive a result to an asynchronous request, computation, &c.*

- a result will be returned
  - *either a value or an error*

- we commonly use *promises* to fetch data from a server
  - *fetch local and remote data*
  - *fetch data from APIs*

ES6 Generators & Promises - promises - example

```javascript
// use built-in Promise constructor - pass callback function with two parameters
const testPromise = new Promise((resolve, reject) => {
  resolve("test return");
  // reject("an error has occurred trying to resolve this promise...");
});


// use `then` method on promise - pass two callbacks for success and failure
testPromise.then(data => {
  // output value for promise success
  console.log("promise value = "+data);
}, err => {
  // output message for promise failure
  console.log("an error has been encountered...");
});
```

- use the built-in *Promise* constructor to create a new promise object

- then pass a function
  - *a standard arrow function in the above example*

ES6 Generators & Promises - promises - executor

- function for a Promise is commonly known as an *executor* function
  - *includes two parameters, `resolve` and `reject`*

- *executor* function is called immediately
  - *as the Promise object is being constructed*

- `resolve` argument is called manually
  - *when we need the promise to resolve successfully*

- second argument, `reject`, will be called if an error occurs

- uses the *promise* by calling the built-in `then` method
  - *available on the promise object*

- `then` method accepts two callback functions
  - *`success` and `failure`*

- `success` is called if the *promise* resolves successfully

- the `failure` callback is available if there is an error

ES6 Generators & Promises - promises - example

explicit use of `resolve`

```js
/*
 * promise1.js
 * wrap Array in Promise using resolve()...
 */

let testArray = Promise.resolve(['one', 'two', 'three']);

testArray.then(value => {
console.log(value[0]);
// remove first item from array
value.shift();
// pass value to chained `then`
return value;
})
.then(value => console.log(value[0]));
```

- Demo - Promise.resolve

ES6 Generators & Promises - promises - callbacks & async

- async code is useful to prevent execution blocking
  - *potential delays in the browser*
  - *e.g. as we execute long-running tasks*

- issue is often solved using *callbacks*
  - *i.e. provide a callback that's invoked when the task is completed*

- such long running tasks may result in errors

- issue with callbacks
  - *e.g. we can't use built-in constructs such as* `try-catch` *statements*

ES6 Generators & Promises - promises - callbacks & async - example

```
try {
  getJSON("data.json", function() {
    // handle return results...
  });
} catch (e) {
  // handle errors...
}
```

- this won't work as expected due to the code executing the callback
  - *not usually executed in the same step of the event loop*
  - *may not be in sync with the code running the long task*

- errors will usually get lost as part of this long running task

- another issue with callbacks is nesting

- a third issue is trying to run parallel callbacks

- performing a number of parallel steps becomes inherently tricky and error prone

ES6 Generators & Promises - promises - further details

- a *promise* starts in a pending state
  - *we know nothing about the return value*
  - *promise is often known as an unresolved promise*

- during execution
  - *if the promise's resolve function is called*
  - *the promise will move into its fulfilled state*
  - *the return value is now available*

- if there is an error or *reject* method is explicitly called
  - *the promise will simply move into a rejected state*
  - *return value is no longer available*
  - *an error now becomes available*

- either of these states
  - *the promise can now no longer switch state*
  - *i.e from rejected to fulfilled and vice-versa...*

ES6 Generators & Promises - promises - concept example

an example of working with a promise may be as follows

- code starts (execution is ready)
- promise is now executed and starts to run
- promise object is created
- promise continues until it resolves
  - *successful return, artificial timeout &c.*
- code for the current promise is now at an end
- promise is now resolved
  - *value is available in the promise*
- then work with resolved promise and value
  - *call `then` method on promise and returned value...*
  - *this callback is scheduled for successful resolve of the promise*
  - *this callback will always be asynchronous regardless of state of promise...*

ES6 Generators & Promises - promises - callbacks & async - example

promise from scratch

```js
/*
 * promisefromscratch-delay.js
 * create a Promise object from scratch...use delay to check usage
 * promise may only be called once per execution due to delay and timeout...
 */


// check promise usage relative to timer...either timeout will cause the Promise
function resolveWithDelay(delay) {
  return new Promise(function(resolve, reject) {
    // log Promise creation...
    console.log('promise created...waiting');
        // resolve promise if delay value is less than 3000
    setTimeout(function() {
      resolve(`promise resolved in ${delay} ms`);
    }, delay);
        // resolve promise if delay is greater than 3000
    setTimeout(function() {
      resolve(`promise resolved in 3000ms`);
    }, 3000);
  })
}

// fulfilled with delay of 2000 ms
resolveWithDelay(2000).then(function(value) {
  console.log(value);
});
// fulfilled with default timeout of 3000 ms
// resolveWithDelay(6000).then(function(value) {
//   console.log(value);
// });
```

- Demo - Promise from scratch

ES6 Generators & Promises - promises - explicitly reject

- two standard ways to reject a promise

- e.g. explicit rejection of promise

```
const promise = new Promise((resolve, reject) => {
    reject("explicit rejection of promise");
});
```

- once the promise has been rejected
  - *an error callback will always be invoked*
  - *e.g. through the calling of the* `then` *method*

```
promise.then(
  () => fail("won't be called..."),
  error => pass("promise was explicitly rejected...");
);
```

- also chain a `catch` method to the `then` method

- as an alternative to the error callback. e.g.

```
promise.then(
  () => fail("won't be called..."))
  .catch(error => pass("promise was explicitly rejected..."));
```

ES6 Generators & Promises - promises - example

promise error handling

```
/*
 * promise-basic-error1.js
 * basic example usage of promise error handling and order...
 */

Promise
  .resolve(1)
  .then(x => {
    if (x === 2) {
      console.log('val resolved as', x);
    } else {
      throw new Error('test failed...')
    }
  })
  .catch(err => console.error(err));
```

- Demo - Promise error handling with catch

ES6 Generators & Promises - promises - real-world promise - getJSON

```javascript
// create a custom get json function
function getJSON(url) {
  // create and return a new promise
  return new Promise((resolve, reject) => {
    // create the required XMLHttpRequest object
    const request = new XMLHttpRequest();
    // initialise this new request - open
    request.open("GET", url);
    // register onload handler - called if server responds
    request.onload = function() {
      try {
        // make sure response is OK - server needs to return status 200 code...
        if (this.status === 200) {
          // try to parse json string - if success, resolve promise successfully
          resolve(JSON.parse(this.response));
        } else {
          // different status code, exception parsing JSON &c. - reject the promi
          reject(this.status + " " + this.statusText);
        }
      } catch(e) {
        reject(e.message);
      }
    };

    // if error with server communication - reject the promise...
    request.onerror = function() {
      reject(this.status + " " + this.statusText);
    };

    // send the constructed request to get the JSON
    request.send();
  });
}
```

ES6 Generators & Promises - promises - real-world promise - usage

```javascript
// call getJSON with required URL, then method for resolve object, and catch for
getJSON("test.json").then(response => {
  // check return value from promise...
  response !== null ? "response obtained" : "no response";
}).catch((err) => {
  // Handle any error that occurred in any of the previous promises in the chain.
    console.log('error found = ', err); // not much to show due to return of json
});
```

ES6 Generators & Promises - promises - chain

- calling `then` on the returned promise creates a new *promise*

- if this promise is now resolved successfully
  - *we can then register an additional callback*

- we may now chain as many `then` methods as necessary

- create a sequence of promises
  - *each resolved &c. one after another*

- instead of creating deeply nested callbacks
  - *simply chain such methods to our initial resolved promise*

- to catch an error we may chain a final `catch` call

- to catch an error for the overall chain
  - *use the `catch` method for the overall chain*

```
getJSON().then()
.then()
.then()
.catch((err) => {
  // Handle any error that occurred in any of the previous promises in the chain.
    console.log('error found = ', err); // not much to show due to return of json
});
```

- if a failure occurs in any of the previous promises
  - *the `catch` method will be called*

ES6 Generators & Promises - promises - wait for multiple promises

- promises also make it easy to wait for multiple, independent asynchronous tasks

- with `Promise.all`, we may wait for a number of promises

```javascript
// wait for a number of promises - all
Promise.all([
// call getJSON with required URL, `then` method for resolve object, and `catch`
getJSON("notes.json"),
getJSON("metadata.json")]).then(response => {
  // check return value from promise...response[0] = notes.json, response[1] = me
  if (response[0] !== null) {
       console.log("response obtained");
    console.log("notes = ", JSON.stringify(response[0]));
    console.log("metadata = ", JSON.stringify(response[1]));
    }
}).catch((err) => {
  // Handle any error that occurred in any of the previous promises in the chain.
    console.log('error found = ', err); // not much to show due to return of json
});
```

- order of execution for tasks doesn't matter for `Promise.all`

- by using the `Promise.all` method
  - *we are simply stating that we want to wait...*

- `Promise.all` accepts an array of promises
  - *then creates a new promise*
  - *promise will resolve successfully when all passed promises resolve*

- it will reject if a single one of the passed promises fails

- return promise is an array of succeed values as responses
  - *i.e. one succeed value for each passed in promise*

ES6 Generators & Promises - promises - racing promises

- we may also setup competing promises
  - *with an effective prize to the first promise to resolve or reject*
  - *might be useful for querying multiple APIs, databases, &c.*

```
Promise.race(
  [
  // call getJSON with required URL, `then` method for resolve object, and `catch
  getJSON("notes.json"),
  getJSON("metadata.json")]).then(response => {
    if (response !== null) {
        console.log(`response obtained - ${response} won...`);
    }
  }).catch((err) => {
    // Handle any error that occurred in any of the previous promises in the chai
    console.log('error found = ', err); // not much to show due to return of json
  });
);
```

- method accepts an array of promises
  - *returns a completely new resolved or rejected promise*
  - *returns for the first resolved or rejected promise*

ES6 Generators & Promises - promises - Fetch API

- MDN - Fetch API

ES6 Generators & Promises - promises - Fetch API - Example

basic usage

```javascript
/*
 * fetch-basic1.js
 * basic example usage of Fetch API...
 */

fetch('./assets/notes.json')
  .then(response => {
    return response.json();
  })
  .then(myJSON => {
    console.log(myJSON);
  });
```

- Demo - Fetch API - basic usage

ES6 Generators & Promises - promises - Fetch API - Example

catching errors

```javascript
/*
 * fetch-basic-error1.js
 * basic example usage of Fetch API...chain `catch` to `then` for error handling
 */

fetch('./assets/item.json')
  .then(response => {
    // reactions passed to `then` used to handle fulfillment of a promise
    return response.json();
  })
  .then(myJSON => {
    console.log(myJSON);
  })
  .catch(err => {
    // reactions passed to `catch` executed with a rejection reason...
    console.log(`error detected - ${err}`);
  });
```

- Demo - Fetch API - catching errors

ES6 Generators & Promises - promises - Fetch API - Example

Fetch with Promise all

```
/*
 * fetch-promise-all.js
 * basic example usage of Promise.all...using Fetch API
 */

Promise
  .all([
    fetch('./assets/items.json'),
    fetch('./assets/notes.json')
  ])
  .then(responses =>
    Promise.all(responses.map(res => res.json()))
  ).then (json => {
    console.log(json);
  });
```

- Demo - Fetch API - Promise all

ES6 Generators & Promises - promises - Fetch API - Example

Fetch with Promise race

```
/*
 * fetch-promise-race.js
 * basic example usage of Promise.race...using Fetch API
 */

Promise
  .race([
    fetch('./assets/items.json'),
    fetch('./assets/notes.json')
  ])
  .then(responses => {
    return responses.json()
  })
  .then(res => console.log(res));
```

- Demo - Fetch API - Promise race

ES6 Generators & Promises - generators

- a *generator* function generates a sequence of values
  - *commonly not all at once but on a request basis*

- generator is explicitly asked for a new value
  - *returns either a value or a response of no more values*

- after producing a requested value
  - *a generator will then suspend instead of ending its execution*
  - *generator will then resume when a new value is requested*

ES6 Generators & Promises - generators - example

```javascript
//generator function
function* nameGenerator() {
  yield "emma";
  yield "daisy";
  yield "rosemary";
}
```

- define a generator function by appending an *asterisk* after the keyword
  - *function* *()*

- use the `yield` keyword within the body of the generator
  - *to request and retrieve individual values*

- then consume these generated values using a standard loop
  - *or perhaps the new* `for-of` *loop*

ES6 Generators & Promises - generators - iterator object

- if we make a call to the body of the generator
  - *an iterator object will be created*

- we may now communicate with and control the generator using the iterator object

```
//generator function
function* NameGenerator() {
  yield "emma";
}
// create an iterator object
const nameIterator = NameGenerator();
```

- iterator object, `nameIterator`, exposes various methods including the `next` method

ES6 Generators & Promises - generators - iterator object - next()

- use `next` to control the iterator, and request its next value

```
// get a new value from the generator with the 'next' method
const name1 = nameIterator.next();
```

- `next` method executes the generator's code to the next yield expression

- it then returns an object with the value of the yield expression
  - *and a property done set to false if a value is still available*

- `done` boolean will switch to *true* if no value for next requested yield

- `done` is set to *true*
  - *the iterator for the generator has now finished*

ES6 Generators & Promises - generators - iterate over iterator object

- iterate over the iterator object
  - *return each value per available yield expression*
  - *e.g. use the `for-of` loop*

```javascript
// iterate over iterator object
for(let iteratorItem of NameGenerator()) {
  if (iteratorItem !== null) {
    console.log("iterator item = "+iteratorItem+index);
  }
}
```

ES6 Generators & Promises - generators - call generator within a generator

- we may also call a generator from within another generator

```
//generator function
function* NameGenerator() {
  yield "emma";
  yield "rose";
  yield "celine";
  yield* UsernameGenerator();
  yield "yvaine";
}

function* UsernameGenerator() {
  yield "frisby67";
  yield "trilby72";
}
```

- we may then use the initial generator, `NameGenerator`, as normal

ES6 Generators & Promises - generators

example - pass generator to function

```javascript
function getRandomNote(gen) {
  console.log(`getRandomNote called...`);
  const g = gen();
  fetch('./assets/input/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json())
  .then(json => {
      return g.next(json);
  })
  .catch(err => g.throw(err))
}

getRandomNote(function* printRandomNote() {
  console.log(`generator function executes...`);
  const json = yield;

})
```

- Demo - Generators - pass generator to function

ES6 Generators & Promises - generator - recursive traversal of DOM

- document object model, or DOM, is tree-like structure of HTML nodes

- every node, except the root, has exactly one parent
  - *and the potential for zero or more child nodes*

- we may now use generators to help iterate over the DOM tree

```javascript
// generator function - traverse the DOM
function* DomTraverseGenerator(htmlElem) {
  yield htmlElem;
  htmlElem = htmlElem.firstElementChild;
  // transfer iteration control to another instance of the
  // current generator - enables sub iteration...
  while (htmlElem) {
    yield* DomTraverseGenerator(htmlElem);
    htmlElem = htmlElem.nextElementSibling;
  }
}
```

- benefit to this generator-based approach for DOM traversal
  - *callbacks are not required*

- able to consume the generated sequence of nodes with a simple loop
  - *and without using callbacks*

- able to use generators to separate our code
  - *code that is producing values - e.g. HTML nodes*
  - *code consuming the sequence of generated values*

ES6 Generators & Promises - traversal with generators

- traversed using depth-first search

- algorithm tries to go deeper into tree structure
  - *when it can't it moves to the next child in the list*

- e.g. define a `class` to create a Node
  - *creates with value and arbitrary amount of child nodes*

```javascript
// Node class - holds a value and arbitrary amount of child nodes...
class Node {
  constructor(value, ...children) {
    this.value = value;
    this.children = children;
  }
}
```

Then, we create a basic node tree,

```javascript
// define basic node tree - instantiate nodes from
const root = new Node(1,
  new Node(2),
  new Node(3,
    new Node(4,
      new Node(5,
        new Node(6)
      ),
      new Node(7)
    )
  ),
  new Node(8,
    new Node(9),
    new Node(10)
  )
)
```

- various implementations we might create for a traversal generator...

ES6 Generators & Promises - generator function

- e.g. depth first generator function for traversing the tree

```javascript
// FN: depthFirst generator
function* depthFirst(node) {
  yield node.value;
  for (const child of node.children) {
    yield* depthFirst(child);
  }
}

// log tree recursion
console.log([...depthFirst(root)]);
```

ES6 Generators & Promises - generator - exchange data with a generator

- also send data to a generator

- enables bi-directional communication

- a pattern might include
  - *request data*
  - *then process the data*
  - *then return an updated value when necessary to a generator*

ES6 Generators & Promises - generator - exchange data with a generator - example

```javascript
// generator function - send data to generator - receive standard argument
function* MessageGenerator(data) {
  // yield a value - generator returns an intermediator calculation
  const message = yield(data);
  yield("Greetings, "+ message);
}


const messageIterator = MessageGenerator("Hello World");
const message1 = messageIterator.next();
console.log("message = "+message1.value);


const message2 = messageIterator.next("Hello again");
console.log("message = "+message2.value);
```

- first call with the `next()` method requests a new value from the generator
  - *returns initial passed argument*
  - *generator is then suspended*

- second call using `next()` will resume the generator, again requesting a new value

- second call also sends a new argument into the generator using the `next()` method

- newly passed argument value becomes the complete value for this yield
  - *replacing the previous value `Hello World`*

- we can achieve the required bi-directional communication with a generator

- use `yield` to return data from a generator

- then use iterator's `next()` method to pass data back to the generator

ES6 Generators & Promises - generator - detailed structure

Generators work in a detailed manner as follows,

- **suspended start**
  - *none of the generator code is executed when it first starts*

- **executing**
  - *execution either starts at the beginning or resumes where it was last suspended*
  - *state is created when the iterator's `next()` method is called*
  - *code must exist in generator for execution*

- **suspended yield**
  - *whilst executing, a generator may reach `yield`*
  - *it will then create a new object carrying the return value*
  - *it will yield this object*
  - *then suspends execution at the point of the yield...*

- **completed**
  - *a `return` statement or lack of code to execute*
  - *this will cause the generator to move to a complete state*

ES6 Generators & Promises - generators & iterables

fibonacci number generator

- example generator for Fibonacci sequence

- generator will output an infinite sequence of numbers

- we may also call individual iterations of the sequence
  - e.g.

```
// generator function - value per iteration & done will not return true...
function* fibonacci() {
  // define start values for fibonacci sequence
  let previous = 0;
  let current = 1;
  // loop will continue to iterate fibonacci sequence
  while(true) {
    // return current value in fibonacci sequence
    yield current;
    // compute next value for sequence...
    const next = current + previous;
    // update values for next iteration of loop in fibonacci sequence
    previous = current;
    current = next;
  }
}

// instantiate iterator object using fibonacci generator
const g = fibonacci();

// call iterator
console.log(g.next());
```

- to improve performance, and prevent memory and execution timeout
  - *add **memoisation** to script*
  - *a type of local cache for the execution of the algorithm...*

ES6 Generators & Promises - async I/O using generators

- use generators and generator helpers to create simple async input and output
  - *use with saving data &c.*
  - *a consistent and abstracted usage design for a custom generator*

```javascript
// called with passed generator function
function saveItems(itemList) {
  const items = [];
  const g = itemList();
  return more(g.next());
  function more(item) {
    if (item.done) {
      return save(item.value);
    }
    return details(item.value);
  }
  function details(endpoint) {
    // check inputs are called & location...
    console.log(`details called - ${endpoint}`);
    return fetch(endpoint)
      .then(res => res.json())
      .then(item => {
        items.push(item);
        return more(g.next(item));
      })
  }
  function save(endpoint) {
    // check output is called & location...
    console.log(`save endpoint - ${endpoint}`);
    /*return fetch(endpoint, {
      method: 'POST',
      body: JSON.stringify({ items })
    })
    .then(res => res.json());*/
  }
}

saveItems(function* () {
  yield './assets/input/items.json';
  yield './assets/input/notes.json';
  return './assets/output/journal.json';
})
```

an example usage for generators and promises,

- `async` function takes a *generator*, calls it, and creates the required *iterator*
  - *use iterator to resume generator execution as needed*
  - *declare a handle function - handles one return value from generator*
  - *one iteration of iterator*
  - *if generator result is a promise & resolves successfully - use iterator's `next` method*
  - *promise value sent back to generator*
  - *generator resumes execution*
  - *if error, promise gets rejected*
  - *error thrown to generator using iterator's `throw` method*
  - *continue generator execution until it returns `done`*

- `generator` - executes up to each `yield getJSON()`
  - *promise created for each `getJSON()` call*
  - *value is fetched async - generator is paused whilst fetching value...*
  - *control flow is returned to current invocation point in `handle` function whilst paused*

- `handle` function
  - *yielded value to `handle` function is a promise*
  - *able to use `then` and `catch` methods with promise object*
  - *registers success and error callback*
  - *execution is able to continue*

ES6 Generators & Promises - lots of examples

e.g.

- generator
  - *basic*
  - *basic-iterator*
  - *basic-iterator-over*
  - *basic-loop*
  - *basic-dom*
  - *basic-send-data*
  - *basic-send-data-2*

- promises
  - *basic*
  - *basic-cors-flickr*
  - *basic-xhr-local*
  - *basic-promise-all*
  - *basic-promise-race*

- generator & promise - async
  - *basic*

ES2017 Async & Await

- in ES2017, JavaScript gained native syntax to describe asynchronous operations

- now use *async/await* to work with asynchronous operations

- `Async` functions allow developers to take a promise-based implementation
  - *then use synchronous-like patterns of a generator*
  - *e.g. async implementation with sync usage patterns...*

- `await` may only be used inside `async` functions
  - *denoted with the `async` keyword*

- `async` function works in a similar manner to standard generators
  - *e.g. suspending execution in local context until a promise settles*

- if awaited expression is not originally a promise object
  - *it will be cast to a promise in this context...*

ES2017 Async & Await - example 1

- example usage with try/catch

```
async function read() {
  // use try/catch to handle errors in awaited promises within async function
  try {
    const model = await getRandomBook();
  } catch (err) {
    console.log(err);
  }
}
// call function as usual
read();
```

- use return Promise object

```
async function read() {
  const model = await getRandomBook();
}
// call function as usual - work with return promise object...
read()
  .then()
```

ES2017 Async & Await - example 2

Node.js and command line

- example usage with command line arguments
- custom Promise object
- async/await with try/catch block
- initial error handling

```javascript
/*
 * basic-error.js
 * - error handling for async...
 */

function getArgs() {
  // Node Process command line arguments
  const args = process.argv;
  // custom Promise object with resolve and reject
  return new Promise((resolve, reject) => {
      if (args[2] === 'test') {
        resolve(args);
      } else {
        reject('no args');
      }
  });
}

async function main() {
  try {
    let data = await getArgs();
    return data;
  } catch(e) {
    throw new Error(`main failed...${e}`);
  }
}

main()
.then(console.log)
.catch(console.log);
```

ES2017 Async & Await - example 3

initial `fetch`

```javascript
// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}
```

ES2017 Async & Await - example 4

- example fetch usage

```
/*
 * basic-async1.js
 * async called with sync-like try/catch block
 * 'awaits' return from fetch to local JSON file
 */


// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}


// FN: async/await
async function read() {
  try {
    const notes = await getNotes();
    console.log(`notes FETCH successful`);
  } catch (err) {
    console.log(err);
  }
}

read();
```

- Demo - Async & Await - Fetch example

ES2017 Async & Await - example 5 - part 1

sample iterable functions

```javascript
/*
 * FNs: iterable computed data
 * functions support all major ES6 data structures
 * - arrays, typed arrays, maps, sets...
 */


// FN: iterable entries() - default iterator for data structure entries
function dataEntryIterator(data) {
  for (const pair of data.entries()) {
    console.log(pair);
  }
}


// FN: iterable keys() - default iterator for data structure keys
function dataKeysIterator(data) {
  for (const key of data.keys()) {
    console.log(key);
  }
}


// FN: iterable values() - default iterator for data structure values
function dataValuesIterator(data) {
  for (const value of data.values()) {
    console.log(value);
  }
}
```

async and await usage - a bit of fun...

```javascript
// FN: async/await
async function read() {
  try {
    // await return from FETCH for notes.json file
    const data = await getNotes();
    const notes = data['notes'];
    // wrap return notes array in iterator
    const iter = notes[Symbol.iterator]();
    // test iterator with next for each result...
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(`notes FETCH successful`);
    dataEntryIterator(notes);
    dataKeysIterator(notes);
    dataValuesIterator(notes);
  } catch (err) {
    console.log(err);
  }
}

read();
```

- Demo - Async & Await - example with iterables

HTML5, CSS, & JS - example - part 1

add grid layout

- update the layout of our Travel Notes application to include a grid layout

- apply this grid layout to the overall application
  - *organisation and presentation of the notes*

- remove the centred, fixed width for the body in our style.css stylesheet

- removes centre styling, results in content spanning full width of browser window

- add the grid layout to help us control this layout

```
<link rel="stylesheet" type="text/css" href="assets/styles/grid.css">
```

- then modify content categories, child elements to use new grid css

```
<!-- document header -->
<header>
  <div class="row">
    <div class="col-5">
      <h3>travel notes</h3>
      <h5>record notes from various places visited...</h5>
    </div>
    <div class="col-7"></div>
  </div>
</header>
```

Image - HTML5, CSS, & JS - grid layout

travel notes

record notes from various places visited...

add note

[   ] add

app's copyright information, additional links...

Grid Layout - Updated Header

HTML5, CSS, & JS - example - part 2

add grid layout

- update our main content to position the `note-input` and `note-controls`

```html
<!-- note input -->
<section class="note-input">
  <div class="row">
    <div class="col-12">
      <h5>add note</h5>
      <input><button>add</button>
    </div>
  <div>
</section>
<!-- note controls for delete... -->
<section class="note-controls">
  <div class="row">
    <div class="col-12">
      <button id="notes-delete">Delete all</button>
    </div>
  </div>
</section>
```

- still need to amend style.css to remove additional fixed styling

Image - HTML5, CSS, & JS - grid layout 2

travel notes

record notes from various places visited...

add note

[   ] add

Delete all

note

app's copyright information, additional links...

Grid Layout - mixed grid and fixed
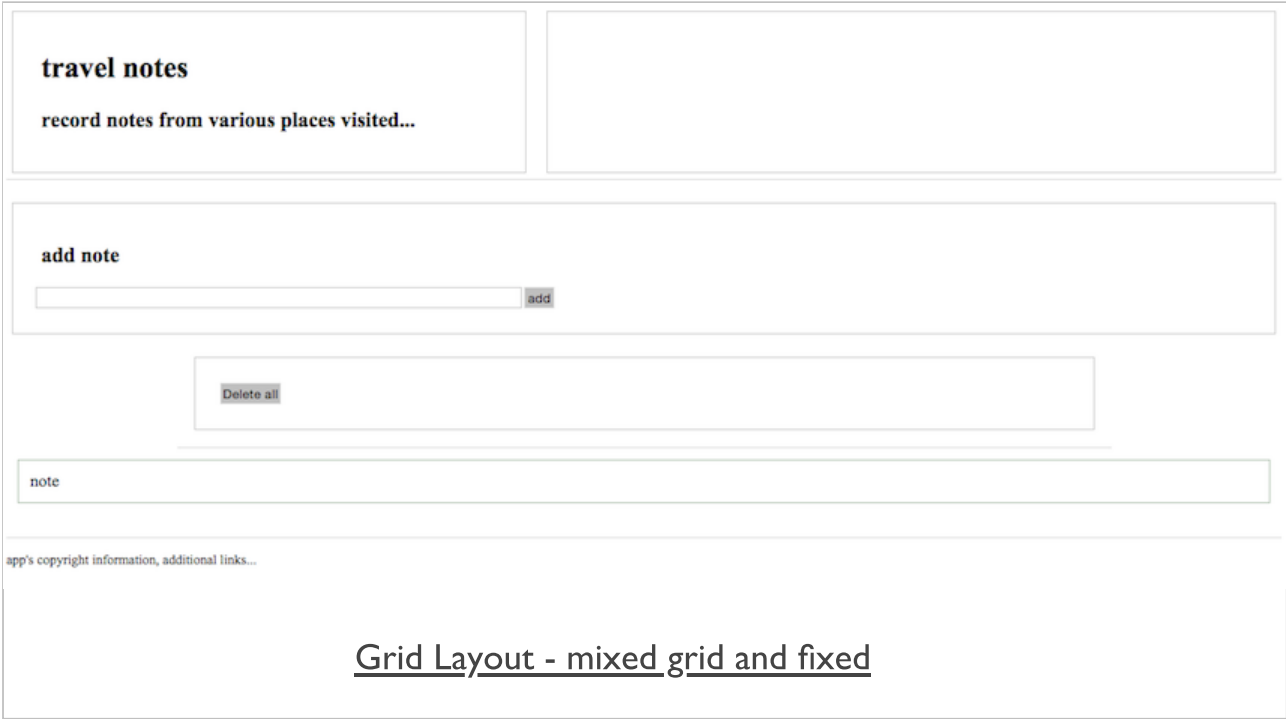
add grid layout

- fix mixed rendering by removing width, margin, and padding for `.note-controls`

```css
/* note controls */
.note-controls {
  border-bottom: 1px solid #dedede;
  display: none;
}
```

- continue to update Travel Notes app
  - *modify output for notes*
  - *add further options for users*

DEMO - Travel Notes - grid layout with media queries

HTML5, CSS, & JS - example - part 4

add flex to grid layout

- an additional option to consider - flex layouts
  - *a recent W3 working draft*
  - *aims to provide efficient way to align and proportion content*

- known as **Flexbox Layout**
  - *idea is to apportion width and height for content*
  - *proportions relative to container even when their size is unknown or dynamic*

- flex layout could, in theory, replace a full grid layout
  - *considered more a complement to overall grid structure*

- defined flex container expands items to fill the container's available space
  - *can also shrink them to prevent any possible overflow*

- think of a flex layout as supporting multiple directions
  - *direction agnostic*

- many properties available for **flex**
  - *focus upon styling flex container and any flex items*

HTML5, CSS, & JS - example - part 5

add flex to grid layout

- we might specify CSS properties for a flex container

```css
.flex-container {
display: flex; /* defines container as flex */
flex-direction: row; /* defines positioning of items added to container */
flex-wrap: wrap; /* defines whether to wrap items to another line */
justify-content: flex-start; /* defines start point and distribution of items */
}
```

- allows us to position our container starting at the left
  - *items contained in a row*
  - *contained items wrapping to additional lines if necessary*

- many additional options available for each property

- also add rulesets for specific styling of items within a flex container

- we could add properties to a flex item such as
  - *specify the order of the flex items*
  - *whether a particular item can grow or shrink relative to content*
  - *default size of an item before any remaining space is distributed*
  - *individual alignment for a given item...*

HTML5, CSS, & JS - example - part 6

add flex to notes

- flex container and items useful for organising and positioning our notes

- due to uncertainty about content, size, and general note requirements
  - *flex positioning and styling removes the need for assumptions or fixed sizes*

- we can start to modify the styling and rendering of our notes using flex

```css
/* flex item */
.flex-item {
  flex-basis: 300px; /* default size before extra */
  flex-grow: 1; /* all items will be equal */
}
```

- gives us a default smallest size for each note

- then the ability for each note to grow to fill the row as required

- also work with responsive layouts
  - *due to the minimum size and the option to grow for each item*
  - *and wrap flex items per flex container*

- modify and update styles as we develop travel notes app

DEMO - Travel Notes - grid layout with flex notes

Image - HTML5, CSS, & JS - Flex Notes

travel notes

record notes from various places visited...

menu...

search...

add note

[   ] add

Delete all

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

cannes

nice

monaco

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

antibes

juan les pins

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

eze

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

st tropez

app's copyright information, additional links...

Grid Layout - flex notes

Image - HTML5, CSS, & JS - Flex Notes 2

travel notes

record notes from various places visited...

menu...

search...

add note

[   ] add

Delete all

cannes	nice
monaco	menton

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

antibes	st tropez

app's copyright information, additional links...

Grid Layout - flex notes - medium

Image - HTML5, CSS, & JS - Flex Notes 3

travel notes

record notes from various places visited...

menu...

search...

add note

[   ] [add]

[Delete all]

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

cannes

monaco

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

antibes

app's copyright information, additional links...

Grid Layout - flex notes - small

add flex to notes

Notes with Flex and Media Queries

HTML5, CSS, & JS - example - part I

add AJAX and JSON - load notes from json

- update our **travel notes** app to allow us to load some test persistent notes from a local JSON file

- initial JSON is as follows

```json
{
  "travelNotes": [{
    "created": "2015-10-12T00:00:00Z",
    "note": "a note from Cannes..."
  }, {
    "created": "2015-10-13T00:00:00Z",
    "note": "a holiday note from Nice..."
  }, {
    "created": "2015-10-14T00:00:00Z",
    "note": "an autumn note from Antibes..."
  }]
}
```

HTML5, CSS, & JS - example - part 2

add AJAX and JSON - load notes from json

- add option to load notes from JSON as app initially loads
  - *use deferred promise pattern*
  - *checks source JSON as it loads via the promise*
  - *then outputs the end result*

- start with the following update

```
//get the notes JSON
function getNotes() {
  //.get returns an object derived from a Deferred object - do not need explicit
  var $deferredNotesRequest = $.getJSON (
    "docs/json/notes.json",
    {format: "json"}
  );
  return $deferredNotesRequest;
}
```

HTML5, CSS, & JS - example - part 3

add AJAX and JSON - load notes from json

- help us better manage logic of our notes from app's loading and execution
  - *create two separate JS files*

- our updated structure might be as follows

```
...
|- assets
   |- scripts
      |- travel.js
      |- notes.js
...
```

- we can extend this further, as needed by app features and data

HTML5, CSS, & JS - example - part 4

add AJAX and JSON - load notes from json

- add our `.when()` function to the app's loader
  - *.when() function accepts a deferred object*
  - *in our case a limited promise*

- then allows us to chain additional deferred functions
  - *including required .done() function*

- for returned data, use standard response object to get `travelNotes`
  - *then iterate over the array for each property*
  - *for each iteration, we can simply call our createNote function*
  - *builds and renders required notes to the app's DOM*

```javascript
//use deferred object from getJson
$.when(getNotes()).done(function(response) {
  //get travelNotes object
  var $travelNotes = response.travelNotes
  //process travelNotes array
  $travelNotes.forEach(function(item) {
    //check each property
    if (item !== null) {
      //get note
      var note = item.note;
      //create each note for rendering
      createNote(note);
    }
  });//end foreach
});
```

HTML5, CSS, & JS - example - part 5

add AJAX and JSON - load notes from json

- simple problem - existing `createNote()` function does not accept a parameter

- need to update the logic of that function to accept and handle a parameter

- also requires a quick update to any functions and calls to the `createNote()`
  - *event handlers for creating a new note using the* add *button and keypress within the input field*

```
//manage input field and new note output
function createNote(data) {

  ...
  //conditional check for data
  if (data !== "") {
  //set content for note
  $note.html(data);

  ...
  }
}
```

HTML5, CSS, & JS - example - part 6

add AJAX and JSON - load notes from json

- update our event handlers for the note input button and input field keypress as follows,

```js
//handle user event for `add` button click
$(".note-input button").on("click", function(e) {
  var $note_data = getNoteInput();
  //call note builder function
  createNote($note_data);
});
```

```js
//handle user event for keyboard press
$(".note-input input").on("keypress", function(e) {
  //check code for keyboard press
  if (e.keyCode === 13) {
    var $note_data = getNoteInput();
    //call note builder function
    createNote($note_data);
  }
});
```

- our notes now load by default as the app starts
- note input button and keypress work as expected
- DEMO - travel notes & JSON

Working with APIs - part 1

remote api options - Flickr

- **Travel Notes** app loads data from a local JSON file
- add option to load different types of data using remote APIs
  - *Flickr API for images, tags...*

- basics and principles are similar to the patterns we've already seen and tested
- test a sample JSON return from the Flickr API
- JSON return - useful properties for app
  - *title*
  - *link*
  - *media (direct url for image - where available)*
  - *description*
  - *...*

- public feed for searching public photos, videos, groups, recent activity...
- Flickr API Public Feed - Cannes and France

Working with APIs - part 2

working with Flickr API

- query Flickr's public feed for photos
  - *we can use our now familiar pattern for requesting JSON*

```
//get the Flickr public feed JSON for images
function getImages() {
  //.get returns an object derived from a Deferred object - do not need explicit
  var $deferredNotesRequest = $.getJSON (
  "http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
    { tags: "cannes,france,boules",
      tagmode: "all",
      format: "json"
    });
  return $deferredNotesRequest;
}
```

- need to make a few specific modifications to the request
  - *JSONP to avoid browser security restrictions*

Working with APIs - part 3

working with Flickr API

- Flickr's public feed includes options
  - *eg: a specific user ID for photos, various tags, how tags are interpreted by the search...*

- use our `.when()` function to load and render some test images from Flickr

```javascript
$.when(getImages()).done(function(response) {
  console.log("done..."+response);
  //use jQuery's generic iterative function for the response...
  $.each( response.items, function( i, item ) {
    buildImage(item.media.m);
    //limit test images to 8
    if ( i === 7 ) {
      return false;
    }
  });
});
```

- DEMO - AJAX and JSON - Flickr api

HTML5, CSS, & JS - example - part 7

working with Flickr API - update travel notes

- add option to Travel Notes app to allow a user to view images from Flickr

- need to update app's HTML, CSS, and JS

- modify how our notes, and associated options, are rendered to our users

- add a search option for photos on Flickr

- render our images to match the notes

- app's structure still reflects three primary content categories
  - *`header, main, and footer` with slight modifications to the `main` category*

- `main` content category updated to create two distinct rows for initial content
  - *contain defined semantic containers*

- row containing `.note-input` and Flickr search option `.contextual-choice`
  - *then split this row into two columns of 6*

HTML5, CSS, & JS - example - part 8

working with Flickr API - update travel notes HTML

- updated HTML for `.note-input` and Flickr search `.contextual-choice`

```html
<div class="row">
  <!-- note input -->
  <section class="note-input col-6">
    <h5>add note</h5>
    <input><button>add</button>
  </section>
  <!-- contextual choice -->
  <section class="contextual-choice col-6">
    <h5>search flickr</h5>
    <input><button>search</button>
  </section>
</div>
```

HTML5, CSS, & JS - example - part 9

working with Flickr API - update travel notes HTML

- update the HTML for rendering the images
  - *add alongside our notes*

- create another row for these containers
  - *add two section containers for `.note-output` and `.contextual-output`*

- make `.note-output` slightly larger to show primary app focus

```html
<div class="row">
  <!-- note output -->
  <section class="note-output col-7 flex-container">
  </section>
  <!-- contextual output -->
  <section class="contextual-output col-5 flex-container">
  </section>
</div>
```

HTML5, CSS, & JS - example - part 10

working with Flickr API - update travel notes JS

- add further functionality to **Travel Notes** app

- split our JS logic into three files to help with oranisation
  - *a main loader file, `travel.js`,*
  - *and a file each for notes and contextual options*

- updated app structure for JS

```
...

|- assets
   |- scripts
      |- contextual.js
      |- notes.js
      |- travel.js
...
```

- underlying logic for the notes will remain the same
  - *move loading of default notes to the `travel.js` main loader file*

- updates for searching, returning, and rendering images from Flickr
  - *added to the `contextual.js` file*

HTML5, CSS, & JS - example - part 11

working with Flickr API - update travel notes JS

- test Flickr API in our app using some set data for image tags
  - *respond to the user clicking on the search button*
  - *submit our query to Flickr*
  - *process the returned JSON for the images*
  - *render them for viewing*

- request and process our images using the familiar pattern

```javascript
//get the Flickr public feed JSON for images
function getImages(data) {
  var img_tags = data;
  //.get returns an object derived from a Deferred object - do not need explicit
  var $deferredNotesRequest = $.getJSON (
    "http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
    { tags: img_tags,
      tagmode: "all",
      format: "json"
    });
  return $deferredNotesRequest;
}
```

HTML5, CSS, & JS - example - part 12

working with Flickr API - update travel notes JS

- returned data using standard deferred promise object
  - *add a new function to handle the processing of the images*

```javascript
function processImages(data) {
  $.when(getImages($img_data)).done(function(response) {
    //use jQuery's generic iterative function for the response...
    $.each( response.items, function( i, item ) {
      createImage(item.media.m);
      //limit test images to 4
      if ( i === 3 ) {
        return false;
      }
    });
  });
}
```

- using deferred promise object with `.when()` function chained to `.done()` function

- add jQuery's generic iterative function to help us process the response
  - *instead of standard JavaScript* `.forEach()` *option*

- loop through each value, and pass the image to our new function, `createImage()`
  - *ready for rendering to our app's DOM*
  - *limit number of images for testing*

HTML5, CSS, & JS - example - part 13

working with Flickr API - update travel notes JS

```javascript
//manage new image output
function createImage(data) {
    //create each image element
    var img = $('<img class="flex-img">');
    //add image
    img.attr("src", data);
    //append to DOM
    $(".contextual-output").append(img);
}
```

- `.createImage()` function accepts a parameter for image data

- then process ready for rendering to the app's DOM

- image is added to a new `img` element with a new class of
  `.flex-img`
  - *creates a flex item for rendering*

- added to the new `.contextual-output` section

- rendered images displayed as thumbnails for the user
  - *complementary to the existing notes*

HTML5, CSS, & JS - example - part 14

working with Flickr API - update travel notes JS

- to add images to the app
  - *a user can enter their requested tags in the search field*
  - *then click on the `search` button to return any available images*

- event handler for this `search` button click uses the requested tags
  - *passes them as a parameter to the `processImages()` function*

```
//handle user event for image `search` button click
$(".contextual-choice button").on("click", function(e) {
  //test tags for testing image search
  $img_data = "cannes,france,boules"
  //process images
  processImages($img_data);
});
```

Image - HTML5, CSS, & JS - Travel Notes & Flickr

travel notes

record notes from various places visited...

menu...

search...

add note

[   ] add

search flickr

[   ] search

Delete all

Cannes, a resort town on the French Riviera, is synonymous with glamour thanks to its world-famous film festival. Its Boulevard de la Croisette, curving along the coast, is lined with sandy beaches, upmarket boutiques and palatial hotels. It's also home to the Palais des Festivals, a modern building complete with red carpet and Allée des Stars – Cannes' walk of fame.

Nice, capital of the French Riviera, skirts the pebbly shores of the Baie des Anges. Founded by the Greeks and later a retreat for 19th-century Europe's elite, the city today balances old-world decadence with modern urban energy. Its sunshine and liberal attitude have long attracted artists, whose work hangs in its museums. With vibrant markets and diverse restaurants, it's also renowned for its food.

Antibes is a resort town between Cannes and Nice on the French Riviera (or Côte d'Azur). It's known for its Mediterranean beaches, annual Jazz à Juan music festival and old town enclosed by 16th-century ramparts. Luxury yachts moor at the huge Port Vauban marina, overlooked by star-shaped, 16th-century Fort Carré. The Promenade Amiral-de-Grasse walkway along Vauban's walls has views of the Alps.

app's copyright information, additional links...

Travel Notes & Flickr - test loading images

HTML5, CSS, & JS - example - part 15

working with Flickr API - update travel notes CSS

- need to update and modify existing CSS
  - *helps with correct rendering of the thumbnail images*

- CSS additions are initially modest
  - *reflects integration with existing app, grid, and flex layouts*

- add new ruleset for image rendering in the `.contextual-output` section

```css
/* contextual output images */
.contextual-output img {
  margin: 5px;
  padding: 5px;
  border: 1px solid #b1c4b1;
}
```

- update `.flex-container` class to change `justify-content` property to value of `space-around`

- add new ruleset for a `.flex-img` class.

```css
/* flex image */
.flex-img {
  flex-basis: 150px;
  flex-grow:0;
}
```

- specify size of a thumbnail image
  - *initially restrict their ability to grow relative to flex*

working with Flickr API - update travel notes JS

- we can now request, process, and render images from Flickr to Travel Notes app
  - *still need to accept and process search queries from search input field.*

- add option to check search input field
  - *then submit query to Flickr for images*

```javascript
//get input value for image search
function getImageInput() {
  //define img value
  var img_val = "";
  //define input field
  var $img_tags = $(".contextual-choice input");
  if ($img_tags.val() !== "") {
    img_val = $img_tags.val();
    return img_val;
  } else {
    return img_val;
  }
}
```

HTML5, CSS, & JS - example - part 17

working with Flickr API - update travel notes JS

- use `getImageInput()` function with a modified `processImages()` function

```js
//process image production, loading, and pass to rendering
function processImages() {
  //check img visibility for contextual-output - clear existing images
  if (checkVisible($(".contextual-output img")) === false) {
    //empty existing images
    $(".contextual-output").empty();
  }
  //get data from image search input field
  var $img_data = getImageInput();
  //use image data to get images, and pass for rendering
  $.when(getImages($img_data)).done(function(response) {
    console.log("done..."+response);
    //use jQuery's generic iterative function for the response...
    $.each( response.items, function( i, item ) {
      createImage(item.media.m);
      //limit test images to 4
      if ( i === 3 ) {
        return false;
      }
    });
  });
}
```

HTML5, CSS, & JS - example - part 18

working with Flickr API - update travel notes JS

- updated `processImages()` function then called within event handlers
  - *for the search button and a keypress in the search input field*

```javascript
//handle user event for image search button click
$(".contextual-choice button").on("click", function(e) {
  //process images
  processImages();
});


//handle user event for keyboard press
$(".contextual-choice input").on("keypress", function(e) {
  //check code for keyboard press
  if (e.keyCode === 13) {
    //process images
    processImages();
  }
});
```

- DEMO - travel notes & Flickr

Image - HTML5, CSS, & JS - Travel Notes & Flickr

travel notes

record notes from various places visited...

menu...

search...

add note

add

search flickr

search

Delete all

Curral das Freiras is a civil parish in the municipality of Câmara de Lobos in the Portuguese archipelago of Madeira. The population in 2011 was 2,001, in an area of 25.03 km². It is situated in the mountainous interior of the island.

Câmara de Lobos (Portuguese pronunciation: [literally, Portuguese: chamber of the wolves) is a municipality, parish and city in the south-central coast of the island of Madeira. Technically a suburb of the much larger capital city of Funchal, it is one of the larger population centres and an extension of the Funchal economy.

Funchal is the largest city, the municipal seat and the capital of Portugal's Autonomous Region of Madeira. The city has a population of 111,892, making it the 6th largest city in Portugal, and has been the capital of Madeira for more than five centuries. Because of its high cultural and historical value, Funchal is one of Portugal's main tourist attractions. It is also popular as a destination for New Year's Eve, and it is the leading Portuguese port on cruise liner dockings.

app's copyright information, additional links...

Travel Notes & Flickr - different layout

working with Flickr API - update travel notes JS

- room for improvement, updates, abstraction, and general refactoring of the existing code

- return to this issue when we consider refactoring the code in general
  - *there are still a few simple features we need to add*

- for example,
  - *add images to the `.contextual-output` section, resize `.note-output` section*
  - *moves focus to the current images*
  - *check loading progress of the notes and images*
  - *show feedback to the user*
  - *need to output a title for the images*
  - *set using the search query*

working with Flickr API - modify travel notes JS

- first modification is to resize the `.notes-output`
  - *create more space for the images*
  - *gently shift focus to the new images*

- update existing `.createImage()` function in the `contextual.js` file

```
//manage new image output
function createImage(data) {
...
    if (checkVisible($(".contextual-output img")) === true) {
      $(".note-output").removeClass("col-12");
      $(".note-output").addClass("col-4");
      $(".contextual-output").fadeIn("slow");
    }
...
}
```

- add check to ensure images are not visible in the DOM

- remove current class from `.note-output` section
  - *12 column class for the grid*

- add new grid class to resize `.note-output` to 4 columns
  - *then fade in the `.contextual-output` class*
  - *set in the app's HTML to a class of `.col-8`*

HTML5, CSS, & JS - example - part 21

working with Flickr API - modify travel notes JS

- next modification is some initial error handling
  - *checking for an empty array of images from the returned Flickr JSON*

- check `processImages()` function for an empty array of image items

```
...

if (response.items.length === 0) {
  var img = "";
  createImage(img);
} else {
  //return images from items array...
}
...
```

- checks images in the items array for the promise object

- if not, send an empty variable as a parameter to our `createImage()` function

HTML5, CSS, & JS - example - part 22

working with Flickr API - modify travel notes JS

- check for empty value in `createImage()` function
  - *handle the simple errors as follows*

```js
if (data !== "") {
  //create each image element
  var $img = $('<img class="flex-img">').attr("src", data);
  //add image
  img_output = $img;
} else {
  var $img_error = $('<p class="flex-item error">').html("No images available..."
  //add error
  img_output = $img_error;
}
```

- we've abstracted the return variable for the image output
  - *can hold either the image or the error output...*

- add a check to see whether the `.contextual-output` section is visible or not

- modify the column class for the `.note-output` section

- then append our image output

- then show the `.contextual-output` section within the app

- DEMO - travel notes & Flickr

Image - HTML5, CSS, & JS - Travel Notes & Flickr

travel notes

record notes from various places visited...

menu...

search...

add note

add

search flickr

search

Delete all

Curral das Freiras is a civil parish in the municipality of Câmara de Lobos in the Portuguese archipelago of Madeira. The population in 2011 was 2,001, in an area of 25.03 km². It is situated in the mountainous interior of the island.

Câmara de Lobos (Portuguese pronunciation: [literally, Portuguese: chamber of the wolves) is a municipality, parish and city in the south-central coast of the island of Madeira. Technically a suburb of the much larger capital city of Funchal, it is one of the larger population centres and an extension of the Funchal economy.

Funchal is the largest city, the municipal seat and the capital of Portugal's Autonomous Region of Madeira. The city has a population of 111,892, making it the 6th largest city in Portugal, and has been the capital of Madeira for more than five centuries. Because of its high cultural and historical value, Funchal is one of Portugal's main tourist attractions. It is also popular as a destination for New Year's Eve, and it is the leading Portuguese port on cruise liner dockings.

No images available. Please try a different search.

app's copyright information, additional links...

Travel Notes & Flickr - error checking

HTML5, CSS, & JS - example - part 23

working with Flickr API - modify travel notes JS

- continue to modify and build our Travel Notes app

- add some metadata for the returned images
  - *using the title and link from the search query response*

- add initial metadata output in the `contextual.js` file
  - *modify the `processImages()` function*
  - *metadata from Flickr JSON response in the deferred promise object*

```
...
//create object for search metadata
var search_meta = {title:response.title, link:response.link};
...
```

- then pass this to a new function, called `metaOutput()`

```
//prepare and render metadata for returned search...
function metaOutput(data) {
  if (data !== "") {
  //search metadata from response
  var search_title = data.title;
  var search_link = data.link;
  //build heading output for metadata heading
  var metaHeading = '<h6>'+search_title+' | <a href="'+search_link+'">Flickr</a><
  //render metadata to contextual-output
  $(".contextual-output").prepend(metaHeading);
  }
}
```

- DEMO - travel notes & Flickr - initial metadata

Image - HTML5, CSS, & JS - Travel Notes & Flickr

Travel Notes & Flickr - initial metadata

HTML5, CSS, & JS - example - part 24

travel notes - basic refactoring of JS

- as we continue to add features and modify existing code
  - *may start to see unnecessary repetition and function calls in the code*

- eg: initial error handling for our contextual images
  - *`createImage()` function is being called in the `processImages()` function*
  - *called regardless of returned image data*

- `createImage()` is being used unnecessarily to manage the error handling

- move check to `processImages()` function
  - *then call function to render necessary error message*

```
function outputError(message) {
  var $img_error = $('<p class="flex-item error">').html(message);
  //check for visible contextual-output - if not visible
  if (checkVisible($(".contextual-output")) === true) {
    $(".note-output").removeClass("col-12");
    $(".note-output").addClass("col-4");
  }
  //append output to DOM
  $(".contextual-output").append($img_error);
  //fade in contextual-output with appended results
  $(".contextual-output").fadeIn("slow");
}
```

travel notes - basic refactoring of JS

- updated `processImages()` function can call
  `.outputError()` function as needed

```
...
if (response.items.length !== 0) {
//logic to add metadata and each image...
}
else {
  var img_error = "No images available - please try a different search.";
  outputError(img_error);
}
...
```

- use this function to output error messages for any type of contextual data

- also remove some unnecessary replication of code
  - *by adding a simple function to change an element's class*

```
//modify element class - from, to
function changeClass(element, size1, size2) {
    $(element).removeClass(size1);
    $(element).addClass(size2);
}
```

- resize a class, for example to modify our grid output
  - *call this function - pass the selector to update, original class to remove, and new class to add*

HTML5, CSS, & JS - example - part 26

working with Flickr API - modify travel notes JS

- add a modification to check for the image loading and the notes
  - *offer status feedback to the user*

```
//add initial loader spinner for ajax...
$(".contextual-output").html('<img class="spinner" src="assets/images/ajax-loader
```

- remove it when the deferred promise object has returned

```
//remove ajax spinner
$(".spinner").remove();
```

- DEMO - travel notes & Flickr - spinner

Image - HTML5, CSS, & JS - Travel Notes & Flickr

travel notes

record notes from various places visited...

menu...

search...

add note

add

search flickr

search

app's copyright information, additional links...

Travel Notes & Flickr - spinner

Demos

- Fetch API
  - *basic usage*
  - *catching errors*
  - *Fetch API & Promise.all*
  - *Fetch API & Promise.race*

- Generators - plain JS
  - *Basic*
  - *Basic Iterator*
  - *Basic Iterator Over*
  - *Basic DOM Traversal*
  - *Basic Send Data*
  - *Basic Send Data 2*
  - *Pass generator to function*

- Promises - plain JS
  - *Basic*
  - *Basic CORS Flickr*
  - *Basic Promise All*
  - *Basic Race*
  - *Basic XHR Local*
  - *Promise error handling with catch*
  - *Promise from scratch*
  - *Promise.resolve*

- Travel notes app - series 3
  - *DEMO 1 - Travel notes - grid layout with media queries*
  - *DEMO 2 - Travel notes - demo2*

- Travel notes app - series 4
  - *DEMO 1 - Travel Notes & JSON*
  - *DEMO 2 - Travel Notes & Flickr*
  - *DEMO 3 - Travel Notes & Flickr - error checking*
  - *DEMO 4 - Travel Notes & Flickr - initial metadata*

- *DEMO 5 - Travel Notes & Flickr - spinner*

Resources

- Flickr
  - *Flickr API - Public feeds*
  - *Flickr API - Public feed - public photos & video*

- 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*

- Various
  - *Create your own AJAX loader*
  - *W3 Selector API*

- W3
  - *W3 - CSS Flexible Box Layout Module 1*
  - *W3 - JS Object*
  - *W3 - JS Performance*