Week 07 Tutorial

Sample Answers

COMP(2041|9044) 19T2

1. What are the MDN docs? Why are they useful?

   The MDN docs are a extensive and detailed set of documentation for most of the web technologies you'll work with when doing web design. This includes HTML, CSS, JS, HTTP etc. If you are in doubt this is your best bet on where to get api information and helpful examples. You can use other sites such as w3schools but be careful as some examples on sites like these are out of date, incorrect or misleading. MDN is a more up to date documentation set.

2. Why should you not mimic the functionality of `img` `button` etc. with `divs`?

   HTML should on it's own be a skeleton of the data you wish to display, by using semantic tags you can state clearly what data is a image, what data is a heading etc. This is good for readability and allows your site to function at the bare minimum without CSS if need be. MOST IMPORTANTLY however many web accessability tools use the HTML to provide features such as dictation to users. These tools are used by many people with disabilities to access the web and by using divs for everything these tools struggle to gain a understanding of your web page making it inaccessable to those users.

3. Sam placed the following bit of HTML on their website then visited the site, what will they see?

```
<p dfsd="1"></p>
```

   A normal p tag, any invalid attributes are ignored. In fact HTML will never fail to parse. Whenever the HTML parser in most browsers meets invalid HTML it will either ignore it or try and reform it into something that does make sense. For example in HTML if you forget to close a tag it will automatically close it before ending the parse. This makes HTML resiliant but also means you will often get weird functionality with no indication of why.

4. What is the `DOCTYPE html` at the top of a HTML file?

   It's like a shebang, it states that what follows is a HTML file.

5. Write a CSS statement to make all `h1` elements red and bolded.

```
h1 {
  color: red;
  font-weight: bold;
}
```

6. Why would you want to use CSS classes as opposed to direct tag selectors?

   Classes allow you to share common CSS accross different elements rather then repeating yourself. This not only makes your code cleaner and more maintable but also helps you keep a consistant style. In addition having a class called `red-shadow` is a lot easier to read and debug then `div ul li p`

7. What happens when we define the same CSS rule twice like this:

```
p {
    color: red;
}

p {
    color: blue;
}
```

   Second one overrides the first. In cases where you don't want this you can use the `!important` property but use this carefully as it makes code much more difficult to debug.

/

8. What is the purpose of the `sources`, `network` and `console` tab on dev tools?

    sources shows you all the HTML, CSS and JS files loaded in the current site. network shows you all the initial and subsequent network requests from the site. It also can show you the responses, any live websocket connections and keeps information on how long each request took so you can narrow down on what resources are causing the most issues. The console is a direct terminal which will run any javascript you write in the context of the current tab, basically letting you test out code or see errors/warnings from any running js.

9. Where can I go for more info on CSS, positioning and website layout?

    There are a couple of good links provided at the course material site. They describe some of the most common solutions to the annoying problem of getting a website to layout the way you want it to.

10. What does it mean when we say Javascript is single threaded

    Javascript will run all the code it needs to one after another. This means that if you click a button 5 times and each button press triggers a function to run, the functions will be run in order one by one. This is important to remember because if you have any Javascript running that takes a while to complete, no other JS will run, no matter what triggers it, until the previous task is complete. (you can get around this by using web workers however, something we'll cover in week 3)

11. What is Automatic Semicolon Injection in JS?

    Javascript relies on semicolons much like C does to tell it when one statement is done and the next begins, however js will automatically put in semicolons into areas it thinks you forgot to put them in. 90% of the time this works fine but there are cases where the algorithm guesses wrong so in general try to consistently use semicolons. (you can invest in a JS linter into your IDE which can help make this easier)

12. Why do we avoid using `var` or simply declaring a variable with `x=1`?

    Using var is problematic thanks to the concept of hoisting and delcaring a variable without any preamble means it's global, similar to Perl. We prefer to use const and let as a result to get more reliable behavior of variables in scopes.

13. What is an arrow function (otherwise known as an anonymous function or a lambda) and how does the syntax work?

    A short hand to declare a quick function, it follows the following syntax. It's important to remember that these functions have no concept of `this` unless they can grab a specific instance of the object from a higher scope.

    ```
    one_argument => expression

    (multiple_arguments) => expression

    one_argument => {
      return;
    }

    (multiple_arguments) => {
      return;
    }
    ```

14. What is the difference between `for(x in items)` and `for(x of items)`

    In quite a annoying way `for(x in items)` will iterate over the index's of items, i.e 0,1,2,3 whereas `for(x of items)` will iterate over the actual elements in items, i.e "apple", "orange", etc.

15. What is the issue with the below code?

/

```
import { sum } from 'package'

console.log(`2 + 2 = ${sum(2, 2)}`)
```

```
function sum(a, b) {
    return a + b
}
```

Missing export statement in package.js

```
export function sum(a, b) {
    return a + b
}
```

16. How and where is JavaScript run?

    JavaScript is environment agnostic; but most commonly it's known to run in the browser, 'client-side', and through Node.js on serverside code. JavaScript is not a compiled language, instead parsed and evaluated into bytecode at runtime.

17. Can JavaScript on a website I visit remove all my files?

    The short answer is no. JavaScript that is run in the browser has no concept of direct I/O and no interface to interact with your local file system. In the browser context, JavaScript runs with the limitations that the browser affords it. Clientside JavaScript must be paried with some sort of server if file system interaction is required.

    That's not to say that JavaScript can't do malicious things within the browser context. A webpage that has been compromised might contain JavaScript that uses your own credentials to make requests on your behalf, log your keystrokes in input fields, or do computationally intensive calculations like bitcoin mining in the background while you're browsing.

18. What is Node.js?

    Node.js is a serverside JavaScript environment that provides an interface with the operating system. Node.js still uses familiar JavaScript syntax but adds features like disk I/O, process management, among other things. A fairly unique feature of node is its async nature: Node.js does not block on I/O calls, which means it can continue execution of blocks that aren't dependent on the the I/O call without pausing, waiting, and wasting its CPU time. The tradeoff however is this can make for some unexpected program behaviour if you're not aware of it.

19. Given a dataset that looks similar to the below. How might you go about finding the average age of men, with first name's starting with 'A'. Hint: you might want to think about using some combination of map, filter, reduce.

/

```
// let's note the power with cleaning some user data.
const users = [
  {
    name: 'Jeff',
    age: 52,
    gender: 'male'
  },
  {
    name: 'Andy',
    age: 25,
    gender: 'male'
  },
  {
    name: 'Sarah',
    age: 30,
    gender: 'female'
  },
  {
    name: 'Phoebe',
    age: 21,
    gender: 'female'
  },
  {
    name: 'Doris',
    age: 81,
    gender: 'female'
  }
];
```

/

```
/*
   JavaScript supports some really nice Array
   methods which make the use of for loops in general
   largely unnecessary.

   key among thse are map, filter and reduce
*/

// let's note the power with cleaning some user data.
const users = [
  {
    name: 'Jeff',
    age: 52,
    gender: 'male'
  },
  {
    name: 'Andy',
    age: 25,
    gender: 'male'
  },
  {
    name: 'Sarah',
    age: 30,
    gender: 'female'
  },
  {
    name: 'Phoebe',
    age: 21,
    gender: 'female'
  },
  {
    name: 'Doris',
    age: 81,
    gender: 'female'
  }
];

// the answer in a broken down way
const isMale = (person) => person.gender === 'male'
const startsWith = (letter) => (person) => person.name.startsWith(letter)
const sum = (total, current) => total + current

// this is one way to get the data we want
const agesOfMaleANames = users
                           .filter(isMale)
                           .filter(startsWith('A'))
                           .map(({ age }) => age)

// now to resolve the average.
const answer = agesOfMaleANames
                           .reduce(sum, 0)/agesOfMaleANames.length

// To ram things home let's get the average age of females.
const females = users.filter(user => user.gender === 'female');
const ageSum = (sum, current) => current.age + sum;

const average = females.reduce(ageSum, 0) / females.length;

// or creating a summary string with map + reduce.
const usersString = users
                      .map(({ name, age, gender }) =>
                         `${name} (${gender}) is ${age} years of age.`)
                      .reduce((all, curr) => `${all}\n${curr}`);
```

20. Understanding 'this' in JavaScript can be a real problem for beginners. Work through the below code and see if you can identify what 'this' will be in each expression and how it will be evaluated.

```
// this can be confusing
const o = {
   bb: 0,
   f() {
      console.log(this.bb);
   }
};

// What does this print out
o.f();

// What does this line do
let a = o.f;

// What would this print out
a();

const oo = {bb: 'Barry'};

// What does call do and what will it print out?
a.call(oo);

// What does bind do, is f() the same as a()?
const f = a.bind(oo);

// what does this print out
f();
```

```
// this can be confusing
const o = {
   bb: 0,
   f() {
      console.log(this.bb);
   }
};

// "this" is the object "o"
o.f(); // Prints out 0

// This line gets a reference to the function
// basically "unbinding it"
let a = o.f;

// we call the function directly, not through
// the object as we did above with "o.f()" so
// "this" is the global context
a(); // prints "undefined"

const oo = {bb: 'Barry'};

// call's "a" but sets "this" to be "oo"
a.call(oo); // prints out Barry

// f() is a new function identical to a() but with "this"
// set to "oo" for any standard call
const f = a.bind(oo);

f(); // prints out Barry
```

21. JavaScript recently added support for classes to the language. Consider the code below. which uses the traditional object creation syntax.

    Can you explain the prototype?

/

```
// note the use of this in this special constructor
// also note the caps (a convention for constructor functions)
function Person(firstName, lastName, age) {
   this.firstName = firstName;
   this.lastName  = lastName;
   this.age       = age;
}

Person.prototype.getFullName = function () {
   return `${this.firstName} ${this.lastName}`;
};

Person.prototype.canDrinkAlcohol = function () {
   return this.age >= 18;
};

// now if we call the constructor function we get this
const jeff = new Person('Jeff', 'Goldblum', 50);
// => Person { firstName: 'Jeff', lastName: 'Goldblum', age: 50 }

jeff.getFullName(); // 'Jeff Goldblum'
```

How might you translate the above to a class?

```
// 1. A prototype is a special property of the function object.
// When the new keyword is invoked, the function 'binds'
// anything that is referenced by this in the function call
// to a new object; which *also* inherits from the function prototype.
// More here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

/*
   An alternate way of processing complex objects as classes
   This is the same as the prototype.js example
*/
class Person {
   constructor(firstName, lastName, age) {
      this.firstName = firstName;
      this.lastName  = lastName;
      this.age       = age;
   }

   getFullName() {
      return `${this.firstName} ${this.lastName}`;
   }

   canDrinkAlcohol() {
      return this.age >= 18;
   }
}

// now if we call the constructor function we get this
const jeff = new Person('Jeff', 'Goldblum', 50);
jeff.getFullName(); // 'Jeff Goldblum'
jeff.canDrinkAlcohol(); // true
// => Person { firstName: 'Jeff', lastName: 'Goldblum', age: 50 }
```

22. Write a function that given a array of strings representing a shopping cart prints out each item in alphabetical order with a count of how many times the item appeared in the array.

```
const cart = ['Apple', 'Orange', 'Apple', 'Strawberry', 'Orange'];

countCart(cart);
// above would print
// Apple 2
// Orange 2
// Strawberry 1
```

/

```
const cart = ['Apple', 'Orange', 'Apple', 'Strawberry', 'Orange'];

function countCart(cart) {
  const count = {};

  for (const item of cart) {
    // check if the key exists, if it doesn't
    // add the key with a initial count of 1 otherwise add 1 to count
    count[item] = count[item] ? count[item] + 1 : 1;
  }

  // one way to do this (there are many ways)
  for (const item of Object.keys(count).sort()) {
    console.log(item, count[item]);
  }
}

countCart(cart);
```

23. The above function was unchanged but the input array was changed to be a array of objects.

```
const cart = [
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Orange',
    cost: 4.50
  },
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Strawberry',
    cost: 6.70
  },
  {
    name: 'Orange',
    cost: 4.50
  }
];

countCart(cart);
```

What would the function output?

Using a object as a key just makes js call "toString" on the object and use the result as a key, i.e all objects would be interperted as "[object Object]". The output would thus be "[object Object] 5".

24. Rewrite the above function to work with an array of objects and print out the total cost of each item rather then just the count, i.e for the above example the output would be

```
Apple 4.6
Orange 9
Strawberry 6.7
```

Standard solution

```
const cart = [
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Orange',
    cost: 4.50
  },
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Strawberry',
    cost: 6.70
  },
  {
    name: 'Orange',
    cost: 4.50
  }
];

function countCart(cart) {
  const count = {};
  for (const item of cart) {
    // be careful cause currCost _can_ be 0
    count[item.name] = count[item.name] !== undefined ? count[item.name] + item.cost : item.cost;
  }

  for (const item of Object.keys(count).sort()) {
    console.log(item, count[item]);
  }
}

countCart(cart);
```

Alternate Answer using a map structure and arrow functions

/

```
const cart = [
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Orange',
    cost: 4.50
  },
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Strawberry',
    cost: 6.70
  },
  {
    name: 'Orange',
    cost: 4.50
  }
];

function countCart(cart) {
  const count = new Map();
  for (const item of cart) {
    if (!count.has(item.name)) count.set(item.name, item.cost);
    else count.set(item.name,count.get(item.name)+item.cost);
  }
  const allItems = [...count.entries()].sort((a,b)=>a[0]>b[0]);
  allItems.map((a)=>console.log(a[0],a[1]));
}

countCart(cart);
```

25. Write a function that takes in an array identical to above and returns a total cost for all items but do not use *any* explict loops (for while etc.).

```
const cart = [
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Orange',
    cost: 4.50
  },
  {
    name: 'Apple',
    cost: 2.30
  },
  {
    name: 'Strawberry',
    cost: 6.70
  },
  {
    name: 'Orange',
    cost: 4.50
  }
];

function total(cart) {
  const sum = (acc, curr) => curr + acc;
  return cart.map((i)=>i.cost).reduce(sum,0);
}
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

26. I wrote a program which given a string returns a integer representing the number of seconds a user wants a time to be set for as shown below.

```
set timer for 5 minutes  --> 300
set a timer for 10m      --> 600
timer 8minutes           --> 480
new timer 60seconds      --> 60
timer for 60s            --> 60
banoodles                --> undefined
```

Here is the program

```
function extractTime(s) {
  let r = /(\d+)\s+([ms])/g;
  r = r.exec(s);
  let [num,unit] = r;
  if (unit === "m") num *= 60;
  return num;
}

console.log(extractTime("set timer for 5 minutes"), 300);
console.log(extractTime("set a timer for 10m"), 600);
console.log(extractTime("timer 8minutes"), 480);
console.log(extractTime("new timer 60seconds"), 60);
console.log(extractTime("timer for 60s"), 60);
```

But it doesn't work, fix the code so it works.

```
function extractTime(s) {
  // should be \s* as 0 spaces are valid
  let r = /(\d+)\s*([ms])/g;
  r = r.exec(s);
  // check if the regex failed
  if (r === null) return undefined;

  // 0th element in r is the whole match, 1th element is group 1
  let [_,num,unit] = r;

  // regex returns a string, for minutes the *= 60 converts
  // from string to num but for seconds we'd be returning a string
  num = parseInt(num);
  if (unit === "m") num *= 60;
  return num;
}

console.log(extractTime("set timer for 5 minutes"), 300);
console.log(extractTime("set a timer for 10m"), 600);
console.log(extractTime("timer 8minutes"), 480);
console.log(extractTime("new timer 60seconds"), 60);
console.log(extractTime("timer for 60s"), 60);
```

27. Create a function that takes in no arguments, and when called the first time emits 0, then 1, then 2, ... etc, etc, etc. Each additional call should output a higher number. Hint: Think about how you might construct a function to do this. Maybe another helper function might come in handy.

```
// a simple closure
function closureFunction() {

  // here the count variable is only in the function's scope
  let count = 0;
  // now we are going to define a new function
  // but because we are defining it within another
  // function we are going to give it a backback
  // with the current context (the count variable)
  // this is called a closure
  return function() {
    return count++;
  }
}

const counter = closureFunction();
// we now have the function function(){return count++}
// but tus variable also has a little backback with the context
// the function was defined in
// function(){return count++} [count = 0]
counter(); // 0

// same function but the backpack is different
// function(){return count++} [count = 1]
counter(); // 1

// .....
```

28. JavaScript offers some quite powerful functional tools. Consider the example below and note the use of small, stateless functions to solve a larger multi-faceted problem and walk through how it works.

```
const shoppingCart = [
  { item: 'Apple', price: 10 },
  { item: 'Orange', price: 12 },
  { item: 'Pineapple', price: 5 },
];

// long version
const multiply_long = function(a) {
  // return a function with a
  // backpack which has a set
  // value "a"
  return function(b) {
    return a * b;
  }
}
// short and sweet version
const multiply = a => b => a * b;

const pluck = key => object => object[key];

// let's say tax of 10% for GST
// and a 5 % first customer discount
const discount = multiply(0.95);
const tax      = multiply(1.10);
const sum      = (acc, curr) => curr + acc;

// Now, for some simple readable, easy to reason about code.
const totalPrice = shoppingCart.map(pluck('price'))
                               .map(discount)
                               .map(tax)
                               .reduce(sum, 0);

console.log(totalPrice);
```

COMP(2041|9044) 19T2: Software Construction is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au