Javascript Questions

* **Explain event delegation?** 
  + Event delegation is a technique involving adding event listeners to a parent element instead of adding them to the descendant element.
  + The listener will fire whenever the event is triggered on the descendant element due to event bubbling up the DOM.
* **Explain how “this” words in javascript?** 
  + A basic explanation of what the value of “this” is depends on how the function is called
  + A few examples of what the value of “this**”** can be are

1. If the “new” keyword is used when calling the function, **“this”** inside the function is brand new object

2. If “apply”, “call”, or “bind” are used to call/create a function, “this” inside the function is the object

3. If a function is called as a method, such as obj.method(), “this” is the object that the function is a property of

4. If a function is invoked as a free function innovation, meaning it was invoked without any of the condition present above, “this”is the global object. In the browser it is the “window” object. If **“**use strict” then “this” will be undefined instead of the global object.

5. If multiple of the above rules apply the rule is higher wins & will set the “this” value.

6. If the function is an ES2015 arrow function, it ignores all the rules above & receives the “this”value of its surrounding scope at the time it is created

* **What is an Immediately Invoked Function Expression ?** 
  + IIFE, is a good way at protecting the scope of your function & the variableswithin in it
* **Explain how prototypical inheritance works?** 
  + All javascript objects have a prototype property, that reference to another object
  + When a property is accessed on an object & if the property is not found the javascript engine looks at the objects prototype, & the prototypes prototype and so on, until is finds the property defined on one of the prototypes or until it reaches the end of the prototype chain.
* **What’s the difference between a variable that is “null”, “undefined” or “undeclared”? How would you go about checking for any of these states?** 
  + ***Undeclared*** variables are created when you assign a value to an identifier that is not previously created using var, let or const. Undeclared variables will be defined globally, outside of the current scope. Undeclared variables are bad just like how global variable are bad. To check for them, wrap its usage in a try/catch block
  + ***Undefined*** is variable that has been declared but not assigned a value that one type of undefined. If a function does not return any value as the result of executing it is assigned to a variable the variable also has the value of undefined. To check for it, compare the strict equality(===) operator of typeof which will give the “undefined” string.
  + ***Null*** will be explicitly assigned to the null value. It represent no value & is different from undefined in the sense that is has been explicitly assigned. To check for null use the strict equality(===) operator.
* **What is a closure, & how/why would you use one?** 
  + Closure is the combination of a function & the lexical environment within which that function was declared.
  + Closure are functions that have access to the outer functions variable scope chain even after the outer function has returned
  + I would use one for data privacy/emulating private methods with closures also for partial applications or currying
* **Can you describe the main difference between a “Foreach” loop & “Map” & why would pick one verse the other?** 
  + **Foreach**, iterates through the elements in an array, executes a callback for each element & doesn’t return a value
  + **Map**, iterates through the elements in an array, it “Maps” each element to a new element by calling the function on each element, creating a new array as a result
* **What’s a typical use case for anonymous functions?** 
  + They can be used in IFFEs to encapsulate some code within a local scope so that variables declared in it do not leak to global scope
  + As a **callback** that is used once & doesn’t need to be used anywhere else the code will seem more self-contained & readable when handlers are defined right inside the code calling them rather than having to search elsewhere to find the function body
* **How do you organize your code? Module pattern, classical inheritance?** 
  + Since I use react/redux which utilize a single-directional data flow based on flux architecture.
  + I would represent my app’s models using plain objects & write utility pure functions to manipulate these objects
  + Stateis manipulated using actions & reducers like in any other redux application
* **What’s the difference between host objects & native objects?** 
  + Native objects are objects that are part of the javascript language defined by the ECMAScript specification such as String, Math, RegExp, Object, Function, etc..
  + Host objects are provided by the runtime environment browser or node such as window, XMLHTTPRequest, etc.
* **Explain Function.prototype.bind?** 
  + The “bind()” method creates a new function that when called has its “this” keyword set to the provided valued, with a given sequence of arguments preceding any provided when the new function is called
* **what’s the difference between “call()” & “apply()”?** 
  + Both “call()” & “apply()” are sued to invoke functions & the first parameter will be used as the value of “this” within the function.
  + “call()” takes in comma-separated argument as the next argument while “apply()” takes in an array of arguments as the next argument.
* **What’s the different between feature detection, feature inference, & using UA string?** 
  + **Feature detection**, involves working out weather a browser supports a certain block of code & running different code depending on whether it does so that the browser can always provide a working experience rather crashing/erroring in some browsers
  + **Feature inference**, checks for feature just like feature detection but uses another function because it assumes it will also exist
  + **UA String**, is browser-report string that allows the network protocol peers to identify the application type, operating system, software vendor or software version of the requesting software user agent.
* **What is “hoisting”?** 
  + Hoisting is a term used to explain the behavior of variable declarations in your code.
  + Variablesdeclared or initialized with **“var”** keyword will have their declaration moved up to the top of the current scope which is refer to as hoisting but only declaration is hosted the assignment will stay where it is
* **Describe event bubbling?** 
  + When a event triggers on a DOM element it will attempt to handle the event if there is a listener attached then the event is bubbled up to its parent & the same things happens
  + This bubbling occurs up the elements ancestors all the way to the document, event bubbling is the mechanic behind event delegation
* **Whats the difference between an “attribute” & a “property”?** 
  + Attributes are defined on the HTML markup but properties are defined on the DOM
* **Why is extending build-in Javascript objects not a good idea?** 
  + Extending a built-in/native javascript object means adding properties/function to its prototype.
  + When yore code uses a few libraries that both extend the Array.prototype by adding the same contains method, the implementation will overwrite each other and you code will break
* **Why is the difference between “==” & “===”?** 
  + “==” is the abstract equality operator while “===” is the strict equality operator.
  + “==“ operator will compare for equal after doing type conversion.
  + “===” operator will not do type conversion so if two values are the same type will return false
* **Explain the same-origin policy w/ regards to Javascript?** 
  + Same-origin policy prevents javascript from making request across domain boundaries
  + An origin is defined as combination of URI scheme, hostname, and port number
  + This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that pages Document Object Model
* **Why is it called ternary expression, what does the word “Ternary” indicate?** 
  + “Ternary” indicates three & ternary express accepts three operands, the test condition, the “then” expression & the “else” expression
* **What is Classical Inheritance?** 
  + Class is description of an object to be created. Classes inherit from the classes and create subclass relationships. Its pattern language like Java uses and javascript tries to mimic but there’s no actual classic inheritance in javascript.
* **Why is “use strict;”? What are the advantages & disadvantages to using it”** 
  + “use strict” is a statement used to enable strict mode to entire scripts or individual functions. Strict mode is way to opt into a restricted variant of Javascript.
  + The Advantages are

1. Makes it impossible to accidentally create global variables.

2. Makes assignments which would otherwise silently fail to throw an exception

3. Requires that function parameter names be unique

4. “this” is undefined in the global context

5. It catches some common coding bloopers, trowing exceptions

6. It disables features are confusing or poorly shout out.

* + Some disadvantages are

1. Many missing features that some developers might used to

2. No more access to “function.caller” & “function.arguments”

3. Concatenation of scripts written in different script modes might cause issues

* **Why you might want to create static class members?** 
  + Static class members properties/methods are not tied to a specific instance of class & have the same value regardless of which instance is referring to it
  + Static properties are typically configuration variables & static method are usually pure utility functions which do not depend on the state of instance
* **How can you share code between files?** 
  + On the client if variables/function are declared in the global scope all scripts can refer to them
  + On the server the common way has been to use CommonJS, each file is treated as a module & it can export variable & functions
  + ES2015 defines a module syntax which allows to do what AMD & CommonJS does
* **Explain the difference between mutable & immutable objects?** 
  + Mutable objects have fields that can be changed, immutable objects have no fields that can be changed after the object is created
* **What advantage is there for using the arrow syntax for a method in a constructor?** 
  + The main advantage of using an arrow function as a method inside a constructor is that the value of **“this”** gets a set at time of the function creation & can’t change after that
  + When the constructor is used to create a new object, **“this”** will always refer to that object
* **Explain the difference between synchronous & asynchronous functions?** 
  + **Synchronous** functions are blocking while asynchronous function are not. In synchronous functions, statement complete before the next statement is run. In this case, the program is evaluated exactly in order of the statements & execution of the program is paused if one of the statements take a very long time.
  + **Asynchronous** functions usually accept a callback as a parameter & execution continue on the next line immediately after the asynchronous function is invoked. The callback is only invoked when the asynchronous operation is complete & the call stack is empty. Heavy operations such as loading data from a web server or querying a database should be done asynchronously so that the main thread can continue executing other operations instead of blocking until that long operation to complete
* **What is event loop? What is the difference between call stack & task queue?** 
  + The event loop is a single-thread loop that monitors the call stack & checks if there is any work to be done in the task queue
  + If the call stack is empty & there are callback function in the task queue, a function is dequeued & pushed into the call stack to be executed
* **What are the differences between ES6 class & ES5 function constructors?** 
  + It’s simply a more convenient notation for the usual constructor function. Inside, it’s the same prototype-based object model
  + You get couple of advantages from using classes, though class declaration are not hoisted as with other ES6-specific features like “let” & “const”, proper lexical scoping is preserved. ES6 classes not their objects also give you a reflective “name” property
* **Can you give an example for destructuring an object or an array?** 
  + Destructuring is an expression available in ES6 which enables a succinct & convenient way to extract values of Objects or Arrays & place them into distant variable
* **What are the differences between variables created using “let”, “var” or “const” ?** 
  + Variable declared using the “var” keyword are scoped to the function in which they are created or if created outside of the any function to the global object
  + “let” & “const” are blocked scoped meaning they are only accessible within the nearest set of curly braces like functions, if-else block, or for-loop
  + “var” allows variables to be hoisted, meaning they can referenced in code before they are declared. “let” & “const” will not allows this instead throwing an error
  + Redeclaring a variable with “var” will not throw an error, but “let” & “const” will
  + “let” & “const” differ in that “let” allows reassigning the variables value while “const” does not
* **Can you offer a use case for the arrow “=>” function syntax? How does this new syntax differ from other functions?** 
  + Arrow functions simplify the syntax needed to create functions w/ a need for the “function**”** keyword
  + “this” within arrow functions is also bound to the enclosing scope which is different compared to regular functions where the “this” is determined by the object calling it. Lexically-scoped “this” is useful when invoking callbacks especially in React components
* **What is the definition of a higher-order function?** 
  + Higher—order function is any function that takes one or more function as arguments, which is uses to operator on some data, and/or returns a function as result
  + Higher-orderfunctions are meant to abstract some operator that is performed repeatedly such as “map()” which takes an array & function as argument
  + Higher-order function doesn’t need to be manipulating arrays as there are many uses cases for returning a function from another function
* **Can you give an example of a curry function & why this syntax offers an advantage?** 
  + Curry is a pattern where a function with more than one parameter is broken into multiple functions that when called in series will accumulate all of the required parameters one at a time
  + For function to be curried, it needs to start out as one function then broken out into a sequence of functions that each accepts one parameter
* **What are the benefits of using spread syntax & how is it different from rest syntax?** 
  + ES6 spread syntax is very useful when coding in a functional paradigm as we can easily create copies of arrays or objects
  + When using spread, you are expanding a single variable into more
  + When using rest, you are collapsing all remaining arguments of a function into on array
* **What are the ways to handle Asynchronous in Javascript**
  + **Callback,** the original method of handling asynchronicity.
  + **Promise,** makes it easier to chain promise together using “.then()”
  + **Async/Await,** most recent way of doing with clean syntax
* **What the difference between Promise & Observable**
  + **Promise** handles a single event when an async operation completes or fails
  + **Observable** is like a Stream and allows to pass zero or more events where the callback is called for each event
  + Observable is preferred over Promisebecause it provides the feature of Promise and more. With Observableit doesn’t matter if you want to handle how many ever events. You can Utilize the same API in each case