**GENERAL**

**### START NEWEST**

Lo 187 - Styles, Attributes Classes

Async functions run in background, with one or more awaits, until finished.

Async await - don't have to chain .then methods, don't have to return anything, don't have to use a callback function.

Async await – synthetic sugar over consuming promises (like classes).

Async await - can't attach .catch method so a try catch block is used.

Try-catch used in regular Javascript also (not specific to async await). Probably been in JS since beginning.

DOM API: Contains many methods and properties provided by the browser such as:

Event Target: addEventListener, removeEventListener

Node Object: parentNode, childNodes, cloneNode, textContent

Element Object: querySelector, addEventListener, createElement, innerHTML, append, closest, remove, insertAdjacentHTML, children, parentElement, classList, setAttribute, matches, scrollIntoView, etc

Number.parseFloat()

Number.isFinite() – Infinity and string numbers (‘23’) are false

Advantage of promises is allows for then sequential processing.

Async function immediately returns a promise and runs in the background.

. then chaining - Can ‘return’ a promise string or a promisified function. If returning a string then the promise is resolved immediately.

.catch – Can be located anywhere in the promise chain and will catch errors located before it. Can have multiple .catch methods in a chain and .then statements located below a .catch will be processed (promise chain will continue). If termination of chain is desired then locate .catch at the end of the .then chain. All errors will cause subsequent .then's in chain to be bypassed until the next closest .catch is executed. Promise is reset to ‘pending’ when next .then is initiatially processed? (Not a new promise?)

Always use try catch with async functions

forEach - Continue and break do not work.

Function side effect – performing work without returning anything – such as with a forEach.

Form button click event is also triggered by the enter key within a input field.

All numbers are represented as floating point (even integers). i.e. 23 == 23.0

Bankist Changes

Add other accessories

Error for bad login, invalid input for loan, transfer, delete.

**### END**

Closure, Scope Chain, Prototype Chain

Num to Str - toFixed – performs boxing to object so toFixed() method can work on it - rounds and/or pads zeros then coverts number to a string.

Str to Num – parseFloat

Number.isFinite() - !isNaN - checks valid floating point / IsInteger - checks valid integer number.

Infinity valid w/NaN.

Number() and parseInt() both used to convert string to number.

ParseFloat() / parseInt() – converts string (with or w/o trailing letters (css values)) to number. Use parseFloat instead of parseInt() because it allows for decimals, or if unsure of integer value. Goto number functions for all types parseFloat() - get css values for decimals and integers;

isFinite() - checks for valid number floating and integers. Use Number.isFinite() instead of isNaN to check for valid number. Infinity and ‘10’ (string number) - isNaN. For integers only – use Number.isInteger.

floor() instead of trunc to round down because it accounts for negative numbers.

Integer rounding = Math.round(), Decimal rounding = +numbervalue.toFixed().

Remainder operator % to determine even or odd number – Divide by 2 of no Remainder then even number. isEven = n => n % 2 === 0.

Boxing – element (such as a string) converted to object in order to perform methods (temporarily converting string to an object so methods can be called on it)

Number and Math objects are considered namespaces – like scopes – allow for unique names within a larger context.

Namespace refers to the programming paradigm of providing scope to the identifiers (names of types, functions, variables, etc) to prevent same name collisions between them.

Can manually throw errors or Javascript can also throw them with try-catch. With try/catch can gracefully continue or terminate processing, instead of an execution error that abruptly terminates processing.

Try/throw/catch/finally.

Use nullish coalescing operator ?? instead of || if zero, ‘’, false, and NaN are to be treated as truthy. Only undefined and null are treated as falsey for ?? operator.

Short circuit || operand to set default values and && operator to execute code in second operand if 1st operand is true. Use the logical assignment operator ||= (includes 0 as falsey) or nullish assignment ??= (only undefined or null). Also use &&= logical assignment operator to assign value if 1st is truthy.

Optional chaining ?. Works for undefined and null values – evaluates to undefined. (? same rules as nullish ??)

BEM : Low specificity – classes, not nested

{} – Code block

Partial application: Specifying arguments in a bind method. Can also use currying.

**STRICT MODE**

Restricted variant of JavaScript. Stricter parsing and error handling at runtime.

Eliminates silent errors by changing them to throw errors. Changes previously accepted "bad syntax" into real errors.

Fixes mistakes that make it difficult for JavaScript engines to perform optimizations.

Prohibits some syntax likely to be defined in future versions of ECMAScript.

Strict mode applies to entire scripts, or to individual functions. It doesn't apply to block statements.

Automatic for modules, class declarations and expressions.

Prevents accidental creation of global variables. Forgetting ‘let’ in loop. Typing errors for variables.

Cannot use undeclared variables.

As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

Deleting a variable, function, or argument will cause error.

Prevents duplication of parameter names.

In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.

In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**BUNDLING**

Better to install development tools locally instead of globally so it forces usage of most current version.

Babel – Babel presets allow for transpiling back to es5 for all browsers except for browsers with less than .25% market usage. Uses Babel Plug-in List web page for new JS syntaxes.

Keep current with JS syntaxes by reviewing Babel Plug-in List web page.

New features (functions) such as many array methods and Promise need to be polyfilled.

For polyfills use library such as ‘core-js’. Default behavior is to polyfill functions even if not used so for final build, to reduce bundle size, can specify functions to be polyfilled. Ex. import ‘core-js/stable/array/find’.

import ‘regenerator-runtime/runtime’ to polyfill async functions.

Parcel automatically uses Babel to transpile.

Bundler – 1) Dynamically installs dependencies from npm. 2) Combines modules into one script file 3) Compresses / minifies 4) Dead code elimination 5) Provides development server 6) Increases performance

module.hot.accept – parcel command: will replace changed module without reloading the web page – such as requiring a log in. Saves state (makes state persistent).

**EVENTS**

Events and event object is made available by browser.

event.stopPropagation() – stops propagation of particular event for all ancestors. All other eventListeners for other types of events remain active for element and all ancestors.

event.stopImmediatePropagation() – inactivates all other eventListeners for a particular element after current eventListener is executed for that element.

Removing eventListeners: When using const cloneEl = el.cloneNode(true) to create new element then using el.replaceWith(cloneEl) that replaces content – but still must update reference pointer by assigning el = cloneEl.

Default for form element with submit button is to send data to source and reload page.

Event Loop (Browser) – Moves tasks from MicroTasks Queue and Callback Queue to Call Stack.

e.currentTarget: element the eventListener is attached to (‘this’ for each event handler).

e.target: element that triggered the event.

e.stopPropagation - not usually used.

**CLASSES / OBJECTS**

Object Constructor functions simulate classes where the name of the Constructor function simulates the name of an actual class. (JS does not have real classes.) Objects created with the Constructor function become instances of the Constructor function (instanceof).

Object.prototype: \_\_proto\_\_ at top of chain. Object is a constructor function.

Private properties not available in parent class or sub class.

Classes: Sub classes without super and with no constructor still uses parent constructor. Sub classes with constructor will not use parents constructor for itself.

With super: execute parents constructor first before using this. Because of this properties and methods in sub-class have not been defined yet and cannot be used in the parent constructor.

Extends – creates prototype inheritance. extends: on a subClass is the same as parentClass.prototype (prototype property of the constructor function) = ‘code in parentClass that is to be made available to subClass’. Note: the above code will overwrite constructor function in \_\_proto\_\_ - it is better to add properties via prototype after.

“this” in parent class constructor refers to “this” in object being created in subclass.

Class fields defined above constructor are automatically populated, implicitly using “this” behind the scene, in the constructor (before any constructor logic is performed – this is true for all classes) after the “super” constructor is executed for the parent-class.

All code is executed within constructor method inside of class when object is instantiated.

“new” keyword when calling a class constructor function: 1) Creates object, 2) points “this” to object instance, 3) returns object automatically without a return statement.

Order of classes not important but must be declared before executing them.

Classes are special constructor functions. Classes can be defined by a declaration or an expression(assigned to a variable - with or without a name)

\_\_proto\_\_ exists on “all” objects – points to prototype: property of parent.

Constructor functions- regular functions but called with the New keyword. Can be function expression or declaration but not an arrow function because no this keyword.

Every object (and function therefore) has a prototype property which is an object, or, a private property that points to a prototype object.

Every prototype is an object which has its own \_\_proto\_\_, or private property that points to a prototype object.

Using the new keyword when calling a function does 4 things.

1) Creates new object {}

2) Sets value of this = object, then calls function with this pointing to new object.

3) Links object to prototype (creates \_\_proto\_\_ property and assigns same reference address as the constructor function prototype property). So that frank.\_\_proto\_\_ === Person.prototype, or, Person.prototype.isPrototypeOf(frank)

4) Function automatically returns object

Every function has a prototype property

Methods should be assigned to the prototype property of the Constructor function. i.e. Person.prototype = function(){}

Also, properties can be assigned to prototype object as well such as Person.prototype.species = ‘Homo Sapien’

Also, frank.hasOwnProperty(‘fname’) === true, frank.hasOwnProperty(‘species’) === false

Super – causes an object to be built for the parentClass.

Avoid overwriting \_\_proto\_\_ - better to add properties to prototype ex. subClass.prototype.newProperty = variable/method?

Polymorphism – A child class can overwrite a method it inherited from a parent class.

An enumerable property in JavaScript means that a property can be viewed if it is iterated using the for…in loop or Object.keys() method. All the properties which are created by simple assignment or property initializer are enumerable by default.

Object.create – Creates an object using another object as its prototype (prototype object becomes part of prototype chain). New properties for created object can use Object.defineProperties() directly in Object.create().

Object.assign- creates new object (or creates a copy of merged objects). Any new properties with the same name as on a copied object – the value will be overwritten by a copied object.

Also use Spread operator to create an object.

Object methods are function expressions (function values), not function declarations.

Object methods are non-enumerable by default in classes.

Object elements are ordered by order in which they are added. Except number keys which are sorted.

Partial application: Specifying some arguments in a bind method. Somewhat similar to currying.

Modules - prevent global namespace pollution such as in multiple scripts. Exported values are by reference. Imported Modules are hoisted and executed immediately. Strict mode is default.

Modules: 1 Encapsulate functionality 2 Privatize data 3 Expose a public API

JSON.stringify() - Transforms object into a JSON string.

JSON.parse() - Transforms JSON string into an object.

Every property in JavaScript objects can be classified by three factors:

1 Enumerable or non-enumerable;

2 String or [symbol](about:blank);

3 Own property or inherited property from the prototype chain.

**FUNCTIONS**

Functions for the most part should be global, but can have functions inside of functions

Because functions are objects they are stored in the heap.

Function declaration is a statement – opposed to function expression.

Function expressions assign anonymous functions to a variable. (Can assign a name).

bind() – Parameters include – this, arguments from bind created function, also any additional arguments passed from actual function call

Function parameter default values is used if no argument or argument is undefined - (null, ‘’(empty string), and 0 are passed as values and default not used).

Function expressions do not need a name (usually anonymous with just a variable name – sometimes even without a variable name) and are not hoisted fully. Function expressions can be an IIFE, and can be used as an argument in another function (using its variable name). If invoked before initialization and declared with const will get ‘Cannot access function-name before initialization’; if declared with var will get ‘function-name is not a function’

Functions that use function keyword have access to arguments variable (array like object). Arguments object was used before ‘rest’ operator became available in es6. Don't use anymore.

Arrow functions cannot use the arguments keyword - must use …rest.

Lo

**THIS**

‘this’ – keyword contains the reference to owner/top parent/current context.

Class static methods and properties – class does not need instantiation. Can use ‘this’ in class even though it's not instantiated.

this – In a normal function call “use strict” mode = undefined, in sloppy mode points to window (global) object. (In the global execution context this refers to the window object for both strict and sloppy modes?)

In global context this refers to window object – good to know for arrow functions.

**ARRAYS**

Spread operator: use where values are separated by comas – only for building an array or passing values into a function. Can use to make shallow copies of arrays or merge arrays. Works on all iterables. As of ES2018 works on objects also.

Rest operator is used to pack elements into an array. Can use as array parameter value in a function. Rest operator can also be used for objects.

Reduce() Can replace many other array methods.

Ways to Create Array:

arr = [], new Array(), concat(), arr = [...spreadArr], Array.from(), arr.map(a => a+a), newArr = arr.slice() - use slice when need to chain methods, for loop,

Arr = [] – preferred method for creating array.

Methods that modify arrays in place (return reference to original array): splice, sort, reverse.

Arrays much easier to get access to the index with forEach instead of for-of

find() and findIndex() – Finds an object in an array.

IndexOf(), lastindexof() – Finds a primitive value in an array. Can specify a starting location as a second argument.

Need to use fill() to add elements to empty array created by new Array(5).

Flat() – Flattens nested arrays. Flatmap() – combines map and flat methods but flatmap only flattens 1 nested deep.

Weak sets and objects allow for garbage collection.

split() – creates an array from a string separated by a character.

join() – creates a string from an array.

HtmlCollections and nodeLists can use for-of, but only nodeLists can use forEach(). (most modern browsers support forEach() for nodeLists).

Methods that modify arrays (return reference to original array): splice, sort, reverse, fill.

Arrays much easier to get access to the index with forEach instead of for-of

Slice and splice similar but slice does not affect original array.

Slice can be used to copy an array.

Array.from()- converts iterable or array-like object and a map-like callback function.

Iterable vs array-like: Array-like objects have a length and index. Array-like objects: node-list, html-collection, arguments default for a function list and String.

String is array-like and an iterable.

Use find() to compare against properties in an object in an array to get access to entire object in the array. Use findIndex() to get access to just the array index of object.

Iterable: Has Iterable protocol and the @@iterator method = Symbol.iterator. Means that iterables can use for-of loop.

Array like object: Has indexes and a length.

Indexof(), lastindexof() – Can specify a starting location as a second argument.

Push and pop much more efficient than unshift and shift

An iterable is an object that has a next() method that returns parts of a data structure.

Iterable are elements that can be iterated over with for..of. Implements the Symbol.iterator method.

The iterator protocol defines how to produce a sequence of values from an object.

An object becomes an iterator when it implements a next() method.

The next() method must return an object with two properties:

value (the next value)

done (true or false)

If an entity is by default iterable - its prototype will have the method Symbol(Symbol.iterator).

In Array.prototype you will find Symbol(Symbol.iterator): ƒ values() method. The array is by default iterable. Also, String, Map & Set are built-in iterables because their prototype objects all have a Symbol.iterator() method.

The Iterator.prototype object is a hidden global object that all built-in iterators inherit from. It provides a @@iterator method that returns the iterator object itself, making the iterator also iterable.

An iterable is an object that has hat returns parts of a data structure.

Iterable are elements that can be iterated over with for-of loop. Implements the iterator protocol (uses Symbol.iterator method) defines how to produce a sequence of values from an object using a next() method. The next() method must return an object with two properties: 1) value (the next value) 2) done (true or false)

If an entity is by default iterable - its prototype will have the method Symbol(Symbol.iterator).

In Array.prototype you will find Symbol(Symbol.iterator): ƒ values() method. The array is by default iterable. Also, String, Map & Set are built-in iterables because their prototype objects all have a Symbol.iterator() method.

The Iterator.prototype object is a hidden global object that all built-in iterators inherit from. It provides a @@iterator method that returns the iterator object itself, making the iterator also iterable.

Generator function\* returns an object that has a next() method on it.

Iterators are methods that are called on arrays to manipulate elements and return some values.

* forEach() Method name: forEach. Returns: undefined. ...
* map() Method Name: map. ...
* filter() Method name: filter. ...
* find() Method name: find. ...
* reduce() Method name: reduce. ...
* every() Method name: every. ...
* some() Method name: some.

**DOM**

Load - Event is fired when the whole page has loaded, including all dependent resources such as stylesheets, scripts, iframes, and images.

window.addEventListener(‘load’, function(e))- when all images have been downloaded as well as all external resources such as CSS.

DOMContentLoaded – Eventfires when the HTML document has been completely parsed (DOM Tree built), and all deferred and type="module" scripts have downloaded and executed. It doesn't wait for other things like images, subframes, and async scripts to finish loading.

document.addEventListen(‘DOMContentLoaded’, function(e))

Chrome developer network tab also has ‘Finish’ status which includes asynchronously loading (non-blocking) objects/elements on the page after ‘load’.

window.addEventListener(‘beforeunload’, function(e)) – right before unloading page. Some browsers require e.preventDefault()

Event handler function can only take one argument (the event object) if any other data is needed use the this keyword for a value or reference (address pointer).

getAttribute(‘’) – to get actual attributes from HTML such as relative src from img or href from link.

Changing property on input value element does not change the html attribute value so initial default value is not lost. Can change attribute with setAttribute but will not overwrite UI so user input is not overwritten. To overwrite - after setAttribute, assign new value with getAttribute.

<Template> is part of the DOM but not rendered. Document Fragment not part of DOM.

.createElement(‘’)

.innerText = ‘’

.textContent = ‘’

Differences between innerText and textContent. Will usually use innerText (applies css styling). textContent - displays exact text content, spacing, and indentation, no styling. innerText displays text as html would display it. E.g. display:none will hide for .innertext but text will display in .textContent.

.innerText applies css styling. Also displays text as html would display it. E.g. display:none will hide for .innertext but text will display in .textContent.

.innerHTML = ‘’ Can add elements, attributes, and text from a given string. BUT .innerHTML is a SECURITY RISK.

Using innerHTML = innerHTML + ‘string’ reparses and rerenders all innerHTML. If just adding content be careful – innerHTML will replace all nested content – such as input if user entered data.

InsertAdjacentText – Does not create a new element.

.remove()

.removeChild() – Can use remove instead

.getAttribute(‘’) – May not need to use because can access most attribute properties directly

.setAttribute(name, value) –

.removeAttribute()

HTML: data-

Javascript: .dataset

Use replaceWith() to replace element with another or even a clone of itself.

getElementsBy commands - Creates HTML collections.

Use cloneNode(true) to clone all descendants (deep clone) – default is false. cloneNode avaliable on all DOM node objects.

insertAdjacentHTML – Favorite way of creating elements

insertAdjacentElement (old support for adding elements before or after other elements) very similar to insertAdjacentHTML but for created elements.

Appending content with innerHTML will rerender all content. Use insertAdjacentHTML instead. Using createElement (always called on ‘document’.createElement) has benefit of providing direct access to element.

DOM Traversing: parentElement, closest, previousElementSibling, nextElementSibling, firstElementChild, lastElementChild, children (child element nodes – text nodes excluded), querySelector. Element is 2nd word except going down (children).

Input values are special because user inputs data there. Changing property on input value element does not change the html attribute value so initial default value is not lost. Can change attribute with setAttribute but will not overwrite ui so user input is not overwritten. To actually overwrite then after setAttribute can assign new value with getAttribute.

\*HtmlCollections and nodeLists can use for-of, but only nodeLists can use forEach().

document.body => Selects the <body> element node.

document.head => Selects the <head> element node.

document.documentElement => Selects the <html> element node

**SCOPE**

JavaScript uses lexical scoping to resolve the variable names when a function is created inside another function. It determines the function's parent scope by looking at where the function was created instead of where it was invoked.

Hoisting occurs before execution.

Hoisting var, let, const: All are hoisted to the top of scope (global, function or block) before execution, but var is assigned = ‘undefined’. However, let and const are hoisted but not assigned at all, and are in a temporal dead zone until declared. If not initialized with declaration then undefined is assigned.

Both Function declarations (are a statement) and variable declarations are hoisted (Function declarations are hoisted first ) but Function expressions are not hoisted fully.

Function declarations must have a name and are hoisted. Are used as recursive functions (function needs a name to call itself).

Hoisting - making variables and functions available before they are assigned a value or the function is defined. Actually, let and const are hoisted (scope aware of them) but are not available.

{} – Code block

TDZ – Easier to avoid errors.

Scope chain – order in which functions are written (lexical scoping) in code – nothing to do with the call stack or the order in which functions are called

**ASYNCHRONOUS**

**JavaScript Runtime**

**JS ENGINE**

**Call Stack**

**Heap**

**WEB APIs** – Asynchronous Tasks

**DOM Asynchronous Tasks**

**DOM Events**

**AJAX**

**Timers**

**Loading images with JavaScript**

**EVENT LOOP**

**Callback Queue** – Only when call stack is empty & no microtasks.

**Microtask Queue** – Promises. Only when call stack is empty.

Browser - Web API Environment

DOM Asynchronous Tasks

DOM Events

AJAX

Timers

Loading images with JavaScript

Callbacks (regular or promise based (.then)) associated with an asynchronous task are registered with Web API. When task is complete in Web API Environment then any callback functions associated with a completed task is loaded in either the callback or microtask queue.

Note: setTimeout() callback timing is not guaranteed. If there are other tasks ahead of it in callback queue or microtask queue.

Promisifying – Convert old async based callback code to promise based by wrapping in a promise.

then() - always returns a promise. If a value (any value) is returned from then – it becomes the fulfilled value.

The only rejection from a fetch promise is if user loses internet connection, not a 404 error.

Two ways of handling promise rejections: 1) Pass a second callback function into the then method callback. 2) Handle with .catch.

Errors will propagate down to .catch.

Throwing an error automatically set promise to rejected and terminated the then handler.

Finally – accepts a promise either from then or catch.

Asynchronous code is non-blocking. Asynchronous code is executed after a task in the background finishes.

Promise – An object used as a placeholder for the future result of an asynchronous operation. 1) Pending, 2) Result = Resolved or Rejected.

Fetch API – Returns a promise.

MicroTasks Queue (Promises) have priority over Callback Queue.

Closure: Variable environment of the execution context is closed (parent scope (variable environment)) to the function where it was created.

Undefined == null. Undefined is a value and type. Null is a value with a typeof object.

An undefined variable does not exist or was declared but value not assigned.

Can use getters and setters good for extra validation, transformations, assigning defaults, read only.

Accessing a property that does not exist on an object will not return an error but will just be undefined.

Object methods are function expressions (function values), not function declarations.

Javascript objects and functionality is based upon prototypical inheritance

Javascript styles are applied as inline styles.

Because text nodes cannot have children, only element nodes can, parentElement and parentNode are the same except for on document.documentElement.

Just like let and const variables, classes are hoisted to the top of the scope they are defined in, but inaccessible until they are initialized (temporal dead zone).Temporal (TDZ)

A variable declared with let, const, or class is said to be in a "temporal dead zone" (TDZ) from the start of the block until code execution reaches the line where the variable is declared (with or without initialization value). When declaration occurs, initialization also occurs – if no value specified then it is initialized with ‘undefined’.

While inside the TDZ, the variable has not been initialized with a value (undefined or otherwise), and any attempt to access it will result in a Reference-Error. The variable is initialized with a value when execution reaches the line of code where it was declared. At declaration time, if no initial value was specified with the variable declaration, it will be initialized with a value of undefined.

This differs from var variables, which will return a value of undefined if they are accessed before they are declared.

Some prefer to see let, const, and class as non-hoisting, because the temporal dead zone strictly forbids any use of the variable before its declaration (reference error).

Event Listeners / Loop – Managed by the browser not JavaScript. The browser informs the javascript engine when an event occurs.

CSS - white-space: pre

Let and Const – Block scope not function scope – ‘curly braces’ – if statements, loops, functions (not curly braces for objects though (right side of equal sign)). Can also use a stand alone block with just curly braces.

Can actually redefine var variables without an error.

For-in loop to loop thru keys in an object.

Symbols are a primitive values (can be used as an object property key. There are built in symbols and can create your own. Uniqueness is guaranteed. Symbol can be accessed as a method or a property.

Symbols are used to create object properties, for example, when you want to assign a unique identifier to an object. They can also be used as a way to create private properties in objects, as they are not enumerable. In addition, Symbols can be used to create unique constants, which are useful when creating APIs.

!! – Converts truthy/falsey to real boolean

Template literal `${}`

Static properties can only be accessed within the class they are defined in – not accessible from instance.

Can use ternary expression in function parameters as default values.

Can use numbers as object key names and access with bracket notation with or without quotes.

document.addEventListen(‘DOMContentLoaded’, function(e)) – Triggered when all HTML is downloaded and parsed (DOM tree built) and all scripts downloaded and executed. Does not wait for images or other external resources.

load – page fully loaded.

window.addEventListener(‘load’, function(e))- when all images have been downloaded as well as all external resources such as CSS.

Chrome developer network tab also has ‘Finish’ status which includes asynchronously loading (non-blocking) objects/elements on the page after ‘load’.

window.addEventListener(‘beforeunload’, function(e)) – right before unloading page. Some browsers require e.preventDefault()

Event handler function can only take one argument (the event object) if any other data is needed use the this keyword for a value or reference (address pointer).

Promisifying- Wrapping an old asynchronous based callback function into a promise based function. The conversion of a function that accepts a callback into a function that returns a promise.

Building a promise using the promise constructor. Promises are just a special type of object. Promise constructor takes an argument which is a executor function. Executor function takes two arguments- resolve and reject.

new Promise(function(resolve, reject) {}),

Executor function within promise constructor handles the asynchronous behavior. So the executor function will return the resulting (future) value of the promise.

resolve function: Sets the promise as fulfilled. Takes the fulfilled value, to be returned to the then method, as an argument.

The fetch command performs an AJAX call with a promise. Callbacks of promises go into the microtasks queue not the callback queue. Microtask queue has priority over the callback queue. If there is more than one microtask in the queue then all of them will be executed before anything in callback queue. Callbacks from promises are called microtasks. However, call stack must be empty before loading callbacks from either. Event loop orchestrates loading callbacks and microtasks into call stack.

When Promises are settled they are either resolved or rejected.

Calling JSON method on a resolved promise. JSON is an asynchronous function that returns a promise.

document.documentElement.style.setProperty(‘—color-primary’, ‘orangered’) Can also set other styles like this but it is longer. (css root: equivalent)

document.documentElement – retrieves document root element - <html> element

Object Constructor functions simulate classes where the name of the Constructor function simulates the name of an actual class. Objects created with the Constructor function become instances of the Constructor function (instanceof)

leftoff

Element Event Phases – 1) Capturing 2) Target 3) Bubbling

Event Bubbling: when an element receives an event, and that event bubbles up (or you can say is transmitted or propagated) to its parent and ancestor elements in the DOM tree until it gets to the root element.

Closure – Variable environment of execution context where created. Keeps references to outer scopes even after they have left call stack.

Closure has priority over scope chain.

Object.assign – To copy objects or merge objects. Spread operator preferred because it is shorter.

Ck for even numbers - Divide by 2 use remainder %.

ToFixed – Rounds and/or pads zeros then coverts number to a string.

Floor instead of trunc to round down because it accounts for negative numbers.

IsFinite – checks valid floating point number.

IsInteger - checks valid integer number.

ParseFloat – converts string with trailing letters to number. Use this instead of parseInt because it allows for decimals.

Reduce() Can replace many other array methods.

Array.from() – will create and programmatically populate new array.

Need to use fill() to add elements to empty array created by new Array(5).

Can use slice() to copy array instead of …spread when functions need to be chained.

Flat() – Flattens nested arrays. Flatmap() – combines map and flat methods but flatmap only flattens 1 nested deep.

Weak sets and objects allow for garbage collection.

Find and findindex methods have callback functions. Find returns whole element.

‘includes()’ method checks for equal condition only. ‘some()’ method allows for more flexible conditions – returns true or false. ‘every()’ method checks for every condition true.

Object elements are ordered by order in which they are added. Except number keys which are sorted.

Array.from() – Will convert an array-like or iterable element to an array – such as a node list or collection.

Iterable - Any element that you can use ‘for of’ on.

Append vs appendChild- append – can add text nodes, also can add multiple nodes at once.

.append() – Can add strings, text nodes, elements and multiple items

.appendChild() – Can only add elements not strings

Closest – uses css selectors like querySelector, but searches up instead of down.

Only ‘element’ nodes can have children ‘text’ nodes cannot

Attributes vs Properties – Attributes are in html, Properties are mapping of Attributes onto DOM Object.

Can use all valid css selectors - ex. pseudo selectors in querySelector

Can use ‘for of loop’ with coverted to array html collections “getby” methods

Window object points to tab not whole window

Functions for the most part should be global, but can have functions inside of functions

Default ‘arguments’ for functions defined with function keyword is array like – used before es6. But should use rest operator (es6).

Rest operator – function parameters- creates array. Also don't need to know number of parameters.

Throw, try, catch, final.

!! – Forces boolean if truthy or falsey value

Stringing ternary operator

Passing undefined argument function will use default parameter (not the case for other falsey values).

Labeled statement - break or continue out of outer loop.

Shift/ enter for newline in console for writing code.

Highlight source code variable to get value.

Shadowed variables – same name different scope.

Semicolon not used after functions.

Implementing login 3:00

211 9:00 Prototypal inherentance on built in objects

Spread operator only valid for function arguments and building arrays.

Emojis – windows key + .

\n\ – newline (can use return key in tics)

performance. now()

Jsperf.com – Bechmark / compare different js code approaches.

Closure has priority over scope chain.

Methods on primitives js converts (boxes) primitives to objects then runs method then coverts back to primitive.

######### Javascript Notes2

Web API - Environment of the Browser – DOM methods, asynchronous tasks timers, AJAX Calls,

Promise States & "finally"

PENDING => Promise is doing work, neither then() nor catch() executes at this moment

RESOLVED => Promise is resolved => then() executes

REJECTED => Promise was rejected => catch() executes

When you have another then() block after a catch() or then() block, the promise re-enters PENDING mode (keep in mind: then() and catch() always return a new promise - either not resolving to anything (if nothing is explicitly returned then the returned value is undefined) or resolving to what you return inside of then()). Only if there are no more then() blocks left, it enters a new, final mode: SETTLED.

Once SETTLED, you can use a special block - finally() - to do final cleanup work. finally() is reached no matter if you resolved or rejected before.

You don't have to add a finally() block (indeed we haven't in the lectures).

Can call the .then method on all promises.

.then method executes a callback function when promise result is available. JavaScript supplies the argument into the callback function which is the result of the promise (i.e. for a fetch it wold be the response from an AJAX call).

json() is method on all response objects from the fetch method. It is also an async function that returns a promise. The resolved promise is the data itself.

The .then method returns a promise regardless. If a value is returned then that value becomes the fulfilled value of the promise.

The input value (callback argument) of the .then method will be the fulfilled value of the prior promise.

Fulfilled value from a fetch promise can be called resp(onse).

Throwing an error inside a then method causes promise to be rejected .

async await – changes consumption of promises but not the creation of Promises (eliminates .then). Async function runs in the background. Enables asynchronous, promise-based behavior to be written in a cleaner style and avoiding the need to explicitly configure promise chains - (.then and callbacks not needed).

An async function essentially wraps its contents into a promise.

An async function always returns a promise which will later contain the value of the fulfilled promise.

i.e. const city = whereAmI() - city will contain promise object / not resolved promise

whereAmI().then(city => {}) - will hold fulfilled value of promise

Await immediately returns a promise and halts execution of code within async function until await promise is fulfilled or rejected – await then becomes the value of the fulfilled or rejected promise which can be stored in a variable. Gives the appearance of synchronous code (asynchronously). Need to handle errors with try/catch statement (consisting of try and catch (blocks)) because a .catch (method) cannot be attached to anything like .then/catch allows for. (not specific to just asynchronous code).

Async await is syntactic sugar for the then method (consuming promises).

Errors in async function will still return a fulfilled promise. must Re-Throw error

Dont need an await keyword n async function can just return a promise and use a .then method.

IIFE – One of the last reasons for using iife is for async function.

Use hasownproperty instead of forin (forin looks up prototype chain)

Use forof keys or entries to get values

Html tags: abbr, meter, progress, kbd, details/summary, samp

Html tags: abbr, meter, progres

Re-education to my first love developing.

Would like to bring my newly developed skills along with my extensive it? Experience to help

Legacy codebaae

API, Form, Objects / Classes, Array methods, private data

Life Motivation, Quote of the day, goals, objectives, tasks, Quote builder, Leaders, Quotes, Age, Education, Vocation

Just in time learning

State based UI

Truthy: A value considered true when evaluated (coerced) in a boolean context.

All values are truthy except: false, undefined, null, NaN, “”, 0, -0, 0n.

Nullish types are undefined and null.

This: The reference to the owner object of the execution context. (What is executing the code).

Arrow function this value is lexically based.

typeof

Let and const are scoped

Cannot reassign const

Easy to understand definitions:

This keyword

Hoisting

Lexical

Context

String = Number + “” - converts number to string

Iifee - <es6?

Use strict

First class functions

High order functions

Currying

Self invoked- recursive

Closure has priority over scope chain which has priority over prototype chain.

Memorization

Monad: A way of structuring operations in a sequential (chained) manner for related data.

Reusable code that enables chaining operations instead of nesting.

Remember, a monad is really nothing more than a chainable computation. It is simply a functional way to sequence things.

Reusable code that enables related operations to be performed in a sequential (chained) manner instead of nesting.

JavaScript Promises implement the monadic interface with . then() method.

Truthy falsey in boolean context

Function machine? Generator?

Advantage of promises:

1. Don't need to rely on events and callbacks
2. Can chain for a sequence of asynchronous operations instead of nesting with callbacks
3. In [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript), a **truthy** value is a value that is considered true when encountered in a [Boolean](https://developer.mozilla.org/en-US/docs/Glossary/Boolean) context. All values are truthy unless they are defined as [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy). That is, all values are truthy except false, 0, -0, 0n, "", null, undefined, and NaN.
4. [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) uses [type coercion](https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion) in Boolean contexts.
5. Examples of truthy values in JavaScript (which will be coerced to true in boolean contexts, and thus execute the if block):
6. JSCopy to Clipboard
7. if (true)
8. if ({})
9. if ([])
10. if (42)
11. if ("0")
12. if ("false")
13. if (new Date())

The values null and undefined are also [nullish](https://developer.mozilla.org/en-US/docs/Glossary/Nullish).

The only falsy object in JavaScript is the built-in [document.all](https://developer.mozilla.org/en-US/docs/Web/API/Document/all)

JavaScript **this** keyword always holds the reference to a single object, which defines the current line of code’s execution context which means this keyword refers to the object that is currently executing the code. Functions in JavaScript, are essentially objects.

### What this returns depends on where it has been used?

Let us look at different positions where **this keyword**can be called.

* **this alone (window)**
* **this in function (calling object or window)**
* **this in strict mode (calling object or undefined)**
* **this in event handlers (html element)**

### The precedence order of **this** keyword is:

* [**JavaScript bind() Method**](https://www.geeksforgeeks.org/javascript-function-prototype-bind-method/amp/)
* [**JavaScript call() and apply() Method**](https://www.geeksforgeeks.org/explain-call-and-apply-methods-in-javascript/amp/)
* [**JavaScript Object Method**](https://www.geeksforgeeks.org/javascript-object-methods/amp/)
* **[JavaScript Global Scope](https://www.geeksforgeeks.org/understanding-variable-scopes-in-javascript/amp/" \l "_blank)**

**In simple terms, we can define looping or iteration as the process where the same set of instructions is repeated multiple times in a single call. In contrast, we can enumerate recursion as the process where the output of one iteration from a function call becomes the input of the next in a separate function call.**

const myDate = new Date();

let object = myDate;

do {

object = Object.getPrototypeOf(object);

console.log(object);

} while (object);

// Date.prototype

// Object { }

// null

\*Shift/enter in browser console allows for adding multi line code.