JavaScript

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **Cues**  NoteGem Horizontal Line      what are the syntactical parts of a function?            what are arguments?      what matters most for parameters?    What is everything after the return statement?      What do the two passes of the JS compiler do?          What is hoisting?    When are function declarations invoked?      What is function expression?              what are arrow functions?        when are they good to use?  When should you not use them?              What are the three variable type differences?    rule of thumb?      what are object literals?      what is a property?          what is dot and bracket notation?        what are three methods for iterating over keys?        what's an array?      what's .push?  what's .unshift?  what's .pop?  what's length?  what's splice?  what's clone  how to get last item?  how to remove last item?  how to add item to end?    how to overwrite at specific index?      how to remove several items?  how to add item at specific index?  how to remove item from specific index?  how to reverse array?  how to delimit array?  how to sort alphabetically?  how to sort numerically?  how to join two arrays?  how to copy specific items?  how to augment items with array?  what's a method?  what is closure?        what does it need  functions and scope?        global scope?  functions inherit in which direction?      when new scope created      how does a variable belong to a function      where do functions seek context?          JS is what two kinds of programming      what's a higher-order function  what's a callback function        how are functions passed  how many arguments can they have                      what is forEach  what does it return                  what is map  what does it return  what is filter  what does it return  what is reduce  what does it return      with things this encompases, what are they? Describe each one                                                                                            constructor function and objects      what is prototype                                        how to call parent-child function                      times/four reasons not to use arrow functions      what are classes  how do you work with classes          what is a constructor function        what do classes return  how to pass properties to your created object      what does extends do  what does super do  how to use extends and super                      what does a returned object from class have  where are its methods        what does extends extend        when must you call super and what does that do                what are template literals        what's a dom      tree structure what does it do            when webpage loads, what happens  why a tree                  how to get desired leaf    what are dom selectors      what's a get element method  what does it return                  getElementById            getElementByClassName                  What is query selector                  how can you select  format and syntax          nodelist/html collection  what methods does it have access to                how to access properties contained in an element  what does the following do  how is it used  text content          set attribute                style                      .classname  .id              .classlist                .children/parent node                .appendChild    .prepend          what does js allow us to do        what's an event                  what is addEventListener  what does it do              callback and event object  most important property of event object and why                                what is preventDefault()  what does it do                    what are components  what are they made of  what do you use it for                    common practice for LESS/CSS        what can you use JS for                                what happens if promise succeeds or fails                                                          what is http        what do http methods do      how to ask a server for info  what's a post request  put      delete  http status codes              what is axios  why use it  how does it work  axios how to get | **Notes**  NoteGem Horizontal Line    function declarations have several syntactical parts:     * function keyword * name of function * optional list of parameters * statements inside block of code     **arguments** = values you pass into a function     * then received into parameters of function * what matters most for parameters is order received     anything written after a return statement will not execute because function terminates after return execution    JS utilizes a two-pass compiler when executing:     * first pass sets up references to all the code * second pass applies values to found rcs     function declarations defined    **hoisting** = compiler made aware of function declaration and adds it to top of execution order    function declarations can be invoked before defined    **function expression** = variable used to store function for later use     * not hoisted * anonymous functions are used     ()=>{\_\_\_\_\_} ()=>\_\_\_\_\_    arrow functions = functions with 'this' keyword removed     * if there is only one return statement, no need to use it (it's implied)   + {} also redundant   + great to use with arrays   + stick with simple conversion first and go to more complicated ones as you progress   + do not use for:     - event handlers     - object methods     - prototype methods     - anytime you need to use arguments object     var can be reassigned and overridden (function scoped)  let can be reassigned, not overridden (block scoped)  const cannot be reassigned or overridden (block scoped)    **rule of thumb** = use const until you can't, then use let    **object literals** = way to store data and give access to it when necessary    **property** = ability to store data and call it     * organized in a key-value pairing     **dot notation** = myObject.firstName, objectname.property  **bracket notation** = myObject["firstName"], objectname["property"]    methods for iterating over keys:     * object.keys (returns property names) * object.values (returns property values) * object.entries (returns both)     **arrays** = indexing starts at 0     * index based and ordered (objects are not) * object[#] = # = index number you want to access     .push() = **adds item to array**  .unshift() =**adds item to front of array**  .pop() = **removes last item in array**  array.length = **length of an array, slots in array**  array.splice = **(0, array.length)**  **clone an array** = array.slice();  **get last item** = array[array.length-1];  **remove first item** = array.shift();  **add item to end** = array.push();  **overwrite item at specific index** = array.splice(x,y,z);     * array[#] = \_\_\_\_     **remove several items** = array.splice(x,y);  **add item at specific index** = (x,y,item);  **remove item from specific index** = array.splice(x,y);  **reverse array** = array.reverse();  **delimit array** = array.join('AND');  **sort in alphabetical order** = array.sort();  **sort in numerical order** = array.sort(function(a,b) {return a-b;});  **join two arrays** = array1.concat(array2);  **copy specific items** = array.slice(x,y);  **augment items within array** =  **methods** = function inside object or belonging to function  **closure** = code identified elsewhere that we can use later  closure must be discussed through lens of     * what data do I currently have access to in my program * gives ability to put functions together * need to be able to access data within narrower scope * functions create their own scope     window/console in browsers = global scope  two types of scope = global + local  can inherit down and not up    new scope created when function declared and created     * any variables declared within that function's scope will be enclosed in scope belonging to function     functions look outward for context     * if variable isn't defined in function scope, function will look outward for variable     javascript is both object oriented and functional programming     * get the ability to choose patterns to use to accomplish tasks     **higher-order function** can receive other functions as parameters  **callback functions** = functions passed in to other functions as args    functions in js are first-class citizens, a type like all others     * can be passed around as arguments to other functions * we can pass as many parameters as we want     **subtract** = -  **divide** = /  **multiply** = \*  functioncb(#1, #2, cb) {return cb(#1, #2);}  examples of callbacks     * NEED EXAMPLES OF CALLBACKS     **forEach** = reiterates over every array item     * passes back item & index to use * does not return like .map does     functional programming lends itself to idea of immutability     * map, filter, reduce used to achieve immutability     **map** returns:     * current item of array state * current index of current item index * entire array data     **this keyword** = big interview question     * pointer to an object * can reference object without referring to its name * how to figure out what this is pointing to     **window/global object binding**     * 'use strict'; = strict mode, prevents window binding * this = window/console object     **implicit binding**     * whenever function is called by dot preceding * can apply to objects with methods * this = object before     **new binding**     * points to new object created * when constructor function used: * missing return * capitalized function name * this = specific instance of object created and returned by constructor function * constructor = function that returns object      * object creator use new keyword when calling it * builds objects     **explicit binding**     * whenever call or apply methods used       all objects in javascript have a prototype property by default     * prototype used as object to attach methods and other properties from children     **call** passes arguments one by one, immediately invokes  **apply** passes arguments as array, immediately invokes  **bind** passes args one by one, does not immediately invoke     * returns brand new function we can invoke later     constructor function needs to take in object literal so it can map object properties to new returned object    **prototype** = mechanism by which all javascript objects inherit from one another     * an object that objects use to hold onto values which can be passed down to other objects     function Person(attributes) {  this.age = attributes.age;  this.name = attributes.name;  this.hometown = attributes.hometown;  this.speak=function(){return `Hello, my name is ${this.name}`;  }; //close speak  } // function close    function Child (childAttributes){  Person.call(this, childAttributes); // bind to Person  this.isChild = childAttributes.isChild; //special attributes to child  } // function close    child.prototype = Object.create(Person.prototype);     * tells Child about Person     arrays can take everything functions, objects can do  functions can do everything objects can do  everything in javascript is an object    this is **prototyped inheritance**    **when not to use arrow functions**     * click handlers * object methods * prototype methods * when you need arguments object     **classes** - not hoisted 'use strict'     * syntactic sugar on top of object built into javascript and object's prototype system * special functions     two ways to work with classes     * class expressions * class declarations     **constructor function** = foundation of every class     * when present, object properties go inside with 'this'     classes will return **objects**  any attributes you want bound to your created object as properties can be passed through constructor    **inheritance with extends super**     * extends will abstract away any classical syntax we're used to * super tells parent to be concerned with child * abstracts away object.create syntax attributes     can put methods in Object/class's prototype    class Animal{constructor(name){this.name=name;}speak(){console.log(this.name + ' makes a noise.');}}  const doggy = new Dog('Grizzly');  doggy.speak();    object returned from class will have its attributes assigned to it and returned out of constructor function  all methods attached to class stored on object's prototype in a special way    **extends** = extends a parent object     * look for these as a clue for class --> subclass     if you use extends, you must call super() in constructor     * passes any new attributes back up to constructor parent     **strict mode** = prevents window binding but also returns undefined     * 'use strict';     **template literals** = text between backticks     * can be multiple lines or one line * `string goes here with variable value ${name}.`     **dom** = document object model  object representation of html elements of web page  gives us interface which we can interact and manipulate page with  tree structure with each dom element being a tree node     * some node properties = informational * some = methods we can use for interaction * each tree node contains same property keys as each other node     creates and propagates event objects  when webpage loads     * looks for html file * uses html as blueprint on how to build page * parses instructions and builds model for how page should look and act using javascript      * this model = javascript object with every element in order on page     dom is built as a tree because parent elements have nested child elements or leaves  follow branches to get to leaf you want to access  each branch can be its own tree  **dom selectors** = select element you want to manipulate  case sensitive    **getElement** methods     * accepts single string with id or class you want      * document.getElementsByTagName('p');      * returns array-like object containing all elements that contain element name supplied      * document.getElementById('idName');      * takes and returns single string with ID of element as argument      * document.getElementsByClassName('className');      * takes single string as argument containing class of desired elements * returns array-like object htmlCollection containing all class's elements     **querySelector** methods     * allows you to select elements based on css style selectors * takes string containing selectors and returns the elements * can select by element, id, class, or others with both methods * document.querySelector('.custom-style');      * this will search for and return first element that matches value passed into method      * document.querySelectorAll('queryString');      * this will search for and return all elements matching string * returns array-like object called nodeList     **htmlcollection & nodelist** = array-like objects     * both have nuymerical-based indices and length property * nodelist has access to forEach * no other array method * can use array from (array-like object)     after capturing an element we use that instance of it to access and assign values to properties natively contained on it    most commonly used ones     * .textContent   + gets and sets text of element, whatever text between element tags   + can use to reset text   + element.textContent = 'something new';      * .setAttribute() or .{attr}      * used as a way to reassign or set attribute as object * takes two arguments attribute to set value to set it to * element.attrName = 'value' OR * element.setAttribute(src', '<https://aquoco.co>')      * .style      * every element has a style object and this accesses it * contains every available style as key and a value as value * not css styles but inline html styles set one element * can access and change property on style object by using   + element.style.color = 'blue' * inline styles have highest specificity   + overriding every style but !important      * .className, .id      * .className accesses string containing all of that class's elements * .id = same, but for IDs      * .classList      * returns array-like object of all classes on the element * is a DomTokenList, an array-like object with numerical zero-based index, length, property, contains, and forEach * add, remove, toggle methods exist * all take single string representing class      * .children & .parentNode      * used for accessing relatives of element * children returns HTMLCollection of all children of element * parentNode returns parent of that element      * .appendChild() & prepend()      * used for adding children to parent elements * appendChild(child) will add an element to its children   + will add to end physically   + ordered last * parentElement.appendChild(childElement) * .prepend(child) adds child to beginning, displays first     javascript allows us to add features or make modifications to site by directly reacting to user interactions    **event** = every interaction a user has with a site     * clicking, moving mouse, scrolling page, keypress     **trigger** = when event happens on a page     * documentation lists all events possible     **.addEventListener** =     * method to use on selected element   + event to listen for   + callback to fire when event is triggered * takes two arguments     **callback and event object**     * callback will take single argument, event object * this is a javascript object and contains all we need to know about event and element it was triggered on * element.addEventListener('click', (event)=>{//handle event}); * one of most important properties of event object is .target      * gives us all info about dom node the event was triggered on * has many of the same properties as regular dom node * can use to manipulate element itself, target in any way we want * example {event.target.style.backgroundColor='blue';}instead of {//handle event} * depending on event type, can have access to information about event such as key pressed, etc.     **preventDefault()**     * some elements have native default reaction to certain events * example form elements refresh page on submit * method on event object that has lots of methods/props on it      * example stopPropagation() will stop our event from bubbling further up the chain     will stop html element from reacting in its default way  components     * reusable pieces of code that can be used to build elements sharing functionality and styling * heart of any dynamic web app and javascript framework * building block to modern app development * made up of several parts * should be modular or standalone      * think of it as if it could be moved around at any moment     use to help control styles = use specificity chain that only matches up with your component  common practice to name preprocessed file after component name     * then import name into main file     javascript used to consume data and output content to dom  glue that ties everything together  can use javascript to consume html    javascript is core component in repeatable nature of components     * using function and createElement can create unique components and add those to DOM     **asynchronous code** = code that doesn't run instantly inline     * example code waiting for something to happen before firing     **promises** = design pattern for use when handling asynchronous code in javascript     * alternative to nesting multiple callbacks * a way to avoid callback hell problem * won't do very often in front-end development     only two methods to handle promises and then catch     * "Promise Library" * proxy for value not necessarily known when promise created * allows you to associate handlers with asynchronous action's eventual success or failure * can return values like synchronous methods instead of immediately returning final value * returns promise to supply the value in the future * promise from the object that it will let us know when it has completed what we have asked it to do      * can exist in one of three states:      * pending = promise not rejected or fulfilled (first state) * fulfilled = all's well & resolved; value can be used by code * rejected = something went wrong and error needs to be dealt with     if promise succeeds, it will return value as parameter into callback passed into .then()  if it fails the callback passed into.catch() runs, taking error as an argument  **http** = network protocol, set of rules that govern the way web clients like browsers communicate over the internet with web servers  http methods provide common language and nomenclature that client can use to let server know what operation it wants to perform  when a client needs to ask a server for info it should do a get request, specifying a URL that points to the desired resource  post request = used to ask server to create/add new resources  put request = used to ask server to make changes to specific resources  delete request = used to request to remove/delete data from server  http status codes = indicate if request has been successful or not and why     * server sets status code for all responses sent to client     **axios** = js library used to send http requests to server     * not necessary * makes things easier * uses promises because all server requests are asynchronous     github.com/axios/axiosn/    <script src="https://www.unpkg.com/axios/dist/axios.min.js"></script>    axios.get(url) = will return promise to us     * will use .then and .catch to deal with returned data | |
| |  | | --- | | **Summary** | |