|  |  |
| --- | --- |
|  |  |
| 1 | Abstractions |
|  |  |
| 2 | Encapsulation |
|  |  |
| 3 | Polymorphism |
|  |  |
| 4 | Inheritance |
|  |  |
| 5 | Prototypal Inheritance |
|  |  |
| 6 | Interfaces |
|  |  |
| 7 | Composition |
|  |  |
| 8 | Object |
|  | Objects are collection or group of variables in key value pair |
| 9 | Object Literal |
|  | const obj { Brand: “Renault”, Model=”Duster”} |
| 10 | Object Literal in ES6 |
|  | in ES6 if key and values are same no need to repeat values. like  const obj = { name: name, age: age} can be written as const obj = {name, age} |
| 11 | new keyword |
|  | new keyword is used to create new object, array or any other data. Like  const myCar = new Object( ) or const myCar = new Car( ) |
| 12 | Object methods |
|  | Properties in object those are functions are called methods |
| 13 ? | Constructor |
|  | const Car = function(make, model, color){  this.make = make;  this.model = model;  this.color = color;  } |
| 14 | Primitive & Reference data Types |
|  | Array and objects are reference data type while others are primitive |
| 15 | Undefined vs Undeclared |
|  | undefined means variable has initialized but value is not decclared |
| 16 ? | Hoisting |
|  |  |
| 17 | Scope |
|  | Scope defines availability or accessibility of variables. Javascript has global and local scope. |
| 18 | Nested Scope |
|  |  |
| 19 | Lexical Scope |
|  | Lexical scope is the ability of inner function to access the outer scope in which it is defined. Functions have access to scope in which it was created not the scope in which it was executed. Local scope of parent function is lexical scope of inner function |
| 20 | Dynamic Scope |
|  | In Dynamic scope it finds variable in scope where it was executed not where it was created. Javascript has lexical scoping not dynamic scoping but ‘this’ works like dynamic scoping in JS |
| 21 | Scope Chain |
|  | Every scope has a link to the parent scope. When a variable is used, javascript looks up the scope chain until it either finds the requested variable or until it reaches the global scope, which is the end of scope chain |
| 22 | JS Context |
|  | Scope is related to variables while context is related to object. It refers to the object within the function is being executed. |
| 23 | this keyword |
|  |  |
| 24 | call.bind() call.apply() |
|  |  |
| 25 | Implicit & explicit Binding |
|  |  |
| 26 | Event Delegation/bubbling/propagation |
|  | In this event starts from target element to its parent elements upto document level. This can be stoped using event.stopPropagation() |
| 27 | Event Capturing |
|  | In this event starts from top element to target element from top to bottom |
| 28 | Event Listening |
|  | element.addEventListener("click", function(){ alert("Hello World!"); }); |
| 29 | array |
|  | [a, b, c] |
| 30 | Typed Arrays |
|  | JavaScript typed arrays are array-like objects and provide a mechanism for accessing raw binary data. |
| 32 | functions |
|  | function xyz ( ) { } |
| 33 | Function Declaration |
|  | function abc() {…………….} this is function statement or function declaration |
| 34 | Function expression |
|  | var abc = function() {………………….} this is function expression. These are useful in closures, as an argument to other functions or as IIFE. Function declaration load before any code is executed while function expression load only when interpreter reaches that line of code |
| 35 ? | Anonymous Function |
|  | functions those are created without assigning any name like function( ) { } |
| 36 | Black Box Functions |
|  |  |
| 37 | Private Functions |
|  |  |
| 38 | Factory Function |
|  |  |
| 39 | Self invoking Functions |
|  | function ( ){ }(); function those are executed right after declaration |
| 40 | Iterator Functions |
|  |  |
| 41 | IIFE |
|  | Immediately invoked function expression or self invoking functions |
| 42 | Closure |
|  |  |
| 43 | Pure Functions |
|  | Pure functions are the functions those given the same input, will always return the same output. |
| 44 | Higher Order Functions |
|  | Higher order functions are functions those either receives a function as an argument or return the functions as an output. map(), filter(), reduce() are higher order functions |
| 45 | Static methods in functions |
|  |  |
| 46 | Method Overloading |
|  | Method or function overloading is have multiple functions having same name but different no. of parameters. Javascript doesn’t support method overloading. It consider method that is defined at last |
| 47 | Method Overriding |
|  | Javascript supports overriding not overloading. When you define multiple functions which have the same name. the last one defined will override the previously defined ones |
| 49 | Arrow Functions |
|  | const abc = ( ) => { } |
| 50 | De-structuring |
|  | const { name, age } = this.props |
| 51 | Classes |
|  | ES-6 classes |
| 52 | Modules |
|  | export default abc; import abc from ‘./abc.js’ |
| 53 | Template Literal |
|  | `some content ${variable} other content` |
| 54 | Let and const |
|  | let and cons |
| 55 | Spread Operator |
|  |  |
| 56 | Rest parameter |
|  |  |
| 57 | Default Parameter |
|  | assigning default value of function parameters if anything has not been passed while invoking |
| 58 | Symbols |
|  |  |
| 59 | Generators |
|  |  |
| 60 | Async await |
|  | Async Await |
| 61 | Promises |
|  | Promises |
| 62 | Getter and Setter |
|  |  |
| 64 | Double & triple equal |
|  | == === |
| 65 | instance of operator |
|  | The **instanceof operator** in JavaScript is used to check the type of an object at run time. It returns a boolean valueif true then it indicates that the object is an instance of a particular class and if false then it is not*.* |
| 66 | try catch |
|  |  |
| 67 | throw statement |
|  |  |
| 68 | finally statement |
|  | The **finally** statement lets you execute code, after try and catch, regardless of the result.  try { }  catch( err ) { }  finally { } |
| 69 | console object ( .log, .clear, .table, .time etc) |
|  |  |
| 70 | Escaping characters |
|  | \ |
| 71 | Window events |
|  |  |
| 73 | Cross site scripting(XSS) |
|  |  |
| 74 | vulnarabilities |
|  |  |
| 75 | CORS |
|  |  |
| 76 | Cross Site Request Forgecy (CSRF) |
|  |  |
| 77 | Script Async/Defer |
|  |  |
| 78 | Prefetch |
|  | If we're certain that a specific resource will be required in the future, then we can ask the browser to request that item and store it in the cache for reference later. For example an image or a script, or really anything that's cacheable by the browser:  <link rel=”prefetch” href=”image.png” /> |
| 79 | Cookie |
|  |  |
| 80 | Local Storage |
|  |  |
| 81 | Strict Mode |
|  |  |
| 82 | Observables |
|  |  |
| 83 | Side Effects |
|  |  |
| 84 | Currying |
|  | Currying is the process of transforming a function with multiple arity(arguments) into the same function with less arity. It transform a function into a sequence of functions each taking a single argument of the function. |
| 85 | Recursion |
|  | Recursion is a technique for iterating over an operation by having a function call itself repeatedly until it arrives at a result. |
| 87 | Functors |
|  | Functors are any mapable objects those can be mapped and apply a function generating another object of same time. Functors can be an array, object, function or string |
| 88 | Monalds |
|  | A monald is way of composing functions that require context in addition to the return Value |
| 89 | Transducers |
|  | Transducer is a composable higher order reducer. It takes a reducer as an input and returns another reducer |
| 90 | Lenses |
|  | A lens is a composable pair of pure getter and setter functions which focus on a particular field inside an object and obey a set of axioms known as the lens laws |
| 91 | enum |
|  |  |
| 92 | Data normalization |
|  | Data Normalization is the process of structuring a relational database to reduce data redundancy and improve data integrity. For example if some values are repeating again and again in data we can restructure that to reuse those values. another example is long array should be converted into object having some unique identification like id or label etc to decrease data search time while finding some data. |
| 93 | Reactive Programming |
|  | Reactive Programming is programming with Asynchronous Data Streams. In this we create data streams of anything and from anything like click events, http requests, messages, notifications, changes of variables, cache events, literally anything that may change or happen. We observe these streams and react when a value is emitted. Rxjs is used for reactive programming is javascript. |
| 94 | The Temporal Dead Zone |
|  | Temporal dead zone is where we are trying to access a variable that has not been initialized/defined |
| 97 | map function |
|  | map functions accepts an callback functions and run it on each item of array resulting each new value in resulting array. Resulting array would always be the same length as the original array  books.map( (book) => book.author ) |
| 98 | filter function |
|  | Filter function runs a callback function in an array and execute some conditions and create new array having only those value those passes that condition |
| 99 | reduce function |
|  | reducer function works like map function and runs a callback function on each item of array but difference is that it passes the result of this callback( the accumator) from one array element to other  The accumator can be anything (integet, string, object etc)  var totalYears = pilots.reduce( function(acc, pilot) {  return acc + pilot.years  }, 0 )  0 is initial value of accumator |
| 100 | Shallow copy & Deep Copy |
|  | **In shallow copy a new object is created that has exact copy of the values in the original object but if any of the field of object are reference to other objects, just the referenced addressed are copied**  **in deep copy it copies all fields and also all reference objects**  https://miro.medium.com/max/390/0*RGt-o4ovYiIt_9nS. |
| 101 | **Call Stack** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| NOTES |  |
| 1 | a way to define global variable inside a module or any other place is by using window global object  window.global\_data = {value: 1} |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |