**Important Js Topics**

**HOISTING:**

Function declarations and variable declarations are moved to the top of the current scope i.e **function and variable declarations are added to memory during the compile phase.**

**Closure:**

A closure is the combination of a function and the lexical environment within which that function was declared.

**Callback Functions:**

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action

**Asynchronous Functions**:

An asynchronous model allows multiple things to happen at the same time. When you start an action, your program continues to run. When the action finishes, the program is informed and gets access to the result (for example, the data read from disk).

Synchronous operations in JavaScript entails having each step of an operation waits for the previous step to execute completely. This means no matter how long a previous process takes, subsquent process won't kick off until the former is completed. Asynchronous operations, on the other hand, defers operations. Any process that takes a lot of time to process is usually run alongside other synchronous operation and completes in the future.

**Event Loop:**

The Event Loop has a single job - it watches the Call Stack and the Callback Queue. When the Call Stack is empty, it takes the first event in the queue and pushes it to the stack which effectively runs it. Such an iteration is called a tick in the Event Loop. Each event is just a function callback.

**‘This’**

Objects are the basic building blocks in Javascript. There is one special object available in Javascript, “this” object. You can see the value of “this” at every line of Javascript execution. The value of “this” is decided based on how the code is being executed.

By default the execution context for an execution is global which means that if a code is being executed as part of a simple function call then “this” refers to global object. “window” object is the global object in case of browser and in Node.JS environment, a special object “global” will be the value of “this”.

If **strict mode** is enabled for any function then the value of “this” will be “undefined” as in strict mode, global object refers to undefined in place of windows object.

When a function is invoked with “new” keyword then the function is known as constructor function and returns a new instance. In such cases, the value of “this” refers to newly created instance.

In Javascript, property of an object can be a method or a simple value. When an Object’s method is invoked then “this” refers to the object which contains the method being invoked.

**“this” with call, apply methods**

A function in javascript is also a special type of object. Every function has call, bind and apply methods. These methods can be used to set custom value of “this” to the execution context of function.

The only difference between call and apply method is the way argument is passed. In case of apply, second argument is an array of arguments where in case of call method, arguments are passed individually.

**“this/” with bind method**

bind method returns a new method with “this” refers to the first argument passed. We are going to use above example to explain bind method

**“this” with fat arrow function**

When a fat arrow is used then it doesn’t create a new value for “this”. “this” keeps on referring to the same object it is referring, outside the function.

**Promises**

**Promises** are a clean way to implement async programming in JavaScript (ES6 new feature). Prior to promises, Callbacks were used to implement async programming.

**Shallow copy**

An object is said to be shallow copied when the source top-level properties are copied without any reference and there exist a source property whose value is an object and is copied as a reference. If the source value is a reference to an object, it only copies that reference value to the target object.

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

**Deep Copy:**

A deep copy will duplicate every object it encounters. The copy and the original object will not share anything, so it will be a copy of the original

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

**Example:**

Shallow Copy: It makes a copy of the reference to X into Y. Think about it as a copy of X’s Address. So, the addresses of X and Y will be the same i.e. they will be pointing to the same memory location.

Deep copy: It makes a copy of all the members of X, allocates different memory location for Y and then assigns the copied members to Y to achieve deep copy. In this way, if X vanishes Y is still valid in the memory.

The correct term to use would be cloning, where you know that they both are totally the same, but yet different (i.e. stored as two different locations in the memory space).

**Shadow DOM:**

Shadow DOM is just normal DOM with two differences: 1) how it's created/used and 2) how it behaves in relation to the rest of the page. Normally, you create DOM nodes and append them as children of another element. With shadow DOM, you create a scoped DOM tree that's attached to the element, but separate from its actual children. This scoped subtree is called a **shadow tree**. The element it's attached to is its **shadow host**. Anything you add in the shadows becomes local to the hosting element, including <style>. This is how shadow DOM achieves CSS style scoping.

The DOM a component author writes. Shadow DOM is local to the component and defines its internal structure, scoped CSS, and encapsulates your implementation details. It can also define how to render markup that's authored by the consumer of your component.

**Arrow Functions:**

An **arrow function expression** has a shorter syntax than a [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function) and does not have its own [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this), [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments), [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), or [new target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target). These function expressions are best suited for non-method functions, and they cannot be used as constructors.

**JIT - Compile TypeScript just in time for executing it.**

* Compiled in the browser.
* Each file compiled separately.
* No need to build after changing your code and before reloading the browser page.
* Suitable for local development.

**AOT - Compile TypeScript during build phase.**

* Compiled by the machine itself, via the command line (Faster).
* All code compiled together, inlining HTML/CSS in the scripts.
* No need to deploy the compiler (Half of Angular size).
* More secure, original source not disclosed.
* Suitable for production builds.

**Map method:**

The map() method creates a new array with the results of calling a provided function on every element in this array.

**Memory Leaks:**

Memory leaks can be defined as memory that is not required by an application anymore that for some reason is not returned to the operating system or the pool of free memory

**Event Propagation/Event Delegation:**

Capturing and bubbling allow us to implement one of most powerful event handling patterns called event delegation.

**Bubbling**

**When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.**

**Capturing phase** – the event goes down to the element.