**Frontend Assignment**

**Web Designing**

**MODULE: 6 (JavaScript ES6)**

**How can you create object in javascript? explain with example.**

**ANS.** There are different ways to create new objects:

* Using an Object Literal

Like : const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Using an Object literal, you both define and create an object in one statement. An object literal is a list of name:value (Like firstname:”John”)pairs inside curly braces{}.

* With Keyword new

Like : const person = new Object();  
 person.firstName = "John";  
 person.lastName = "Doe";

Using ‘new’ keyword we can create object like an above example and then we set the property and value in object.

* Define an object constructor, and then create objects of the constructed type.

Like : function person (name, age){;  
 this.name = name;  
 this.age = age;

}

Let emp = new person(‘john’, 25);

Here you can see in the example, we create object constructor function. And we can call this constructor making a new person constructor.

**How can you create Array in javascript? explain with example.**

**ANS.** We can create an array in different ways:

* Using an Array literal

Like : const cars = [car1, car2, car3];

Using Array literal easy way to create an Array. Define array name and it store many values in single variable. To get that values we can write the array name and that index values in square braces. And index value always start with 0 position.

* We can also create an array and then provide the elements.

Like : const cars = [];  
 cars[]= ‘car1’;

cars[]= ‘car2’;

**What is Callback? Explain with example.**

**ANS.** 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.

Example:

const message = function() {

console.log("This message is shown after 3 seconds");

}

setTimeout(message, 3000);

There is a built-in method in JavaScript called “setTimeout”, which calls a function or evaluates an expression after a given period of time (in milliseconds). So here, the “message” function is being called after 3 seconds have passed. (1 second = 1000 milliseconds)

In other words, the message function is being called after something happened (after 3 seconds passed for this example), but not before. So the message function is an example of a callback function.

**What are the ways to define a variable in javascript?**

**ANS.** There is 4 Ways to Declare a JavaScript Variable**:**

* Using var
* Using let
* Using const
* And nothing

**What is the difference between null and undefined?**

**ANS.**

|  |  |
| --- | --- |
| **Null** | **Undefined** |
| It is an assignment value. It can be assigned to a variable which indicates that a variable does not point any object. | It is not an assignment value. It means a variable has been declared but has not yet been assigned a value. |
| It is an object. | It is a type itself. |
| The null value is a primitive value which represents the null, empty, or non-existent reference. | The undefined value is a primitive value, which is used when a variable has not been assigned a value. |
| Null indicates the absence of a value for a variable. | Undefined indicates the absence of the variable itself. |
| Null is converted to zero (0) while performing primitive operations. | Undefined is converted to NaN while performing primitive operations. |

**What is NaN in javascript?**

**ANS.** In JavaScript, NaN is short for "Not-a-Number". In JavaScript, NaN is a number that is not a legal number. The Global NaN property is the same as the Number.

**How can you convert the string of any base to integer in javascript?**

**ANS.** To convert a string to an integer parseInt(), Number(), and Unary operator(+) function is used in javascript. parseInt() function returns Nan( not a number) when the string doesn’t contain number. If a string with a number is sent, then only that number will be returned as the output.

* **Using the parseInt() method**

parseInt(string)

let a = parseInt("100"); //output : 100

let b = parseInt("100.234"); //output : 100

let c = parseInt([1.2]); //output : 1

let d = parseInt(true); //output : NAN

let e = parseInt("hello") //output : NAN

* **Using the Number() method**

Number(value)

let a = Number("100"); //output : 100

let b = parseInt(Number("100.234")); //output : 100

let c = Number ([1]); //output : 1

let d = Number (true); //output : NAN

* **Using Unary Operator**

+op

let a = +"100"; // output : 100

let b = +"-100.234"; output : -100.234

**What will be the output of below code?**

var Y = 1;

if (function F(){})

{

    y += Typeof F;</span>

    }

    console.log(y);

**ANS.**

1 undefined

**When should I use Arrow function in Javascript?**

**ANS.** When you should use them. Arrow functions shine best with anything that requires this to be bound to the context, and not the function itself. Despite the fact that they are anonymous, I also like using them with methods such as map and reduce , because I think it makes my code more readable.

**Explain the Prtotype with example.**

**ANS.** The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.

Every function includes prototype object by default.

The prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function.

Example:

function Student() {

this.name = 'John';

this.gender = 'Male';

}

Student.prototype.age = 15;

var studObj1 = new Student();

alert(studObj1.age); // 15

var studObj2 = new Student();

alert(studObj2.age); // 15

**What is the difference between .forEach loop and .map loop?**

**ANS.**

| forEach() | map() |
| --- | --- |
| The forEach() method does not returns a  new array based on the given array. | The map() method returns an entirely new array. |
| The forEach() method returns “undefined“. | The map() method returns the newly created array according to the provided callback function. |
| The forEach() method doesn’t return anything hence the method chaining technique cannot be applied here. | With the map() method, we can chain other methods like, reduce(),sort() etc. |
| It is not executed for empty elements. | It does not change the original array. |

**What is constructor in javascript? Explain with example.**

**ANS.** A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

* The constructor keyword is used to declare a constructor method.
* The class can contain one constructor method only.
* JavaScript allows us to use parent class constructor through super keyword.

Example:

**<script>**

class Employee {

  constructor() {

    this.id=101;

    this.name = "Martin Roy";

  }

}

var emp = new Employee();

document.writeln(emp.id+" "+emp.name);

**</script>**

**What is the role of closure in javascript?**

**ANS.** Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

Example:

function OuterFunction() {

var outerVariable = 100;

function InnerFunction() {

alert(outerVariable);

}

return InnerFunction;

}

var innerFunc = OuterFunction();

innerFunc(); // 100

In the above example, return InnerFunction; returns InnerFunction from OuterFunction when you call OuterFunction(). A variable innerFunc reference the InnerFunction() only, not the OuterFunction(). So now, when you call innerFunc(), it can still access outerVariable which is declared in OuterFunction(). This is called Closure.

**What is the output of below code?**

var array = ["DataFlair", 2019, 1.0, true];

        var msg = "Array: [";

        for(var i = 0; i < array.length-1; i++){

            msg += array[i] + ", ";

        }

        msg += array[array.length-1] + "]";

            console.log(msg);

**ANS.** The output of above code is:

Array: [DataFlair, 2019, 1, true]

**What is meant by “this” in javascript?**

**ANS.** this points to a particular object. Now, which is that object is depends on how a function which includes 'this' keyword is being called.

The following four rules applies to this in order to know which object is referred by this keyword.

1. Global Scope
2. Object's Method
3. call() or apply() method
4. bind() method
5. **Global Scope:** If a function which includes 'this' keyword, is called from the global scope then this will point to the window object.
6. **Object's Method**: you can create an object of a function using new keyword. So, when you create an object of a function using new keyword then this will point to that particular object.
7. **call() or apply() method** : In JavaScript, a function can be invoked using () operator as well as call() and apply() method
8. **bind() method** : The bind() method was introduced since [ECMAScript](https://www.tutorialsteacher.com/articles/what-is-ecmascript) 5. It can be used to set the context of 'this' to a specified object when a function is invoked. The bind() method is usually helpful in setting up the context of this for a callback function.

**How to validate a form in javascript?**

**ANS.** Javascript validate form example is on below the link :

<https://github.com/Chigs18/Assignment/blob/main/validate_form_with_js.html>

**What are object prototypes?**

**ANS.** object's prototype property is invisible. Use Object.getPrototypeOf(obj) method instead of \_\_proto\_\_ to access prototype object.

Example:

function Student() {

this.name = 'John';

this.gender = 'M';

}

var studObj = new Student();

Student.prototype.sayHi= function(){

alert("Hi");

};

var studObj1 = new Student();

var proto = Object.getPrototypeOf(studObj1); // returns Student's prototype object

alert(proto.constructor); // returns Student function

The prototype object includes following properties and methods.

1. Constructor : Returns a function that created instance.
2. \_\_proto\_\_: This is invisible property of an object. It returns prototype object of a function to which it links to.

**What is the rest parameter?**

**ANS.** The rest parameter syntax allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.

### **Syntax**

function fun(a, b, ...theArgs) {

  // statements

}

### **Example**

function show(...args) {

  let sum = 0;

**for** (let i of args) {

      sum += i;

  }

  console.log("Sum = "+sum);

}

show(10, 20, 30);

**What is the use of promises in javascript?**

**ANS.** The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a *promise* to supply the value at some point in the future.

**What are classes in javascript?**

**ANS.** Classes are a template for creating objects. They encapsulate data with code to work on that data. Classes in JS are built on prototypes but also have some syntax and semantics that are unique to classes. For more examples and explanations, see the Using classes guide.

ES6 introduced a new syntax for declaring a class as shown in this example:

class Person {

constructor(name) {

this.name = name;

}

getName() {

return this.name;

}

}

This Person class behaves like the Person type in the previous example. However, instead of using a constructor/prototype pattern, it uses the class keyword.

In the Person class, the constructor() is where you can initialize the properties of an instance. JavaScript automatically calls the constructor() method when you instantiate an object of the class.