**Task-5**

**What is prototype?**

Javascript, invented by Brendan Eich in 1955 was initially just designed to run on the browser. It was popular as the client-side scripting language and was used to enhance the functionality on the client side.

The landscape of Javascript is not the same anymore. Several years after its invention, Javascript has diversified into its utility and hence you can easily understand that the importance of Javascript in the world of technology has taken a big leap.

In order to learn Javascript, there are various concepts you need to master. One of them is Javascript prototyping. In this article, we are going to have a comprehensive look and understand Prototyping in Javascript in detail.

According to the formal definition, prototype in JavaScript is the mechanism by which objects inherit properties from each other. But there is more to Javascript prototyping than just the definition. In order to understand this prototyping in Javascript, we need to get to the basics of objects and then prototyping in Javascript.

Javascript is an object-oriented programming language. It creates objects to solve programming problems. They represent entities, add logic so that they become behavior. In Javascript, objects interact with objects to solve computing problems.

Before learning about prototyping in Javascript, it is mandatory to have an in-depth understanding of objects. Let’s start our journey by learning about objects.

Objects in Javascript is a collection of properties and methods. These properties of an object are defined as a key-value pair. Objects in Javascript can be created in multiple ways. The simplest way to create objects in Javascript is -

* var myOBJ() = {
* “foo”: “value”;
* “age”: 21;
* } ;

The properties of an object are just like any other javascript variables that are attached to a particular object. You can access the property of a Javascript object with a dot operator. The name of the property in Javascript is case sensitive just like the object name and any other variable in this language.

Javascript also has methods which are basically just properties that are functions.

If you want a detailed explanation of the objects in Javascript, you can refer to this [BS1] article. Now, let’s get started with the prototype in Javascript.

The prototype is associated with every function and object by default. Every object that is created using the literal syntax or by using the constructor function or with the help of the new keyword contains the includes the prototype object.

* function Konfnity () {
* this.name = Student;
* this.interest = 'Technology';
* }
* var studObj1 = new Konfinity();
* var studObj2 = new Konfinity();

The code above makes a constructor function by the name of Konfinity. This function has two properties. Two instances studObj1 and studObj2 are created.

The question is, what If at a later stage, we want to add a new property to this object that shall be shared across all its instances. The answer to this question is prototype.

Let’s look at the code below to see how prototype actually adds a new property to the object.

* Konfinity.prototype.age = 15;
* var studObj3 = new Konfinity ();
* alert(studObj3.age); // 15

In the code above, student.prototype.age adds a new property to the object ‘age’. This property is added to every new instance of the object Konfinity.

We would now understand the prototype object, its method and properties in detail.

**Prototype Object**

Prototype objects enables other objects to inherit characteristics from them. The prototype object includes the following built-in methods and properties.

* Constructor – The constructor returns the function that created the instance.
* \_proto\_ - It returns the prototype object of the function to which that created the particular object. It is an invisible property.
* hasOwnProperty() – This is a function of the prototype object which returns a Boolean value (true or false). It is used
* to know if the object contains a specified property as a direct property of that object. The property should not be inherited from the protoype chain.
* isPrototypeOf() –This is another function of the prototype object which returns a Boolean value. This method indicates whether the object is in the prototype chain. The prototype chain is of the object this method is called upon.
* valuesOf() – This method returns the primitive value of the specified object.

Now that we have enough theoretical background on prototyping in Javascript, let us practically implement and understand the concept. We would begin by writing a simple constructor function by the name of Konfinity(). Let’s begin!

* function Konfinity ()
* {
* this. course = Web development;
* }
* Var k = new Konfinity () ;

The function above is a simple constructor function by the name of Konfinity. The new keyword is a special keyword which makes a new object of the function and assign it to the variable k.

If we go to the javascript console, the variable k is defined as –

* Konfinity
* Course = Web development
* \_\_proto\_\_: object

Let's invest some time in understanding the ‘ \_\_proto\_\_ ‘. This refers to an object. Prototype. It basically means the creator or the parent object from which the other object is created. Here the concept of prototype chain comes into picture which we would talk about when we discuss inheritance. There are a lot of methods that we talked about earlier like hasOwnProperty().

Now let’s write some more code and go back to the console log–

* Konfinity.prototype.show = function () {
* point (this.name); }

In this code we have made a function by the name of **show**in the object prototype. This means that every instance of the object prototype will inherit the **show**function automatically. The prototype object in the function defined by user refers to the function **show**that we defined as Konfinity.prototype.show .

The output in the console looks like -

* Konfinity
* Course = Web development
* \_\_proto\_\_: object
* Constructor: Konfinity ()
* Show: ()
* \_\_proto\_\_: Object

The difference is that the function **Konfinity**doesn’t have its own method rather the prototype of that object consists of that the method **show ()**.

The object looks for methods and properties in its prototype and includes them in its own.

We are now going to discuss our next topic, Prototype in Inheritance.

**Prototype Inheritance**

By far, we know that every function has a property called prototype and you can add a set of methods and functions to the prototype. Every instance of the object inherits by itself the methods and properties in the prototype.

Javascript is an object-oriented programming language and supports all OOP’s concept but their implementation is a bit different. Javascript, unlike other OOP’s language does not have a class implementation. Javascript has a prototype-based inheritance.

When it comes to inheritance, objects are most important in Javascript. The Prototype chain is the central concept in Inheritance.

* Every object has a link which holds to another, called the prototype
* The prototype object has a prototype of its own.
* The chain goes on until the prototype becomes null.
* Null has no prototype.
* The prototype as null becomes the final link.

The prototype chain begins with the first prototype of the object and ends with Null, the last prototype in the chain. It’s very important to get a hang of the prototypal inheritance and the prototype chain before getting into complex codes. Another thing to keep in mind is the length of the prototype chain. If it gets too long, it’s a good practice to break it down into smaller pieces for smooth execution of code and better performance.

It’s time we understood the inheritance in Javascript with an example.

We create one constructor function and define it

* var Konfinity (course, scope) = {
* this.course: course,
* this.scope: scope
* }

We now create objects student1 and student2 using the prototype

* var student1 = new Konfinity (“Web development”, “High”)
* var student2 = new Konfinity (“Front end development”, “High”)

We have used the new keyword to create two objects of the prototype.

Here comes the concept of inheritance by which one object can be based on other. They share each other’s properties and methods. We are now going to create a function by using the concept of inheritance with the help of prototype.

* Konfinity.prototype.dialogue = function () {
* Console.log ( ‘I am enrolled in ‘ + this.course )
* }
* console.log ( student1.dialogue ());

We have here put dialogue on Konfinity.prototype, we have made it available to all instances of Konfinity.

This was an easy example of Inheritance in Javascript using prototype. However, there are multiple complex ways in which inheritance can be implemented in Javascript. Differential inheritance and Future inheritance are some of the examples a javascript developer should study and understand.

Inheritance is an efficient way to make your code clean, compact and well structured. It helps developers to concentrate and focus on the main topic rather than keep getting stuck on unnecessary and lengthy code. Inheritance is one of the most prominent features of any computing language.

Here we come to an end of our brief introduction to Prototypes in Javascript. As always, if you have any doubts regarding the topics discussed above or you think we missed out on something important, you can always pop up questions and we would be more than happy to answer.

Prototypes in Javascript are very important in the world of web development. It is a complex concept and developers often find it difficult to understand and therefore implement it in their programs. But if you want to make a mark and master the language of Javascript then prototypes in Javascript is crucial in your journey.

The importance of Javascript can be gauged by the fact that over 94% of companies across the globe are using Javascript as a part of their development process. Hence, the demand for developers with a sound knowledge of Javascript is growing day by day.

If you are enthusiastic to become a web developer, Javascript can not be neglected. We have covered objects in Javascript in detail. However, in order to get sound knowledge and become an excellent professional according to standards set by industry, you require professional guidance.