1.// Task – difference -(Day-08 - 27.12.2021)

// 1. call// 2. apply// 3. Bind

**// 1. call**

* Call invokes the function and allows you to pass in arguments one by one.

**// 2. apply**

* Apply invokes the function and allows you to pass in arguments as an array.

**// 3. Bind**

* Bind returns a new function, allowing you to pass in a this array and any number of arguments.

Apply vs. Call vs. Bind Examples

**Call**

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {

console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);

}

say.call(person1, 'Hello'); // Hello Jon Kuperman

say.call(person2, 'Hello'); // Hello Kelly King

**Apply**

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {

console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);

}

say.apply(person1, ['Hello']); // Hello Jon Kuperman

say.apply(person2, ['Hello']); // Hello Kelly King

**Bind**

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say() {

console.log('Hello ' + this.firstName + ' ' + this.lastName);

}

var sayHelloJon = say.bind(person1);

var sayHelloKelly = say.bind(person2);

sayHelloJon(); // Hello Jon Kuperman

sayHelloKelly(); // Hello Kelly King

# 2.Prototypal Inheritance using \_\_proto\_\_ in JavaScript

Every object with its methods and properties contains an internal and hidden property known as **[[Prototype]]**. The Prototypal Inheritance is a feature in javascript used to add methods and properties in objects. It is a method by which an object can inherit the properties and methods of another object. Traditionally, in order to get and set the [[Prototype]] of an object, we use Object.getPrototypeOf and Object.setPrototypeOf. Nowadays, in modern language, it is being set using **\_\_proto\_\_**

**Syntax:**

ChildObject.\_\_proto\_\_ = ParentObject

**Example** In the given example, there are two objects ‘person’ and ‘GFGuser’. The object ‘GFGuser’ inherits the methods and properties of the object ‘person’ and further uses them.

|  |
| --- |
| <!DOCTYPE html>  <**html** lang="en">      <**head**>          <**meta** charset="UTF-8" />          <**title**>prototype</**title**>      </**head**>      <**body**>          <**script**>              // object person              let person = {                  talk: true,                  Canfly() {                      return "Sorry, Can't fly";                  },              };              // Object GFGuser              let GFGuser = {                  CanCode: true,                  CanCook() {                      return "Can't say";                  },                  //  Inheriting the properties and methods of person                  \_\_proto\_\_: person,              };                // Printing on console              // Property of person              console.log("Can a GFG User talk: " + GFGuser.talk);                // Method of person              console.log("Can a GFG User fly: " + GFGuser.Canfly());                // Property of GFGuser              console.log("Can a GFG User code: " + GFGuser.CanCode);                // Method of GFGuser              console.log("Can a GFG User cook: " + GFGuser.CanCook());          </**script**>      </**body**>  </**html**> |

Output:

//Can a GFG User talk: true

//Can a GFG User fly: Sorry, Can’t fly

//Can a GFG User code: true

//Can a GFG User cook: Can’t say