|  |
| --- |
| This always points to an Object, if there is no object in case of browser engine then it will point Window Object. In case of Node environment it will point to an empty Object. |
| const A = {  name : "Gaurav",  callback : function(){  console.log(this.name);  } }  A.callback();  Output: Gaurav |
| console.log(this);  const A = {  name : "Gaurav",  callback : ()=>{  console.log(this.name);  } }  A.callback();  Output: Window Object  Output: (Blank) |
| console.log(this);  this.name = "Raj" ;  const A = {  name : "Gaurav",  callback : ()=>{  console.log(this.name);  } }  A.callback();  Output: Window Object  Output: Raj  Note: this.name = “Raj”  It will add name(key) and Raj(value) in window object. |
| const A = {  name : "Gaurav",  callback : ()=>{  console.log(this);  } }  A.callback();  Output: Window Object |
| const A = {  name : "Gaurav",  callback : function(){  arrow=()=>{  console.log(this);  }  arrow();  } }  A.callback();  Output: A Object  This in arrow function points to an object of its parent function. In case there is no any parent function to arrow function then it will point to Window Object. |
| const A = {  name : "Gaurav",  callback : function(){  arrow=()=>{  console.log(this.name);  }  arrow();  } }  A.callback();  Output: Gaurav |
| this.name = "Gaurav Raj" ;  const A = {  name : "Gaurav",  callback1 : function(){    console.log(this.name);  } ,  callback2 : ()=>{  console.log(this.name);  }  }  A.callback1();  A.callback2();  Output: Gaurav  Output: Gaurav Raj |
| this.name = "Gaurav Raj" ;  const A = {  name : "Gaurav",  callback1 : function(){    function abc(){  console.log(this.name);  }  abc();  }  }  A.callback1();  Output: Gaurav Raj  Notice that abc function is not pointing parent function’s object. abc is pointing to window Object. |
| this.name = "Gaurav Raj" ;  const A = {  name : "Gaurav",  callback1 : function(){    abc=()=>{  console.log(this.name);  }  abc();  }  }  A.callback1();  Output: Gaurav  While arrow function pointing to an object of its parent function. |
| var roll = 7 ;  function callnotchild(){  console.log(this.roll);  }  const A = {  roll : 1,  parent : function(notchild){  notchild();  }  }  A.parent(callnotchild);  Output: 7 |
| var roll = 7 ;  var roll3 = 37 ;  this.roll5 = 57 ;  // all these are now the member of window object.  callchild=()=>{  console.log(this.roll);  }  callchild();  Output: 7 |
| var roll = 17 ;  callchild=()=>{  console.log(this.roll);  }  const A = {  roll : 1,  parent : function(child){  child();  }  }  A.parent(callchild);  Output : 17 |
| var roll = 17 ;  callchild=()=>{  console.log(this.roll);  }  const A = {  roll : 1,  parent : function(child){  child();  console.log(arguments);  arguments[0]();  console.log(arguments[2]);  }  }  A.parent(callchild, 5, 7);  Output: 17  Output: Aruments(3)  Output: 17  Output: 7 |
| This is known as Implicit Binding.  var calculator = {  total: 0 ,  addition : function(addparam){  this.total = this.total + addparam ;  return this;  },  subtraction : function(subparam){  this.total = this.total - subparam ;  return this ;  },  multiplication : function(multiparam){  this.total = this.total \* multiparam ;  return this;  },  division : function(divparam){  this.total = this.total / divparam ;  return this ;  }  }  var FinalResult = calculator.addition(13).subtraction(7).multiplication(5).division(2) ;  console.log(FinalResult);  console.log(FinalResult.total);  Output: calculator Object  {  total: 15,  addition: [Function: addition],  subtraction: [Function: subtraction],  multiplication: [Function: multiplication],  division: [Function: division]  }  Output : 15 |