Self-Invoking Functions:  
invoked automatically without being called  
this is a anonymous self invoking function(function without name)  
(function(){  
var x="hai";  
})();  
  
Arrow function:  
short syntax for writing function expression  
you don't need function keyword and the return keyword and the curly brackets  
Arrow functions are not hoisted.   
They must be defined before they are used.  
Using const is safer than using var, because a function expression is always constant value.  
ES5  
var x=function(x, y){  
rerurn x\*Y;  
}  
  
ES6  
var x=(x,y)=> x\*y  
x(5,5)  
  
class:  
class class\_name{  
  
}  
class expression  
var var\_name=class class\_name{  
}  
constructor-allocate memory for the objects of the class  
Ex:  
class Polygon {   
constructor(height, width) {   
this.height = height;   
this.width = width;   
}   
}  
Creating a object  
create a instance of a class.we need to use new keyword  
var object\_name= new class\_name([ arguments ])   
  
Ex:  
'use strict'   
class Polygon {   
constructor(height, width) {   
this.h = height;   
this.w = width;  
}   
test() {   
console.log("The height of the polygon: ", this.h)   
console.log("The width of the polygon: ",this. w)   
}   
}   
//creating an instance   
var polyObj = new Polygon(10,20);   
polyObj.test();   
  
instanceof operator  
returns true if the object belongs to the specified type  
Ex:  
'use strict'   
class Person{ }   
var obj = new Person()   
var isPerson = obj instanceof Person;   
console.log(" obj is an instance of Person " + isPerson);   
Output: obj is an instance of Person True   
  
Inheritance  
extends keyword used to inherit a class from parent class  
single=>  
multiple=>ES6 doesn’t support multiple inheritance.  
multi-level=>  
Ex:  
'use strict'   
class Root {   
test() {   
console.log("call from parent class")   
}   
}   
class Child extends Root {}   
class Leaf extends Child   
  
//indirectly inherits from Root by virtue of inheritance {}   
var obj = new Leaf();  
obj.test()   
  
method overloading  
Method Overriding is a mechanism by which the child class redefines the superclass method.  
Ex:  
'use strict' ;  
class PrinterClass {   
doPrint() {   
console.log("doPrint() from Parent called… ");  
}  
}  
class StringPrinter extends PrinterClass {   
doPrint() {   
console.log("doPrint() is printing a string…");   
}   
}   
var obj = new StringPrinter();   
obj.doPrint();  
Output: doPrint() is printing a string…   
  
Super keyword  
ES6 enables a child class to invoke its parent class data member. This is achieved by using the super keyword. The super keyword is used to refer to the immediate parent of a class.  
Ex:  
'use strict'   
class PrinterClass {   
doPrint() {  
console.log("doPrint() from Parent called…")   
}   
}   
class StringPrinter extends PrinterClass {   
doPrint() {   
super.doPrint()   
console.log("doPrint() is printing a string…")   
}   
}   
var obj = new StringPrinter()   
obj.doPrint()  
Output: doPrint() from Parent called.   
doPrint() is printing a string.   
  
Closure  
problem:we can access variable from outside defined variable   
lexical scope  
let i=1  
const f=()=>{  
console.log(i)  
}  
f();  
the variable available in inside block  
outside block  
let f;  
if(true){  
let i=1  
f=()=>{  
console.log(i)  
}  
}  
console.log(i)  
f();  
if we print i outside block it is not available  
  
closer inner function  
let f=()=>{  
let i=1  
let j=2  
return (){  
console.log(i)  
console.log(j)  
return i;  
}  
}  
f();  
  
closer inside a loop  
  
  
collection  
Es6 enable two different data structure  
Maps:This data structure enables mapping a key to a value  
Sets:set are similar to array,However set do not encourage duplicates  
  
Ex:  
Var myMap=new Map()  
myMap.set("J", "john");   
myMap.set("M", "mary");   
myMap.set("T", "tom");  
myMap.size  
OUTPUT  
3  
  
set()  
set() function sets the value for the in the map object  
it takes two parameters key and it's value  
This function returns the Map object.  
Ex:  
var map = new Map();   
map.set('name','Tutorial Point');   
  
has()  
The has() function returns a Boolean value indicating whether the specified key is found in the Map object  
This function takes a key as parameter  
var map = new Map();   
map.set(1,true);   
console.log(map.has("1")); //false   
  
map.set("1",true);   
console.log(map.has("1")); //true  
  
clear()  
Removes all key/value pairs from the Map object.  
var myMap = new Map();   
myMap.set("bar", "baz");   
console.log(myMap.size);   
  
myMap.clear();   
console.log(myMap.size)   
OUTPUT:  
1  
0  
  
delete(key)  
Parameters  
Key − key of the element to be removed from the Map.  
  
Return Value  
Returns true if the element exited and was removed; else it returns false.  
  
var myMap = new Map();   
myMap.set("id", "admin");   
myMap.set("pass", "admin@123");   
console.log(myMap.has("id"));   
  
myMap.delete("id");   
console.log(myMap.has("id"));   
OUTPUT  
true  
false  
  
entries()  
Returns a new Iterator object that contains an array of [key, value] for each element in the Map object in insertion order.  
  
var myMap = new Map();   
myMap.set("id", "admin");   
myMap.set("pass", "admin@123");   
console.log(myMap.entries());   
OUTPUT:  
{ [ 'id', 'admin' ], [ 'pass', 'admin@123' ] }  
  
For Loop  
'use strict'   
var roles = new Map([   
['r1', 'User'],   
['r2', 'Guest'],   
['r3', 'Admin'],   
]);  
for(let r of roles.entries())   
console.log(`${r[0]}: ${r[1]}`);  
OUTPUT:  
r1: User  
r2: Guest  
r3: Admin  
  
Promises:  
Promises are a clean way to implement async programming in JavaScript   
Prior to promises, Callbacks were used to implement async programming  
  
Callback  
A function may be passed as a parameter to another function. This mechanism is termed as a Callback. A Callback would be helpful in events.  
Ex:  
<script>   
function notifyAll(fnSms, fnEmail) {   
console.log('starting notification process');   
fnSms();   
fnEmail();   
}   
notifyAll(function() {   
console.log("Sms send ..");   
},   
function() {   
console.log("email send ..");   
});   
console.log("End of script");   
//executes last or blocked by other methods   
</script>   
  
OUTPUT:  
starting notification process   
Sms send ..   
Email send ..   
End of script   
  
modules