```
'use strict';

const normalObj = {}; // create a normal object
const nullProtoObj = Object.create(null); // create an object with "null"
prototype

console.log(`normalObj is: ${normalObj}`); // shows "normalObj is:
[object Object]"
//console.log(`nullProtoObj is: ${nullProtoObj}`); // throws error:
Cannot convert //object to primitive value

normalObj.valueOf(); // shows {}
//nullProtoObj.valueOf(); is error as it's calling toString() from Object

//console.log(nullProtoObj.toString());
console.log(Object.prototype.toString.call(nullProtoObj));

//assign
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

//It copies enumerable own properties from source to target.
//If a property already exists in target, it will be overwritten.
//It returns the modified target object.


console.log(target);
console.log(returnedTarget);

//defineProperties
const object={};
Object.defineProperties(object,{
      property:{value:33,writable:true},
      property2:{},
});
console.log(object.property);
object.property=44;
console.log(object.property);

Object.defineProperty(object,'def',{ // specifying the key of the
property to be defined
      value:"defining single property",
      writable:false,

});
console.log(object.def);

console.log(Object.entries(returnedTarget));

//entries
const teams={aston:"alonso","cadillac":"bottas",ferrari:"schumacher"};
for(const [k,v] of Object.entries(teams))
{
      console.log(`${k} : ${v}.`);
}
```

```
//object to map
const map = new Map(Object.entries(teams));
console.log(map);

//freeze
Object.freeze(object);
console.log(`Is Frozen:`,Object.isFrozen(object));
//makes existing properties non-writable and non-configurable

//Object.freeze(teams); // it throws error only in strict mode but in no
strict mode it does not throws error but does not allow to change the
value
teams.aston="Lance";
console.log(teams);

//Object.fromEntries() static method transforms a list of key-value pairs
into an object.
const arr = [
  ["0", "a"],
  ["1", "b"],
  ["2", "c"],
];

//fromEntries
const conv=Object.fromEntries(arr);
console.log(conv);

//getOwnPropertyDescriptor
let d = Object.getOwnPropertyDescriptor(object,'def');
console.log(d);

//getOwnPropertyDescriptors
let ds = Object.getOwnPropertyDescriptors(object);
console.log(ds);

//getOwnPropertyNames
console.log(Object.getOwnPropertyNames(object));
const symb=Object.getOwnPropertySymbols(object);
console.log(symb.length);

const prototype = {};
const prop_obj = Object.create(prototype);
console.log(Object.getPrototypeOf(prop_obj) === prototype);


const inventory = [
  { name: "asparagus", type: "vegetables", quantity: 9 },
  { name: "bananas", type: "fruit", quantity: 5 },
  { name: "goat", type: "meat", quantity: 23 },
  { name: "cherries", type: "fruit", quantity: 12 },
  { name: "fish", type: "meat", quantity: 22 },
];

//groupBy
const result = Object.groupBy(inventory, ({ quantity }) =>
  quantity < 6 ? "restock" : "sufficient",
);
console.log(result.restock);
```

```
//hasOwn
console.log(Object.hasOwn(object,"def"));

// Evaluation result is the same as using ===
console.log(Object.is(25, 25)); // true
console.log(Object.is("foo", "foo")); // true
console.log(Object.is("foo", "bar")); // false


//isExtensible
const object2 = {};
console.log("Extensible:");
console.log(Object.isExtensible(object2));
Object.preventExtensions(object);
console.log("Extensible:");
console.log(Object.isExtensible(object));

//Seal ,Changing property values on a sealed objectstill works.
console.log("Seal:");
console.log(Object.isSealed(object2));
Object.seal(object2);
console.log("Seal:");
console.log(Object.isSealed(object2));

//keys
const keyss = Object.keys(target);
console.log(keyss);

const obj = {};
const obj2 = Object.preventExtensions(obj);
console.log(obj === obj2);

//setPrototypeOf
const empty={};
const prop = {Name:"Lewis"};
Object.setPrototypeOf(empty,prop);
console.log(empty.Name);

//values
console.log(Object.values(teams));

//hasOwnProperty
console.log(`HasOwnProperty:`,prop.hasOwnProperty("Name"));

//propertyIsEnumerable
console.log(`IsEnumerable:`,object.propertyIsEnumerable("def"))

//constructor
const o1 = {};
console.log('object:',o1.constructor === Object); // true
const o2 = new Object();
o2.constructor === Object; // true
const a1 = [];
console.log('array:',a1.constructor === Array); // true
const a2 = new Array();
a2.constructor === Array; // true
const n = 3;
```

```
console.log('Number:',n.constructor === Number); // true

console.log(n.constructor);

//The base valueOf() method returns the this value itself
const objs = { foo: 1 };
console.log(objs.valueOf() === obj);

//isPrototypeOf
console.log(`IsPrototypeOf: `,prop.isPrototypeOf(empty));



class F1Driver {
  constructor(name, team, nationality, championships) {
    this.name = name;
    this.team = team;
    this.nationality = nationality;
    this.championships = championships;
  }
  toString() {
    return `F1 Driver ${this.name} races for ${this.team}, is
${this.nationality}, and has won ${this.championships}-8 world
championships.`;
  }
}
const lewis = new F1Driver("Lewis Hamilton", "Mercedes", "British", 7);

console.log(lewis); //
//console.log(`${lewis}`); // [object Object] if no toString() method is
defined
console.log(`${lewis}`); // this calls a toString() method that is
defined explicitly
console.log(lewis.toString()); // returns same output as above
```