```
// 1. constructor

const obj1 = {};

console.log("1. constructor:");

console.log(obj1.constructor); // [Function: Object]

console.log(obj1.constructor === Object); // true



const arr = [];

console.log(arr.constructor === Array); // true

console.log("\n");



// 2. hasOwnProperty()

const obj2 = { a: 1 };

console.log("2. hasOwnProperty:");

console.log(obj2.hasOwnProperty('a')); // true

console.log(obj2.hasOwnProperty('toString')); // false

console.log("\n");



// 3. isPrototypeOf()

function Animal() {}

function Dog() {}

Dog.prototype = Object.create(Animal.prototype);
```

```
const myDog = new Dog();



console.log("3. isPrototypeOf:");

console.log(Animal.prototype.isPrototypeOf(myDog)); // true

console.log(Dog.prototype.isPrototypeOf(myDog));    // true

console.log("\n");



// 4. propertyIsEnumerable()

const obj3 = { a: 1 };

Object.defineProperty(obj3, 'b', {

  value: 2,

  enumerable: false,

});



console.log("4. propertyIsEnumerable:");

console.log(obj3.propertyIsEnumerable('a')); // true

console.log(obj3.propertyIsEnumerable('b')); // false

console.log(obj3.propertyIsEnumerable('toString')); // false

console.log("\n");



// 5. toLocaleString()

const date = new Date(Date.UTC(2025, 7, 25));

const number = 123456.789;
```

```
console.log("5. toLocaleString:");

console.log(date.toLocaleString()); // Locale-based date string

console.log(number.toLocaleString('en-US')); // "123,456.789"

console.log(number.toLocaleString('de-DE')); // "123.456,789"

console.log("\n");



// 6. toString()

const obj4 = {};

const array = [1, 2, 3];



console.log("6. toString:");

console.log(obj4.toString()); // "[object Object]"

console.log(array.toString()); // "1,2,3"

console.log(date.toString()); // Full date string

console.log("\n");



// 7. valueOf()

const obj5 = {

  valueOf() {

    return 100;

  }

};
```

```
console.log("7. valueOf:");

console.log(obj5 + 50); // 150

const numObj = new Number(42);

console.log(numObj.valueOf()); // 42

console.log("\n");



// 8. __proto__ (legacy)

const obj6 = {};

console.log("8. __proto__ (legacy, avoid):");

console.log(obj6.__proto__ === Object.prototype); // true



// Better alternative:

console.log(Object.getPrototypeOf(obj6) === Object.prototype); // true
```

```
//1.

const target={a:"hello",b:"Dolly"};

const source ={c:12,d:13};

//Object.assign(target,....(nultiple sources can be given )source)

console.log(Object.assign(target,source));



//all the key value pair in the source is copied to the target and then the

target ios printed.this is wat the function does.




//2.

//we have to create a prototype and then create a new object for it.

const proto={hii() {return "This is the inside the prototype"}};

const obj=Object.create(proto);

console.log(obj.hii());



//3.this will help to set the properties of the object

const obj1={};

Object.defineProperty(obj1,"monny",{

    value:12,

    writable:false,
```

```
});



obj1.monny=34;

console.log(obj1.monny)



//4.to give multiple properties for a single object

const obj2={};

Object.defineProperties(obj2,{

    donny:{value:"Dolly"},

    nonny:{value:23}

    }

);

console.log(obj2.donny);

console.log(obj2.nonny);



//5.this returns a keyvalue pair

const obj3={a:1,b:2};

console.log(Object.entries(obj3));



//6.this the opposite of the object.entries ,it turns the key value pair to

object

const entries=[['dolly',29],['poppy',34]]

console.log(Object.fromEntries(entries));
```

```
//7.it makes the object immutable ,ie it makes it not avalible to make

changes in the properties of the object

const obj4={a:23,b:66,c:82};

Object.freeze(obj4);

obj4.b=22;

console.log(obj4);



//8.it give the propertiess of the the

object.(writeable,enumerable,configurable)



let arr=Object.getOwnPropertyDescriptor({x:23},'x');

console.log(arr);



//9.this is same as  getownpropertyDescription but will give all the details



let arr1=Object.getOwnPropertyDescriptor({x:23},'x');

console.log(arr1);



//10.it give the name of the property that is there in the object

const obj5=Object.create({},{

    hidden:{value:34,enumerable:false},

    open:{value:33,enumerable:true}
```

```
});

console.log(Object.getOwnPropertyNames(obj5));



//11.its gives the boolen value if the object is extendable or not

console.log(Object.isExtensible(obj3));



//12.isFrosen(obj) is again a boolean type

console.log(Object.isFrozen(obj3));



//13.Returns true if properties can't be added or removed, but can still be

changed.

console.log(Object.isSealed(obj3))



//14.Returns enumerable property names (string keys).



console.log(Object.keys(obj4));
```