**Objects predefined methods**

1. **Object.create()**

Objects can be created using the Object.create().

|  |
| --- |
| var person = {              name: "Rowan",              displayName() { console.log(this.name)},          };          var person1 = Object.create(person);          person1.displayName(); |

# Object.prototype.valueOf()

# converts this value to an object.

|  |
| --- |
| function MyNumberType(n) {              this.number = n;          }          MyNumberType.prototype.valueOf = function () {              return this.number;          };          var object1 = new MyNumberType(4);          console.log(object1 + 3); |

1. **toLocaleString()**

toLocaleString() method calls toString().

|  |
| --- |
| var obj = {              toString() {                  return "My Object"; }          };          console.log(obj.toLocaleString()); |

1. **\_\_defineGetter\_\_()**

method binds an object's property to a function, \_\_defineGetter\_\_(prop, func)

* prop is a name of prototype that getter function bound

|  |
| --- |
| var obj = {};          obj.\_\_defineGetter\_\_("gimmeFive", function () {              return 5;          });          console.log(obj.gimmeFive); |

1. **get()**

get look like \_\_defineGetter\_\_() but I doesn't take function and prop it's merge them

|  |
| --- |
| const obj = {              get gimmeFive() {                  return 5;              },          };          console.log(obj.gimmeFive); |

1. Object.getPrototypeOf()

It's return prototype of object

|  |
| --- |
| var prototype1 = {};          var object1 = Object.create(prototype1);          console.log(Object.getPrototypeOf(object1) === prototype1); |

# Object.hasOwn()

# returns true if the object has the indicated property as its own property. Otherwise false

|  |
| --- |
| var object1 = {              prop: ''          };          console.log(Object.hasOwn(object1, 'prop'));          //output: true          console.log(Object.hasOwn(object1, 'toString'));          //output: false |

# 8-9-call() and apply()

# The call() method calls the function with a given this value and. it's almost like apply() except that call() accepts argument list.

# apply() accepts a single array of arguments

|  |
| --- |
| func.apply(this, ['eat', 'bananas'])          func.call(this, 'eat', 'bananas'). |

# 10- isPrototypeOf()

# checks if an object exists in another object's prototype chain.

|  |
| --- |
| function Foo() { }          function Bar() { }  var bar = new Bar();          Bar.prototype = Object.create(Foo.prototype);          console.log(Foo.prototype.isPrototypeOf(bar));          // Expected output: true |