Object Data Type Methods:

1. **`Object.assign(target, ...sources)`**

Merges properties from one or more source objects into a target object. Existing properties in the target object are overwritten if they are present in the source objects.

```javascript
let target = { a: 1 };
let source1 = { b: 2 };
let source2 = { c: 3, d: 4 };

Object.assign(target, source1, source2);
console.log(target); // Output: { a: 1, b: 2, c: 3, d: 4 }
```

2. **`Object.create(proto, [propertiesObject])`**

Creates a new object with the specified prototype object and optional properties.

```javascript
let proto = { greet: function() { console.log("Hello!"); } };
let obj = Object.create(proto);
obj.greet(); // Output: Hello!
```

3. **`Object.defineProperty(obj, prop, descriptor)`**

Defines a new property directly on an object, or modifies an existing property, and returns the object.

```javascript
let obj = {};
Object.defineProperty(obj, "name", {
  value: "John",
  writable: false,
  configurable: false,
  enumerable: true,
});
console.log(obj.name); // Output: John
```

4. **`Object.defineProperties(obj, props)`**

Defines multiple properties on an object.

```javascript
let obj = {};
Object.defineProperties(obj, {
  name: {
    value: "John",
    writable: true,
    enumerable: true,
  },
  age: {
    value: 30,
    writable: false,
    enumerable: true,
  }
});
console.log(obj); // Output: { name: 'John', age: 30 }
```

5. **`Object.entries(obj)`**

Returns an array of key-value pairs (entries) from the provided object.

```javascript
let obj = { a: 1, b: 2, c: 3 };
console.log(Object.entries(obj)); // Output: [['a', 1], ['b', 2], ['c', 3]]
```

6. **`Object.freeze(obj)`**

Freezes an object, preventing it from being modified.

```javascript
let obj = { a: 1, b: 2 };
Object.freeze(obj);
obj.b = 3; // Attempt to modify the object
console.log(obj.b); // Output: 2 (modification prevented)
```

7. **`Object.getOwnPropertyDescriptor(obj, prop)`**

Returns the descriptor of a property directly on an object.

```javascript
```

```
let obj = { name: "John" };
let descriptor = Object.getOwnPropertyDescriptor(obj, "name");
console.log(descriptor); // Output: { value: 'John', writable: true, enumerable: true, configurable: true }
```

8. **`Object.getOwnPropertyDescriptors(obj)`**

Returns an object containing all own property descriptors of the given object.

```javascript
let obj = { name: "John", age: 30 };
let descriptors = Object.getOwnPropertyDescriptors(obj);
console.log(descriptors);
/*
Output:
{
  name: {
    value: 'John',
    writable: true,
    enumerable: true,
    configurable: true
  },
  age: {
    value: 30,
    writable: true,
    enumerable: true,
    configurable: true
  }
}
*/
```

9. **`Object.getOwnPropertyNames(obj)`**

Returns an array of all own property names (keys) of an object, including non-enumerable properties.

```javascript
let obj = { a: 1, b: 2, c: 3 };
Object.defineProperty(obj, "hidden", {
  value: "secret",
  enumerable: false,
});
```

console.log(Object.getOwnPropertyNames(obj)); // Output: ['a', 'b', 'c', 'hidden']

10. **`Object.getOwnPropertySymbols(obj)`**

Returns an array of all own property symbols of an object.

```javascript
let symbolKey = Symbol("symbol");
let obj = {
  [symbolKey]: "hidden"
};
console.log(Object.getOwnPropertySymbols(obj)); // Output: [Symbol(symbol)]
```

11. **`Object.getPrototypeOf(obj)`**

Returns the prototype of the specified object.

```javascript
let obj = { a: 1 };
let prototype = Object.getPrototypeOf(obj);
console.log(prototype); // Output: Object.prototype (or null if the object does not have a
prototype)
```

12. **`Object.is(value1, value2)`**

Determines whether two values are the same value.

```javascript
console.log(Object.is(0, -0)); // Output: false
console.log(Object.is(NaN, NaN)); // Output: true
```

13. **`Object.isExtensible(obj)`**

Determines whether an object is extensible (whether new properties can be added to it).

```javascript
let obj = { a: 1 };
console.log(Object.isExtensible(obj)); // Output: true
Object.preventExtensions(obj);
console.log(Object.isExtensible(obj)); // Output: false
```

14. **`Object.isFrozen(obj)`**

Determines whether an object is frozen (whether its properties are immutable).

```javascript
let obj = { a: 1, b: 2 };
console.log(Object.isFrozen(obj)); // Output: false
Object.freeze(obj);
console.log(Object.isFrozen(obj)); // Output: true
```

15. **`Object.isSealed(obj)`**

Determines whether an object is sealed (whether its properties cannot be added or removed, though their values can still change).

```javascript
let obj = { a: 1, b: 2 };
console.log(Object.isSealed(obj)); // Output: false
Object.seal(obj);
console.log(Object.isSealed(obj)); // Output: true
```

16. **`Object.keys(obj)`**

Returns an array of enumerable property names (keys) of an object.

```javascript
let obj = { a: 1, b: 2, c: 3 };
console.log(Object.keys(obj)); // Output: ['a', 'b', 'c']
```

17. **`Object.preventExtensions(obj)`**

Prevents any new properties from being added to an object.

```javascript
let obj = { a: 1 };
Object.preventExtensions(obj);
obj.b = 2; // Attempt to add a new property
console.log(obj.b); // Output: undefined (new property addition prevented)
```

18. **`Object.seal(obj)`**

Seals an object, preventing the addition or removal of properties.

```javascript
let obj = { a: 1 };
Object.seal(obj);
delete obj.a; // Attempt to remove a property
console.log(obj.a); // Output: 1 (property removal prevented)
```

19. **`Object.setPrototypeOf(obj, prototype)`**

Sets the prototype of the specified object to the provided prototype.

```javascript
let proto = { greet: function() { console.log("Hello!"); } };
let obj = { a: 1 };
Object.setPrototypeOf(obj, proto);
obj.greet(); // Output: Hello!
```

20. **`Object.values(obj)`**

Returns an array of values from an object.

```javascript
let obj = { a: 1, b: 2, c: 3 };
console.log(Object.values(obj)); // Output: [1, 2, 3]
```