Cheatsheet

with explanation and examples of each of methods for an interview

JS

OBJECT METHODS IN JS

JavaScript objects have a variety of built-in methods that allow you to perform various operations on them.

1. **Object.assign(target, source1, source2, ...)**: Copies the values of all enumerable properties from one or more source objects to a target object, creates a shallow copy.
2. **Object.keys(obj)**: Returns an array of strings that represent all the enumerable properties of an object.
3. **Object.values(obj)**: Returns an array of values corresponding to the enumerable properties of an object.
4. **Object.entries(obj)**: Returns an array of arrays, where each inner array contains a key-value pair representing an enumerable property of the object.
5. **Object.getOwnPropertyNames(obj)**: Returns an array of all property names (enumerable or not) of an object.
6. **Object.getOwnPropertyDescriptor(obj, prop)**: Returns an object describing the property attributes of a specified property.
7. **Object.create(proto, [propertiesObject])**: Creates a new object with the specified prototype object and optional properties.
8. **Object.defineProperty(obj, prop, descriptor)**: Adds or modifies a property with a specified descriptor to an object.
9. **Object.defineProperties(obj, props)**: Adds or modifies multiple properties with specified descriptors to an object.
10. **Object.getPrototypeOf(obj)**: Returns the prototype of the specified object.
11. **Object.setPrototypeOf(obj, prototype)**: Sets the prototype (i.e., the internal **[[Prototype]]** property) of an object.
12. **obj.hasOwnProperty(prop)**: Returns a boolean indicating whether the object has the specified property as its own property (not inherited).
13. **obj.propertyIsEnumerable(prop)**: Returns a boolean indicating whether the specified property is enumerable.
14. **obj.toString()**: Returns a string representation of the object.
15. **obj.toLocaleString()**: Returns a string representation of the object using local conventions.
16. **obj.valueOf()**: Returns the primitive value of the object.

Please note that these methods are generally available for objects in JavaScript, but some objects (such as those created using constructors or classes) might have additional methods specific to their type. Additionally, the behavior of these methods might vary depending on the version of JavaScript you're using. Always refer to the official documentation for the most accurate and up-to-date information.

Object.**assign(**target, source1, source2, ...**)**:

```
const target = {};
const source = { a: 1, b: 2 };
Object.assign(target, source);
console.log(target);  //  { a: 1, b: 2 }
```

Object.**keys(**obj**)**:

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

Object.**values(**obj**)**:

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

Object.**entries(**obj**)**:

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

Object.**getOwnPropertyNames(**obj**)**:

```
const obj = { a: 1, b: 2 };
const propertyNames = Object.getOwnPropertyNames(obj);
console.log(propertyNames);  //  [ 'a', 'b' ]
```

Object.**getOwnPropertyDescriptor(**obj, prop**)**:

```
const obj = { a: 1 };
const descriptor = Object.getOwnPropertyDescriptor(obj, 'a');
console.log(descriptor);  // { value: 1, writable: true, enumerable: true, configurable: true }
```

Object.**create(**proto, [propertiesObject]**)**:

```
const protoObj = { greet: function() { console.log('Hello!'); } };
const newObj = Object.create(protoObj);
newObj.greet();  // Hello!
```

Object.**defineProperty(**obj, prop, descriptor**)**:

```
const obj = {};
Object.defineProperty(obj, 'x', { value: 42, writable: false });
console.log(obj.x);  // 42
obj.x = 10;  // Won't change the value due to `writable: false`
console.log(obj.x);  // 42
```

Object.**defineProperties(**obj, props**)**:

```
const obj = {};
Object.defineProperties(obj, {
    x: { value: 1, writable: true },
    y: { value: 2, writable: false }
});
console.log(obj.x, obj.y);  // 1 2
```

Object.**getPrototypeOf(**obj**)**:

```
const parent = { a: 1 };
const child = Object.create(parent);
console.log(Object.getPrototypeOf(child) === parent);  // true
```

Object.**setPrototypeOf(**obj, prototype**)**:

```
const parent = { a: 1 };
const child = {};
Object.setPrototypeOf(child, parent);
console.log(child.a);  // 1
```

obj.**hasOwnProperty(**prop**)**:

```
const obj = { a: 1 };
console.log(obj.hasOwnProperty('a'));  //true
console.log(obj.hasOwnProperty('b'));  // false
```

obj.**propertyIsEnumerable(**prop**)**:

```
const obj = { a: 1 };
console.log(obj.propertyIsEnumerable('a'));  // true
console.log(obj.propertyIsEnumerable('toString'));  // false
```

obj.**toString()**:

```
const obj = { a: 1 };
console.log(obj.toString());  // [object Object]
```

obj.**toLocaleString()**:

```
const date = new Date();
console.log(date.toLocaleString());  // 8/5/2023, 12:34:56 PM (format may vary)
```

obj.**valueOf()**:

```
const num = new Number(42);
console.log(num.valueOf());  // 42
```

Please note that these examples provide a basic understanding of each method. The exact behavior and usage might differ in specific contexts.

LIKE/ SHARE/ SUBSCRIBE:

LINKEDIN.COM/IN/HENNADIIZHUKOV

THANK YOU!