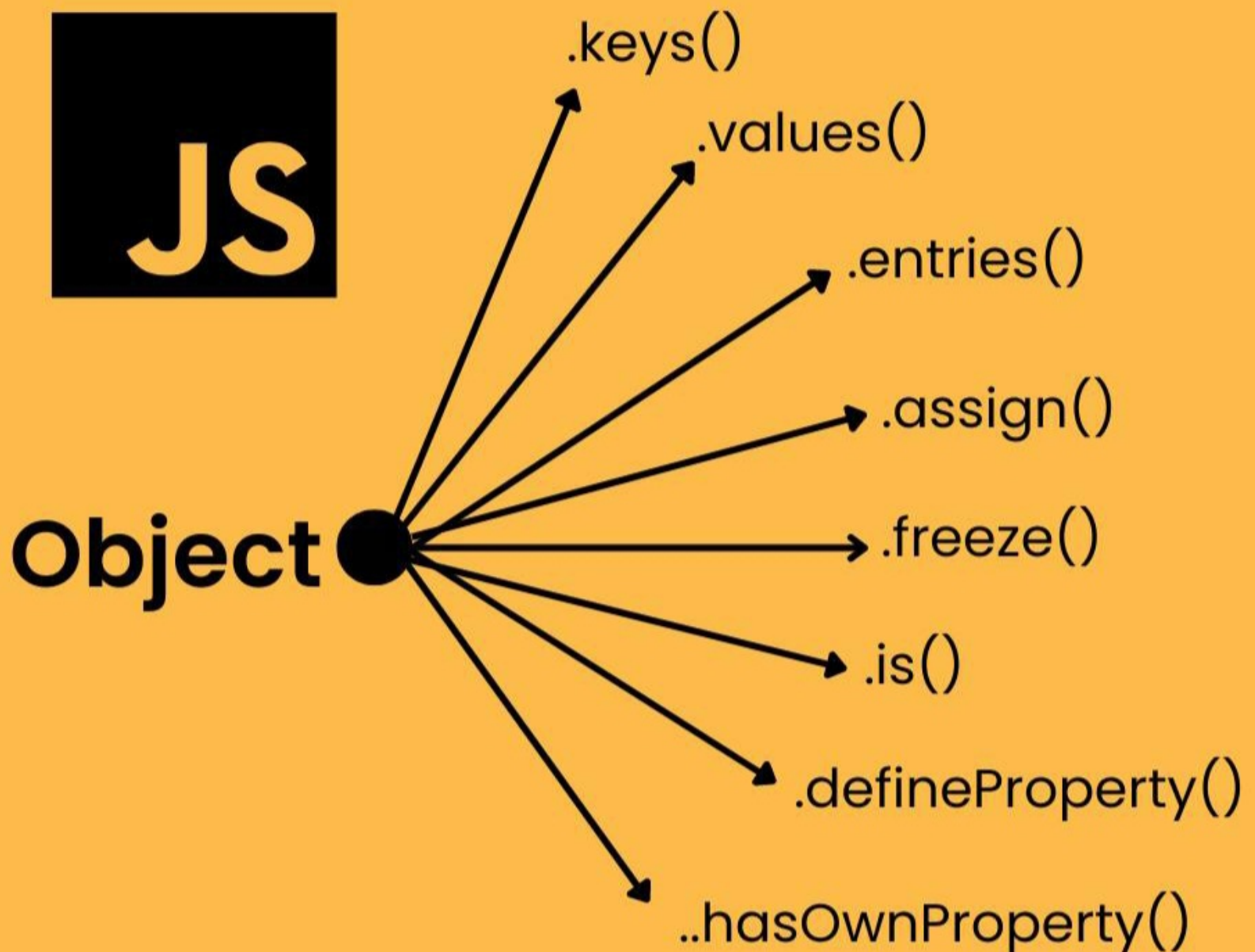


# JavaScript Object Methods



# 1. keys

This method retrieves the names of all the enumerable properties from an object.

It's handy when you need to iterate over properties or when you just want to get a quick overview of the object's structure.



```
1  const car = { make: 'Toyota', model: 'Camry' };  
2  console.log(Object.keys(car)); // ['make', 'model']
```

## 2. values

This method, similar to `Object.keys()`, gives you the values instead of the keys.

It's particularly useful when you are more interested in the data stored rather than their property names.



```
1 console.log(Object.values(car)); // ['Toyota', 'Camry']
```

### 3. entries

This method combines both keys and values to produce an array of pairs.

Each pair, represented by an array, consists of a property name followed by its corresponding value.

This can be beneficial when working with functions that expect key-value pairs.



```
1 console.log(Object.entries(car));  
2 // [['make', 'Toyota'], ['model', 'Camry']]
```

## 4. assign

It's like a photocopier for objects. This method is used to copy values from one or more source objects to a target object.

It's essential when you need to merge objects or create copies with additional properties.



```
1  const details = { color: 'red' };  
2  const carWithDetails = Object.assign(car, details);  
3  console.log(carWithDetails);  
4  // { make: 'Toyota', model: 'Camry', color: 'red' }
```



## 5. freeze()

Think of this as a protective shield for your object.

Once an object is frozen, you can't add, delete, or modify its properties. It's useful when you want to ensure data integrity.



```
1 Object.freeze(car);  
2 car.year = 2020;  
3 // Throws an error because car is frozen.
```

## 6. is()

While it may look like the strict equality (`===`) operator, this method has some nuanced differences, especially when comparing NaN values.

It checks if two values are the same, including distinguishing between positive and negative zeros.



```
1 console.log(Object.is('hello', 'hello')); // true
2 console.log(Object.is(NaN, NaN));
3 // true (whereas NaN === NaN returns false)
```

## 7. `defineProperty()`

This method is your advanced tool for adding new properties or modifying existing ones.

You can set characteristics like enumerability, writability, and configurability for the property, offering fine-grained control over its behavior.



```
1 Object.defineProperty(car, 'year', { value: 2020, writable: false });  
2 console.log(car.year); // 2020
```



## 8. `hasOwnProperty()`

This method checks if an object has a specific property as its direct property (not inherited from its prototype).

It's a safer way to check for properties than using the `in`-operator, especially when working with objects that might have overridden built-in properties.



```
1 .log(car.hasOwnProperty('make')); // true
```