🏆 JavaScript Objects Cheat Sheet 🏆

📦 Object Basics

- **Creation**: `let obj = {key: value};`
- **Access**: `obj.key` or `obj["key"]` (brackets for dynamic/special keys)
- **Add/Modify**: `obj.newKey = value;` or `obj["newKey"] = value;`
- **Delete**: `delete obj.key;`
- **Check existence**: `"key" in obj` or `Object.hasOwn(obj, "key")`

🔄 Object Methods

- **Keys**: `Object.keys(obj)` → array of keys
- **Values**: `Object.values(obj)` → array of values
- **Entries**: `Object.entries(obj)` → array of [key, value] pairs
- **FromEntries**: `Object.fromEntries(array)` → object from [key, value] pairs
- **Loop**: `for (let key in obj) { ... }`

🚀 ES6 Shortcuts

- **Property shorthand**: `const obj = { name, age };` (uses existing variables)
- **Method shorthand**: `const obj = { method() { ... } };`
- **Dynamic keys**: `const obj = { [keyName]: value };`

🛠️ Creating Objects

1. **Object literal**: `let obj = {key: value};`
2. **Constructor**: `function Person(name) { this.name = name; }`
3. **Factory**: `function createPerson(name) { return {name}; }`

🧩 Advanced Concepts

- **Nested objects**: `obj.address.city` (objects inside objects)
- **Optional chaining**: `obj?.prop?.subprop` (prevents errors)
- **Destructuring**: `const {name, age} = person;`
- **Rename in destructure**: `const {name: fullName} = person;`

- **Nested destructuring**: `const {address: {city}} = person;`

📋 Copying Objects

- **Shallow copy**:
  - `Object.assign({}, obj)`
  - `{...obj}`
- **Deep copy**:
  - `structuredClone(obj)`
  - `JSON.parse(JSON.stringify(obj))`

🔒 Immutability

- **Freeze**: `Object.freeze(obj)` (no changes allowed)
- **Seal**: `Object.seal(obj)` (modify existing only)

🧠 Remember

- Objects are reference types (copied by reference)
- Modifying nested objects in shallow copies affects the original
- Use dot notation for known keys, bracket notation for dynamic keys
- Always use `this` in object methods to refer to the object itself

🏆 JavaScript Objects Quick Reference Tables 🏆

📦 Basic Object Operations

Operation	Syntax	Example
Create	`{}`	`let person = {name: "John", age: 30};`
Access	`.` or `[]`	`person.name` or `person["name"]`
Add/Update	`.prop =` or `["prop"] =`	`person.email = "john@example.com";`
Delete	`delete`	`delete person.age;`
Check existence	`in` or `hasOwn()`	`"name" in person` or `Object.hasOwn(person, "name")`

🔄 Object Methods

Method	Purpose	Example	Result
`Object.keys()`	Get all keys	`Object.keys(person)`	`["name", "age"]`
`Object.values()`	Get all values	`Object.values(person)`	`["John", 30]`
`Object.entries()`	Get key-value pairs	`Object.entries(person)`	`[["name","John"], ["age",30]]`
`Object.fromEntries()`	Array to object	`Object.fromEntries([["a",1], ["b",2]])`	`{a: 1, b: 2}`
`for...in`	Loop over keys	`for (let key in person) {...}`	Iterates all keys

🚀 ES6 Shortcuts

Feature	Old Syntax	ES6 Syntax
Property shorthand	`{name: name}`	`{name}`
Method shorthand	`{add: function(x,y){...}}`	`{add(x,y){...}}`
Dynamic keys	`obj["key"] = value`	`{[keyName]: value}`
Spread operator	`Object.assign({}, obj)`	`{...obj}`

🛠️ Ways to Create Objects

Method	Syntax	Use Case
Object literal	`let obj = {key: value};`	Simple objects
Constructor	`function Person(name){this.name=name}`	Multiple similar objects
Factory function	`function createPerson(name){return {name}}`	Objects with private data
Object.create	`Object.create(prototype)`	Prototypal inheritance

🧩 Advanced Features

Feature	Syntax	Purpose
Nested objects	`person.address.city`	Organize related data
Optional chaining	`person?.address?.city`	Safe access to nested props
Destructuring	`const {name, age} = person;`	Extract multiple properties
Rename in destructure	`const {name: fullName} = person;`	Extract with new names
Nested destructuring	`const {address: {city}} = person;`	Extract from nested objects

📋 Object Copying

Copy Type	Methods	Behavior with Nested Objects
Shallow	`Object.assign({}, obj)` `{...obj}`	Nested objects remain references
Deep	`structuredClone(obj)` `JSON.parse(JSON.stringify(obj))`	All nested objects are copied

🔒 Immutability Options

Method	Effect	Can Modify Properties?	Can Add/Delete?
`Object.freeze()`	Complete immutability	❌ No	❌ No
`Object.seal()`	Prevents adding/deleting	✅ Yes	❌ No

🧠 Common Patterns & Gotchas

Scenario	Description	Solution
Reference comparison	`{} === {}` is always false	Compare specific properties instead
Mutating original in copies	Changes to nested objects affect original	Use deep copy methods
Method context	`this` may change when method is passed around	Use bind(), arrow functions, or method shorthand
Dynamic access	Need to use variable as key	Use bracket notation: `obj[dynamicKey]`