JavaScript OOP Cheat Sheet

What is OOP?

Object-Oriented Programming organizes code into objects with properties (data) and methods (functions).

Core Principles:

- Encapsulation: Hiding internal details
- Inheritance: Share functionality from parent
- Polymorphism: Same method, different behavior
- Abstraction: Hide implementation behind interface

Creating Objects:

- Object literal:

```
const obj = { name: 'Alice', greet() { console.log('Hi'); } };
```

- Constructor function + prototype:

```
function Person(name) { this.name = name; }
Person.prototype.greet = function() { console.log(...); };
```

- ES6 Class:

```
class Person { constructor(name) { this.name = name; } greet() { ... } }
```

Classes and Instances:

```
class Car { constructor(make, model) { this.make = make; this.model = model; } drive() { ... } }
const car1 = new Car('Toyota', 'Corolla');
```

Inheritance:

class Animal { speak() { ... } }

class Dog extends Animal { speak() { ... } }

super():

class Dog extends Animal {

constructor(name) { super(name); ... }

}

Static Methods:

class MathUtils { static add(a, b) { return a + b; } }

MathUtils.add(2, 3);

Encapsulation:

class Person {

#ssn;

constructor(name, ssn) { this.name = name; this.#ssn = ssn; }

getSSN() { return this.#ssn; }

}

Polymorphism:

class Shape { draw() { ... } }

class Circle extends Shape { draw() { ... } }

this Keyword: refers to current instance.

Prototypes: Every object inherits from prototype.

Getters and Setters:

```
class User {
get name() { return this._name; }
set name(value) { this._name = value; }
}
```

instanceof:

```
console.log(car1 instanceof Car);
```

Common Patterns:

- Factory
- Singleton
- Observer
- Prototype