The four core principles of Object-Oriented Programming (OOP)

are Encapsulation, Abstraction, Inheritance, and Polymorphism. These principles promote code organization, reusability, and maintainability.

1. Encapsulation:

Encapsulation involves bundling data (properties) and methods that operate on that data within a single unit, typically a class. It also restricts direct access to some of an object's components, preventing external code from directly manipulating internal state.

JavaScript

```
class BankAccount {
  #balance; // Private property using a hash prefix

  constructor(initialBalance) {
    this.#balance = initialBalance;
  }

  deposit(amount) {
    if (amount > 0) {
      this.#balance += amount;
      console.log(`Deposited: $${amount}. New balance: $${this.#balance}`);
    } else {
      console.log("Deposit amount must be positive.");
    }
  }

  getBalance() {
    return this.#balance;
  }
}

const myAccount = new BankAccount(100);
myAccount.deposit(50);
// console.log(myAccount.#balance); // This would cause an error as #balance is pr
console.log(`Current balance: $${myAccount.getBalance()}`);
```

2. Abstraction:

Abstraction focuses on showing only essential information and hiding complex implementation details. It allows users to interact with objects at a high level without needing to understand the underlying mechanisms.

JavaScript

```javascript
class Car {
  constructor(make, model) {
    this.make = make;
    this.model = model;
  }

  // Abstracting the complex process of starting the car
  start() {
    this.#igniteEngine();
    this.#checkFuel();
    console.log(`${this.make} ${this.model} is started.`);
  }

  #igniteEngine() {
    // Complex engine ignition logic
    console.log("Engine ignited.");
  }

  #checkFuel() {
    // Fuel level check
    console.log("Fuel level checked.");
  }
}

const myCar = new Car("Toyota", "Camry");
myCar.start(); // User only needs to know 'start()', not the internal steps
```

3. Inheritance:

Inheritance allows a new class (subclass/child class) to inherit properties and methods from an existing class (superclass/parent class). This promotes code reuse and establishes a hierarchical relationship between classes.

JavaScript

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

  speak() {
    console.log(`${this.name} makes a sound.`);
  }
}

class Dog extends Animal {
  constructor(name, breed) {
    super(name); // Call parent constructor
    this.breed = breed;
  }

  speak() {
    console.log(`${this.name} barks.`); // Overriding parent method
  }
}

const doggo = new Dog("Buddy", "Golden Retriever");
doggo.speak();
```

4. Polymorphism:

Polymorphism means "many forms." In OOP, it allows objects of different classes to be treated as objects of a common type, and for methods with the same name to behave differently based on the object's type.

JavaScript

```javascript
class Shape {
  draw() {
    console.log("Drawing a generic shape.");
  }
}

class Circle extends Shape {
  draw() {
    console.log("Drawing a circle.");
  }
}

class Rectangle extends Shape {
  draw() {
    console.log("Drawing a rectangle.");
  }
}

function renderShape(shape) {
  shape.draw(); // Polymorphic call: 'draw' behaves differently based on 'shape'
}

const myCircle = new Circle();
const myRectangle = new Rectangle();

renderShape(myCircle);
renderShape(myRectangle);
```