```
console.clear();

// 💻 Example 1 - Basic Prototype Link

// 'person' object with a greet method

const person = {

    greet() {

        console.log("Hello from person!");
    }
};

// 'student' object with a study method

const student = {

    study() {

        console.log("Studying hard...");
    }
};

// Set student's prototype to person

// This means 'student' will inherit all properties and methods of 'person'

student.__proto__ = person;

// Call methods

student.greet(); // Found in 'person' via prototype chain

student.study(); // Found directly in 'student'

// 🧠 Explanation:
```

```
38
39  // JavaScript looks for 'greet()' in 'student'.
40
41  // Since it's not there, it goes up the prototype chain to 'person' and finds it there.
42
43
44  // 💻 Example 2 – Using Object.create()
45
46  // Create a new object 'employee' whose prototype is 'person'
47
48  const employee = Object.create(person);
49
50  // Add a new method to 'employee'
51
52  employee.work = function () {
53
54      console.log("Working..");
55  };
56
57  // 'employee' can access both its own and its prototype's methods
58
59  employee.greet(); // Found in 'person'
60
61  employee.work();  // Found in 'employee'
62
63  // 🧠 Explanation:
64
65  // Object.create() creates a new object and directly links it to the specified prototype.
66
67
68  // 💻 Example 3 – Function Constructor Prototype Example
69
70  // Constructor function to create User objects
71
72  function User(name, age) {
73
74      this.name = name;
75
```

```
76      this.age = age;
77  }
78
79  // Add a shared method to User's prototype
80
81  // All instances of User will have access to this without duplicating it in memory
82
83  User.prototype.sayHi = function () {
84
85      console.log(`Hi, my name is ${this.name} and I'm ${this.age} years old.`);
86  };
87
88  // Create a new User instance
89
90  const user1 = new User("John", 30);
91
92  // Call the shared prototype method
93
94  user1.sayHi();
95
96  // 🧠 Explanation:
97
98  // 'sayHi' is not defined directly on 'user1', but found on User.prototype, which 'user1' inherits from.
99
100
101 // 💻 Example 4 – Prototype Chain Visualization
102
103 // Check if user1's prototype is the same as User.prototype
104
105 console.log(user1.__proto__ === User.prototype); // ✅ true
106
107 // Check if User.prototype's prototype is Object.prototype
108
109 console.log(User.prototype.__proto__ === Object.prototype); // ✅ true
110
111 // The end of the prototype chain is always null
112
113 console.log(Object.prototype.__proto__); // ✅ null
```

```
114
115 // 🧠 Explanation:
116
117 // Every object in JavaScript ultimately inherits from Object.prototype, which forms the base of the prototype chain.
118
119
120 // 💻 Example 5 – ES6 Class Syntax (Syntactic Sugar)
121
122 // Define a base class 'Animal'
123
124 class Animal {
125
126     constructor(name) {
127
128         this.name = name;
129     }
130
131     // Method available to all Animal instances
132
133     speak() {
134
135         console.log(`${this.name} makes a noise.`);
136     }
137 }
138
139 // Define a subclass 'Dog' that extends 'Animal'
140
141 class Dog extends Animal {
142
143     // Override the speak method from Animal
144
145     speak() {
146
147         console.log(`${this.name} barks!`);
148     }
149 }
150
151 // Create an instance of Dog
```

```
152
153 const dog = new Dog("Buddy");
154
155 // Call the overridden method
156
157 dog.speak(); // Output: Buddy barks!
158
159 // 🧠 Explanation:
160
161 // 'Dog' inherits from 'Animal' using 'extends'.
162
163 // The 'super' keyword (not used here) can call parent class methods.
164
165 // ES6 classes are just syntactic sugar over prototypes for cleaner, modern syntax.
```