Functions (regular, arrow, anonymous)

Exercise 1

Write a regular function `calculateDiscount` that takes `price` and `discountPercent` (e.g., 20 for 20%) and returns the discounted price. Create an arrow function `formatPrice` that takes a price and returns it as a string like "$X.XX". Use an anonymous function to log the result of `calculateDiscount(100, 25)` through `formatPrice`.

Exercise 2

Create an arrow function `createCounter` that returns a function. The returned function should increment a counter (starting at 0) each time it's called. Use a callback to log the counter value after calling it 3 times. Test it.

Arrays

Exercise 1

Create an array of 5 favorite books. Use `splice` to replace the second book with a new one and `push` to add another. Use `forEach` to log each book with its index (e.g., "1: BookName").

Example Output: 1: Book1, 2: NewBook, ..., 6: LastBook

Exercise 2

Write a function that takes an array of numbers, uses `slice` to get the last three elements, and calculates their product using a loop. Log the product. Test with `[2, 4, 6, 8, 10]`.

Example Output: 480 (6 * 8 * 10)

Objects

Exercise 1

Create an object for a movie with `title`, `year`, `genres` (array), and a method `info` returning "Title (Year): Genres". Update the `year` and add a `director` property. Call `info`.

Example Output: Movie (2023): Action, Drama

Exercise 2

Create a `playlist` object with a `songs` array and a method `addSong` that pushes a song and logs the updated array. Call `addSong` twice and use `Object.keys` to log all properties.

Example Output: ['Song1', 'Song2'], { songs: [...] }

Constructors and Classes

Exercise 1

Define a class `Vehicle` with `type` and `speed` properties and a method `move` returning "Type moves at Speed mph". Create a constructor `Bicycle` with the same properties/method. Create one instance of each and call `move`.

Example Output: Car moves at 60 mph, Bicycle moves at 15 mph

Exercise 2

Create a class `Pet` with `name` and `type`, and a method `describe` returning "Name is a Type". Extend it to `Dog` with `breed` and override `describe` to include breed. Create a `Dog` instance and call `describe`.

Example Output: Rover is a Dog, breed: Labrador

DOM Manipulation (Browser Required)

Exercise 1

Using `dom-manipulation.html`, write JS to create 4 `<span>` elements with texts "Item 1" to "Item 4". Append them to `#container`. Use `querySelectorAll` to select all `<span>`s and set their `fontWeight` to 'bold'.

Expected Result: Four bold items in the container.

Exercise 2

In `dom-manipulation.html`, add an event listener to `#myButton` that toggles the `display` style of `#text` between 'none' and 'block'. Log the current display state after each click.

Expected Result: Paragraph hides/shows on click, logs 'none' or 'block'.

Final Combined Exercise)

Create an employee directory:

- Define a class `Employee` with `name`, `role`, and a method `introduce` returning "Name, Role".
- Create a constructor `Department` with an `employees` array (empty initially).
- Add a prototype method `addEmployee` to `Department` that pushes an `Employee` instance.
- Write an arrow function `listRoles` that takes a `Department` instance and returns an array of all employee roles.
- Use an anonymous function with `forEach` to log each employee's `introduce` result.
- Create a `Department`, add 3 `Employee`s (e.g., "Alice, Manager", "Bob, Developer", "Eve, Designer").
- Log all roles using `listRoles`.

Example Output:
Alice, Manager
Bob, Developer
Eve, Designer
['Manager', 'Developer', 'Designer']