1. Nullish coalescing operator ( ?? )
   1. **Basics of ?? operator**

Write a JavaScript function called chooseValue that takes two arguments, x and y, and returns the value of x if it is not null or undefined; otherwise, it returns the value of y.

Here's the function signature to get you started:

function chooseValue(x, y) {

// Your code here

}

Your task is to complete the function so that it returns the desired result based on the conditions mentioned.

After writing the function, you can test it with various inputs to ensure it works as expected.

**TEST:**

console.log(chooseValue(null, "Hello"));

console.log(chooseValue(undefined, "World")); // Output: "World"

console.log(chooseValue("Goodbye", "Cruel World")); // Output: "Goodbye"

Source: ChatGPT

Answer: /js/1-1-nul-coal.js

* 1. **Using of ?? for objects**

Write a JavaScript function called setDefaults that takes an object and an optional set of default values. The function should use the nullish coalescing operator to set default values for object properties if they are not already defined.

Here's the function signature to get you started:

  function setDefaults(obj, defaults) {

    // Your code here

  }

The obj parameter is an object, and the defaults parameter is an object containing default values. Your task is to use the nullish coalescing operator to update properties of the obj with the corresponding values from the defaults object only if the properties in obj are null or undefined.

For example, if you call setDefaults with the following objects:

const obj = { name: "Alice", age: null, location: "Wonderland" };

const defaults = { age: 30, location: "Unknown" };

setDefaults(obj, defaults);

console.log(obj);

The setDefaults function should update the age property of obj to 30, but it should not update the location property because it is already defined in obj.

**TEST:**

  // Test 1

  const obj1 = { name: "Alice", age: null, location: "Wonderland" };

  const defaults1 = { age: 30, location: "Unknown" };

  setDefaults(obj1, defaults1);

  console.log(obj1);

  // Expected Output: { name: "Alice", age: 30, location: "Wonderland" }

  // Test 2

  const obj2 = { x: 42, y: undefined, z: null };

  const defaults2 = { x: 10, y: 20, z: 30 };

  setDefaults(obj2, defaults2);

  console.log(obj2);

  // Expected Output: { x: 42, y: 20, z: 30 }

  // Test 3

  const obj3 = { a: "Hello", b: "World", c: "!!!", d: null };

  const defaults3 = { a: "Goodbye", b: "Universe" };

  setDefaults(obj3, defaults3);

  console.log(obj3);

  // Expected Output: { a: "Hello", b: "World", c: "!!!", d: null }

Source: ChatGPT

Answer: /js/1-2-obj.js

* 1. **Using ?? Operator for Default Values**

Write a JavaScript function called getUserProfile() that takes an object representing a user's profile and returns a modified profile object with default values for missing properties. Use the nullish coalescing operator to set default values.

Here's the function signature to get you started:

function getUserProfile(userProfile) {

    // Your code here

}

The userProfile object can have properties like name, email, age, and so on. Your task is to complete the function by using the nullish coalescing operator to set default values for missing properties. For example, if the userProfile object doesn't have a name property, set it to "Unknown." If it doesn't have an age property, set it to 0.

After writing the function, you can test it with different userProfile objects to ensure it correctly sets default values for missing properties.

TESTS:

const userProfile1 = { name: "Alice", age: 30, email: "alice@example.com" };

const userProfile2 = { name: "Bob" };

const userProfile3 = { age: 25, email: "charlie@example.com" };

const userProfile4 = { name: "David", email: "david@example.com" };

console.log(getUserProfile(userProfile1));

// Expected Output: { name: "Alice", age: 30, email: "alice@example.com" }

console.log(getUserProfile(userProfile2));

// Expected Output: { name: "Bob", age: 0, email: "Unknown" }

console.log(getUserProfile(userProfile3));

// Expected Output: { name: "Unknown", age: 25, email: "charlie@example.com" }

console.log(getUserProfile(userProfile4));

// Expected Output: { name: "David", age: 0, email: "david@example.com" }

Source: ChatGPT

Answer: /js/1-3-user-data.js

* 1. **Calculate Total Price**

# Regular Expressions (REGEX)

* 1. **Validate an Email Address**

Write a JavaScript function called isValidEmail() that takes a string as input and returns true if the string is a valid email address and false otherwise. To validate an email address, you can use a regular expression pattern to check if it matches the standard email format.

Here's the function signature to get you started:

function isValidEmail(email) {

    // Your code here

}

Your task is to complete the function by using a regular expression to check if the provided email string follows the standard email format, which typically includes an email name, the "@" symbol, and a domain (e.g., "example@example.com").

For example:

isValidEmail("example@example.com") should return true because it's a valid email.

isValidEmail("invalid-email") should return false because it's not in a valid email format.

You can test the function with various email addresses to ensure it works correctly. Regular expressions are a powerful tool for such validation tasks.

TESTS:

console.log(isValidEmail("example@example.com")); // true

console.log(isValidEmail("user@domain.co.uk")); // true

console.log(isValidEmail("invalid-email")); // false

console.log(isValidEmail("no@tld.")); // false

console.log(isValidEmail("missingat.com")); // false

# Prototype chain and Static classes

**3-1. Create a Prototype Chain for animals**

Create the "Animal" Constructor:

1. Define a constructor function called "Animal" that takes two parameters: type and message. This constructor is the base for all animal objects.
2. Add a "makeSound" Method: Extend the "Animal" prototype by adding a makeSound method. This method logs the animal's sound message.
3. Create the "Cat" Constructor:

Create a constructor function called "Cat" for representing cats. In the constructor, inherit properties from the "Animal" constructor using Animal.call(this, "cat", "Meow!"), and set a custom property name specific to cats.

1. Create the "Dog" Constructor:

Create a constructor function called "Dog" for representing dogs. Similar to the "Cat" constructor, inherit properties from the "Animal" constructor using Animal.call(this, "dog", "WOOF!"), and set a custom property name for dogs.

1. Set Up Prototype Inheritance:

Establish prototype inheritance for both "Cat" and "Dog" constructors by using Object.create(Animal.prototype). This ensures that instances of these constructors inherit methods and properties from the "Animal" prototype.

1. Create Instances and Make Sounds:

Create instances of "Cat" and "Dog" (e.g., c1 and d1) with custom names.

Access and log properties such as name, type, and message for each instance.

Call the shared makeSound method on both instances to hear their respective sounds.

TESTS:

const c1 = new Cat("Alex");

console.log(c1.name, c1.type, c1.message);

c1.makeSound();

const d1 = new Dog("Snoopy");

console.log(d1.name, d1.type, d1.message);

d1.makeSound();

//

console.log("\nTesting:");

console.log("Is c1 an instance of Cat?", c1 instanceof Cat); // true

console.log("Is c1 an instance of Animal?", c1 instanceof Animal); // true

console.log("Is d1 an instance of Dog?", d1 instanceof Dog); // true

console.log("Is d1 an instance of Animal?", d1 instanceof Animal); // true

**3-2. Implement a Library Catalog**

Create a basic library catalog system using prototype-based programming in JavaScript. You'll define a base "Book" constructor and child constructors for different book genres. Each book should have a title, author, and a shared method for displaying information. Here are the steps:

1. Create a "Book" Constructor:

Define a constructor function called "Book" that takes parameters for the book's title and author. This constructor should set these properties.

1. Add a Prototype Method:

Extend the "Book" prototype by adding a method called displayInfo. This method should log information about the book, including its title and author.

1. Create Child Constructors:

Create child constructors for specific book genres, such as "Fiction" and "Non-Fiction." In these constructors, use Book.call(this, title, author) to inherit the title and author properties from the "Book" constructor.

1. Set Up Prototype Inheritance:

Use Object.create(Book.prototype) to establish prototype inheritance for the child constructors. This ensures that instances of "Fiction" and "Non-Fiction" inherit the displayInfo method from the "Book" prototype.

1. Create Instances:

Create instances of both "Fiction" and "Non-Fiction" books, each with a title and author.

1. Display Book Information and test the code:

Call the displayInfo method on the book instances to display their information.

TESTS:

const b1 = new Fiction("Fiction book title 1", "John Doe");

b1.displayInfo();

const b2 = new NonFiction("Non-fiction book title 1", "John Jackson");

b2.displayInfo();

// Additional Testing

console.log("Is b1 instance of Book?", b1 instanceof Book); // true

console.log("Is b1 instance of Non-Fiction?", b1 instanceof NonFiction); // false

console.log("Is b2 instance of Book?", b2 instanceof Book); // true

console.log("Is b2 instance of NonFiction?", b2 instanceof NonFiction); // true