**Coding Questions, Assignments and Projects.**

**VARIABLES & STRINGS:**

1.**Build a greeting Bot( getting to know about let, const ,console.log() and basic string usage**.

Add a console.log() statement that outputs the string "Hi there!" to the console.   
Now you should see the first message from the bot in the console. It is time to add a second message from the bot. Add another console.log statement to output the message "I am excited to talk to you." to the console.  
Use the let keyword to declare a variable called bot.  
**NOTE: You are using let here because later on in the code, you will be changing the value of the bot variable.**  
Use let to declare a variable named botLocation. Assign the string "teacherBot" to the bot variable and the string "the universe" to the botLocation variable.  
Add another console statement to the code that logs the message "Allow me to introduce myself."  
Create a variable called botIntroduction. Then use string concatenation with the + operator to join the string "My name is " followed by the bot variable followed by a period (.). Assign this value to the botIntroduction variable. Then, log the botIntroduction variable to the console.  
Create a variable called botLocationSentence. Then use string concatenation with the + operator to join the string "I live in " with the botLocation variable followed by a period (.). Assign this value to the botLocationSentence variable. Then, log the value of botLocationSentence to the console.  
Using reassignment, assign the string "professorBot" to the bot variable.   
Now it is time to see the new bot value. Start by creating a new variable called nicknameIntroduction. Use string concatenation to join the string "My nickname is " with the bot variable followed by a period (.). Assign the resulting string to the nicknameIntroduction variable. Then, log the value of nicknameIntroduction to the console.  
Using reassignment, assign the string "awesomeTeacherBot" to the bot variable.   
To see the bot's new nickname, you will need to log a new message to the console. Create a new variable called newNicknameGreeting. Then use string concatenation to join the string "I love my nickname but I wish people would call me " with the bot variable followed by a period. Assign the result to the newNicknameGreeting variable. Then, log the value of newNicknameGreeting to the console. The last few messages from the bot will focus on the bot's favorite subject.   
Start by creating a new variable called favoriteSubject and assign it the string "Computer Science". Next, create a variable called favoriteSubjectSentence. Use string concatenation to join the string "My favorite subject is " with the favoriteSubject variable followed by a period. Assign the result to the favoriteSubjectSentence variable. Then, log the value of favoriteSubjectSentence to the console. For the final step, you will log the bot's message of "Well, it was nice to talk to you. Have a nice day!" to the console.

Code:

console.log("Hi there!");

console.log("I am excited to talk to you.");

let bot;

let botLocation;

bot = "teacherBot";

botLocation = "the universe";

console.log("Allow me to introduce myself.");

const botIntroduction = "My name is " + bot + ".";

console.log(botIntroduction);

const botLocationSentence = "I live in " + botLocation + ".";

console.log(botLocationSentence);

bot = "professorBot";

const nicknameIntroduction = "My nickname is " + bot + ".";

console.log(nicknameIntroduction);

bot = "awesomeTeacherBot";

const newNicknameGreeting = "I love my nickname but I wish people would call me " + bot + ".";

console.log(newNicknameGreeting);

const favoriteSubject = "Computer Science";

const favoriteSubjectSentence = "My favorite subject is " + favoriteSubject + ".";

console.log(favoriteSubjectSentence);

console.log("Well, it was nice to talk to you. Have a nice day!”)

**2.Build a JavaScript Trivia Bot:**

1. You should log "Hello! I'm your coding fun fact guide!" to the console as a greeting message to the user.
2. You should create three variables: botName, botLocation, and favoriteLanguage, that store the bot's name, where it's from, and its favorite coding language, respectively.
3. You should use string concatenation to log "My name is (botName) and I live on (botLocation)." to the console.
4. You should use string concatenation to log "My favorite programming language is (favoriteLanguage)." to the console.
5. You should use let to create a codingFact variable and assign it a string that is a fun fact about your bot's favorite coding language and uses string concatenation to include the use of the favoriteLanguage variable.
6. You should log the codingFact to the console.
7. You should reassign the codingFact variable to a new fact about the bot's favorite language using the favoriteLanguage variable again.
8. You should log the codingFact to the console again.
9. You should reassign the codingFact variable again to another new fact about the bot's favorite language using the favoriteLanguage variable.
10. You should log the codingFact to the console a third time.
11. You should log "It was fun sharing these facts with you. Goodbye! - (botName) from (botLocation)." to the console as a farewell statement from the bot.

1. You should log "Hello! I'm your coding fun fact guide!" to the console.

2. You should declare a botName variable. Double check for any spelling or casing errors.

3. Your botName variable should be a string.

4. You should declare a botLocation variable. Double check for any spelling or casing errors.

5. Your botLocation variable should be a string.

6. You should declare a favoriteLanguage variable. Double check for any spelling or casing errors.

7. Your favoriteLanguage variable should be a string.

8. You should log to the console "My name is (botName) and I live on (botLocation)." using concatenation to add the variables to the string.

9. You should log to the console "My favorite programming language is (favoriteLanguage)." using concatenation to add the variable to the string.

10. You should use let to declare a new variable codingFact.

11. You should give codingFact a value that includes favoriteLanguage using concatenation.

12. You should log codingFact to the console.

13. You should assign a new value to codingFact that also contains favoriteLanguage, and log it to the console.

14. You should assign a value to codingFact for the third time that also contains favoriteLanguage, and log it to the console.

15. You should log to the console "It was fun sharing these facts with you. Goodbye! - (botName) from (botLocation)." using concatenation to add the values of the variables.

**Code:**

console.log("Hello! I'm your coding fun fact guide!");

let botName = "Bot1";

let botLocation = "Bangalore";

let favoriteLanguage = "JavaScript";

let codingFact = "It is fun to code with "+ favoriteLanguage;

console.log("My name is " + botName +" and I live on " + botLocation + ".");

console.log("My favorite programming language is " + favoriteLanguage + ".");

console.log(codingFact);

codingFact = favoriteLanguage + " Is more fun";

console.log(codingFact);

codingFact = favoriteLanguage + " I'm getting excited.";

console.log(codingFact);

console.log("It was fun sharing these facts with you. Goodbye! - "+ botName + " from " + botLocation + ".");

**3. Build a Sentence maker:**

You will create two different stories using a sentence template. You will use variables to store different parts of the story and then output the stories to the console.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should declare the following variables using let:
   * adjective
   * noun
   * verb
   * place
   * adjective2
   * noun2
2. You should assign the above variables some string values of your choice.
3. You should declare a firstStory variable.
4. You should use the following story template to create the first story and assign it to the firstStory variable: "Once upon a time, there was a(n) [adjective] [noun] who loved to eat [noun2]. The [noun] lived in a [place] and had [adjective2] nostrils that blew fire when it was [verb].";
5. You should output your first story to the console using the message "First story: [firstStory]".
6. You should assign new values to your adjective, noun, verb, place, adjective2, and noun2 variables.
7. You should declare a secondStory variable.
8. Create another story using the same template and assign it to the secondStory variable.
9. You should output your second story to the console using the message "Second story: [secondStory]".

1. You should declare an adjective variable.

2. You should declare a noun variable.

3. You should declare a verb variable.

4. You should declare a place variable.

5. You should declare an adjective2 variable.

6. You should declare a noun2 variable.

7. You should assign a string value to the adjective variable.

8. You should assign a string value to the noun variable.

9. You should assign a string value to the verb variable.

10. You should assign a string value to the place variable.

11. You should assign a string value to the adjective2 variable.

12. You should assign a string value to the noun2 variable.

13. You should declare a firstStory variable.

14. You should use the correct story format for the first story: "Once upon a time, there was a(n) [adjective] [noun] who loved to eat [noun2]. The [noun] lived in a [place] and had [adjective2] nostrils that blew fire when it was [verb].". Pay attention to spaces.

15. You should log your first story using the message "First story: [firstStory]".

16. You should declare a secondStory variable.

17. You should reassign the adjective variable for the second story.

18. You should reassign the noun variable for the second story.

19. You should reassign the verb variable for the second story.

20. You should reassign the place variable for the second story.

21. You should reassign the adjective2 variable for the second story.

22. You should reassign the noun2 variable for the second story.

23. You should use the correct story format for the second story: "Once upon a time, there was a(n) [adjective] [noun] who loved to eat [noun2]. The [noun] lived in a [place] and had [adjective2] nostrils that blew fire when it was [verb].". Pay attention to spaces.

24. You should log your second story using the format "Second story: [secondStory]".

25. The firstStory should not be the same as the secondStory.

**Code:**

let adjective, noun, verb, place, adjective2, noun2;

adjective ="happy";

noun = "cat";

verb = "sleeping";

place = "small town";

adjective2 = "Big";

noun2 = "Bread";

let firstStory;

firstStory = "Once upon a time, there was a(n) " + adjective + " "+ noun + " who loved to eat " + noun2 + ". The " + noun + " lived in a " + place + " and had " + adjective2 + " nostrils that blew fire when it was " + verb + ".";

console.log("First story: " + firstStory);

let secondStory;

adjective ="sad";

noun = "dog";

verb = "barking";

place = "village";

adjective2 = "small";

noun2 = "Meat";

secondStory = "Once upon a time, there was a(n) " + adjective +" "+ noun + " who loved to eat " + noun2 + ". The " + noun + " lived in a " + place + " and had " + adjective2 + " nostrils that blew fire when it was " + verb + ".";

console.log("Second story: " + secondStory);

**Build a teacher Bot:**

To begin, add a console statement, with the message of "Hi there!". Create a variable called botName and assign it the string value of "teacherBot".   
Start by creating a variable called greeting. Next, using template literal syntax, assign a string that says My name is, followed by the botName variable, and ending with a period (.). Finally, log the greeting variable to the console.  
Create a variable called subject and assign it the string value "JavaScript". Then create a variable called topic and assign it the string value "strings". Start by creating a sentence variable. Using template literal syntax, assign the string “Today, you will learn about [topic variable goes here] in [subject variable goes here]”. to the sentence variable. You will replace the [topic variable goes here] and [subject variable goes here] placeholders with the topic and subject variables and ${} syntax. Finally, log the sentence variable to the console.  
For this next portion of the project, the bot will teach working with the string length property. Start by creating a new variable called strLengthIntro. Then using template literal syntax, assign the string Here is an example of using the length property on the word [subject]. to the strLengthIntro variable. Replace [subject] with the subject variable like you did earlier. Finally, log the strLengthIntro variable to the console.  
Start by using the length property to get the length of the subject string and log that value to the console. Start by outputting the message Here is an example of using the length property on the word [topic]. to the console. Remember to replace [topic] with the topic variable and use proper template literal syntax as you did in the previous steps. Then, add a second console.log statement that outputs the length of the topic string to the console.  
Start by outputting the message Here is an example of accessing the first letter in the word [subject]. to the console. Remember to replace [subject] with the subject variable and use proper template literal syntax like you did in the previous steps.  
Start by adding another console statement. Inside the console statement, output the first letter of the subject variable using bracket notation and the correct index number. Now it is time to access the second letter of the subject variable. Start by adding a console statement that outputs the message Here is an example of accessing the second letter in the word [subject]. Remember to replace [subject] with the actual value of the subject variable and use correct template literal syntax. Then add another console statement that outputs the second letter of the subject variable using bracket notation and the correct index number. Now it is time to access the last character of a string. Start by adding another console statement that outputs the message Here is an example of accessing the last letter in the word [subject]. Remember to replace [subject] with the actual value of the subject variable and use correct template literal syntax. Create a new variable called lastCharacter and assign it the value of the last character in the subject variable. Then, log the value of the lastCharacter variable to the console.  
Start by creating a variable called learningIsFunSentence and assign it the string value of "Learning is fun.". The next step is to add another console statement that outputs the string "Here are examples of finding the positions of substrings in the sentence." Add a new console statement that outputs the result of using the indexOf method on the learningIsFunSentence variable to find the index position of the substring "Learning".  
Next, you will add a new console statement that outputs the result of using the indexOf method on the learningIsFunSentence variable to find the position of the substring "fun". Below that console statement, add a new console statement that outputs the result of using the indexOf method to find the position of the substring "learning". Take note of what the last console statement outputs.  
Add a console statement that outputs the message "I hope you enjoyed learning today." to the console.

**Code:**console.log("Hi there!");  
const botName = "teacherBot";  
const greeting = `My name is ${botName}.`;  
console.log(greeting);  
const subject = "JavaScript";  
const topic = "strings";  
const sentence = `Today, you will learn about ${topic} in ${subject}.`;  
console.log(sentence);  
const strLengthIntro = `Here is an example of using the length property on the word ${subject}.`;  
console.log(strLengthIntro);  
console.log(subject.length);  
console.log(`Here is an example of using the length property on the word ${topic}.`);  
console.log(topic.length);  
console.log(`Here is an example of accessing the first letter in the word ${subject}.`);  
console.log(subject[0]);  
console.log(`Here is an example of accessing the second letter in the word ${subject}.`);  
console.log(subject[1]);  
console.log(`Here is an example of accessing the last letter in the word ${subject}.`);  
const lastCharacter = subject[subject.length - 1];  
console.log(lastCharacter);  
const learningIsFunSentence = "Learning is fun.";  
console.log("Here are examples of finding the positions of substrings in the sentence.");  
console.log(learningIsFunSentence.indexOf("Learning"));  
console.log(learningIsFunSentence.indexOf("fun"));  
console.log(learningIsFunSentence.indexOf("learning"));  
console.log("I hope you enjoyed learning today.");

**Build a MathBot:**

Step 1: First step, start by creating a variable called botName and assign it the string value of "MathBot". Then, create another variable called greeting and assign it the sentence "Hi there! My name is [botName goes here] and I am here to teach you about the Math object!". In place of the [botName goes here] placeholder, use the botName variable. You are free to use template literals or string concatenation with the + operator to achieve this. Finally, log the greeting variable to the console.  
Step 2 : Start by adding another console.log() that logs the string "The Math.random() method returns a pseudo random number between 0 and less than 1." to the console.  
Step 3: Now, it is time to generate a random number using the Math.random() method. Create a variable called randomNum and assign it the value of the result of calling the Math.random() method. Then, log the randomNum variable to the console.  
Step 4: Start by adding another console.log() that logs the message "Now, generate a random number between two values." to the console. Then, create a variable called min and assign it the value of 1 and create a variable called max and assign it the value of 100. In the next step, you will generate a random number between these two values.  
Step 5: Create a variable called randomNum2 and assign it the result of generating a value between the min and max values. Then, log the randomNum2 variable to see the result. Try refreshing the page to see different results.  
Step 6: Start by adding a console.log() that logs the message "The Math.floor() method rounds the value down to the nearest whole integer."  
Step 7: Create a variable called numRoundedDown and assign it the result of rounding the floating point number 6.7 down to the nearest whole integer. Then, log the numRoundedDown variable to the console to see the result.  
Step 8: Start by logging the string "The Math.ceil() method rounds the value up to the nearest whole integer." to the console.  
Step 9: Create a variable called numRoundedUp and assign it the result of rounding the floating point number 3.2 up to the nearest whole integer. Then, log the numRoundedUp variable to the console to see the result.  
Step 10: Start by adding a console.log() that logs the message "The Math.round() method rounds the value to the nearest whole integer." to the console.  
Step 11: Create a new variable called numRounded and assign the result of rounding the number 2.7. Then, log the value of numRounded to the console. Below that, create another new variable called numRounded2 and assign the result of rounding the number 11.2. Then, log the value of numRounded2 to the console.  
Step 12: Start by adding a console.log() that logs the message "The Math.max() and Math.min() methods are used to get the maximum and minimum number from a range." to the console.  
Step 13: Create a variable called maxNum and assign it the result of finding the maximum number between the numbers 3, 125, 55, 24. Then, log the value of maxNum to the console. Below that, create a variable called minNum and assign it the result of finding the minimum number between the numbers 6, 90, 14, 90, 2. Then, log the value of minNum to the console.

**Code:**

const botName = "MathBot";

const greeting = `Hi there! My name is ${botName} and I am here to teach you about the Math object!`;

console.log(greeting);

console.log("The Math.random() method returns a pseudo random number between 0 and less than 1.");

const randomNum = Math.random();

console.log(randomNum);

console.log("Now, generate a random number between two values.");

const min = 1;

const max = 100;

const randomNum2 = Math.random() \* (max - min) + min;

console.log(randomNum2);

console.log("The Math.floor() method rounds the value down to the nearest whole integer.");

const numRoundedDown = Math.floor(6.7);

console.log(numRoundedDown);

console.log("The Math.ceil() method rounds the value up to the nearest whole integer.");

const numRoundedUp = Math.ceil(3.2);

console.log(numRoundedUp);

console.log(

  "The Math.round() method rounds the value to the nearest whole integer."

);

const numRounded = Math.round(2.7);

console.log(numRounded);

const numRounded2 = Math.round(11.2);

console.log(numRounded2);

console.log("The Math.max() and Math.min() methods are used to get the maximum and minimum number from a range.");

const maxNum = Math.max(3, 125, 55, 24);

console.log(maxNum);

const minNum = Math.min(6, 90, 14, 90, 2);

console.log(minNum);

**Buid a Fortune Teller:**

In this question, you will define five fortunes and randomly select one of them to display the fortune to the user.

**User Stories:**

1. You should initialize the five variables fortune1, fortune2, fortune3, fortune4, and fortune5 with a string value of your choice. You can use the below options if you like:
   * "Your cat will look very cuddly today."
   * "The weather will be nice tomorrow."
   * "Be cautious of your new neighbors."
   * "You will find a new hobby soon."
   * "It would be wise to avoid the color red today."
2. You should select a random number between 1 and 5, inclusive, and assign it to the variable randomNumber.
3. You should create a selectedFortune variable and assign the appropriate fortune based on these rules:
   * If randomNumber is 1, assign the value of fortune1 to selectedFortune.
   * If randomNumber is 2, assign the value of fortune2 to selectedFortune.
   * If randomNumber is 3, assign the value of fortune3 to selectedFortune.
   * If randomNumber is 4, assign the value of fortune4 to selectedFortune.
   * If randomNumber is 5, assign the value of fortune5 to selectedFortune.
4. You should log the selectedFortune to the console.

* 1. You should initialize fortune1 with a string value.
* 2. You should initialize fortune2 with a string value.
* 3. You should initialize fortune3 with a string value.
* 4. You should initialize fortune4 with a string value.
* 5. You should initialize fortune5 with a string value.
* 6. You should use the Math.random() method to generate a random number.
* 7. You should generate a random number between 1 and 5, inclusive, and assign it to the variable randomNumber.
* 8. You should have a selectedFortune variable that is assigned a value based on the value of randomNumber.
* 9. The randomNumber should correspond to its fortune. For example, if randomNumber is 1, the selectedFortune should be equal to fortune1 and so on.
* 10. You should output the selectedFortune to the console.

**Code:**

let fortune1 = "Your cat will look very cuddly today.";

let fortune2 = "The weather will be nice tomorrow.";

let fortune3 = "Be cautious of your new neighbors.";

let fortune4 = "You will find a new hobby soon.";

let fortune5 = "It would be wise to avoid the color red today.";

let randomNumber = Math.floor(Math.random()\*(5-1))+1;

let selectedFortune;

if (randomNumber==1){

  selectedFortune = fortune1;

}

else if (randomNumber==2){

  selectedFortune = fortune2;

}

else if (randomNumber==3){

  selectedFortune = fortune3;

}

else if (randomNumber==4){

  selectedFortune = fortune4;

}

else{

  selectedFortune = fortune5;

}

console.log(selectedFortune);

**Build A Calculator:**

Start by creating a function called addTwoAndSeven. You can choose to use the regular function syntax or the arrow function syntax.  
Inside your addTwoAndSeven function, return the sum of 2 and 7.  
Add a console.log statement, and inside it, call the addTwoAndSeven function. You should now see the sum of 2 and 7 logged to the console.  
Now, it is time to add another function. Declare a function called addThreeAndFour that returns the sum of 3 and 4. Then call the addThreeAndFour function inside a console.log to see the result.  
Declare a function called calculateSum that takes two parameters, num1 and num2.  
Inside your calculateSum function, you will need to return the sum of the two parameters.  
Add a console.log that calls the calculateSum function with the arguments 2 and 5.  
Add another console.log that calls the calculateSum function with the arguments 10 and 10. Below that console.log, add another console.log that calls the calculateSum function with the arguments 5 and 5.  
Now that your calculator can add two numbers together, it is time to create a function that will subtract two numbers. Declare a function called calculateDifference that takes two parameters, num1 and num2. Inside the function, return the difference between the two parameters.  
Start by adding a console.log that calls the calculateDifference function with the arguments 22 and 5. Then, add a second console.log that calls the calculateDifference function with the arguments 12 and 1. Finally, add a third console.log that calls the calculateDifference function with the arguments 17 and 9.  
Now it is time to add the multiplication functionality to your calculator. Declare a function called calculateProduct that takes two parameters, num1 and num2. Inside the function, return the product of the two parameters. Then, add a console.log that calls the calculateProduct function with the arguments 13 and 5.  
The next step is to add the division functionality to your calculator. Declare a function called calculateQuotient that takes two parameters, num1 and num2. Inside the function, return the quotient of the two parameters. Then, add a console.log that calls the calculateQuotient function with the arguments 7 and 11.  
Add a console.log that calls the calculateQuotient function with the arguments 3 and 0. Make sure to take a close look at the output of this call.  
If you look in the console, you will see the Infinity value. Infinity is a special value in JavaScript that represents a number that is greater than any other number. The division by zero is not a valid operation in mathematics. To account for this edge case, you should update your calculateQuotient function to instead check if num2 is zero. If it is, the function should return the string "Error: Division by zero". Otherwise, it should return the result of dividing num1 by num2.  
It would be nice to have your calculator calculate the square of a number. The square of a number is the number multiplied by itself. To calculate the square of a number in JavaScript, you can use the Math.pow() method. The Math.pow() method takes two arguments: the base number and the exponent. You can also use the exponentiation operator (\*\*) to calculate the square of a number. Declare a function called calculateSquare that takes a num parameter and returns the square of num.  
Add a console.log that calls the calculateSquare function with the argument of 2. Then, add another console.log that calls the calculateSquare function with the argument of 9.  
Declare a function called calculateSquareRoot that takes a num parameter and returns the square root of num.  
Add a second console.log that calls the calculateSquareRoot function with the argument of 25. Then, add a third console.log that calls the calculateSquareRoot function with the argument of 100. And with those changes, your calculator app is complete!

**Code:**

function calculateSum(num1, num2) {

  return num1 + num2;

}

console.log(calculateSum(2, 5));

console.log(calculateSum(10, 10));

console.log(calculateSum(5, 5));

function calculateDifference(num1, num2) {

  return num1 - num2;

}

console.log(calculateDifference(22, 5));

console.log(calculateDifference(12, 1));

console.log(calculateDifference(17, 9));

function calculateProduct(num1, num2) {

  return num1 \* num2;

}

console.log(calculateProduct(13, 5));

function calculateQuotient(num1, num2) {

  return num2 === 0 ? "Error: Division by zero" : num1 / num2;

}

console.log(calculateQuotient(7, 11));

console.log(calculateQuotient(3, 0));

function calculateSquare(num) {

  return num \*\* 2;

}

console.log(calculateSquare(2));

console.log(calculateSquare(9));

function calculateSquareRoot(num) {

  return Math.sqrt(num);

}

console.log(calculateSquareRoot(25));

console.log(calculateSquareRoot(100));

**Build a Boolean Check Function**

In this lab you will build a function that check if a value is classified as a boolean primitive.

Boolean primitives are true and false.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a function called booWho that receives one argument.
2. If the argument received is a boolean primitive, the function should return true.
3. If the argument is any other value, the function should return false.

* 1. You should have a booWho function.
* 2. booWho(true) should return true.
* 3. booWho(false) should return true.
* 4. booWho([1, 2, 3]) should return false.
* 5. booWho([].slice) should return false.
* 6. booWho({ "a": 1 }) should return false.
* 7. booWho(1) should return false.
* 8. booWho(NaN) should return false.
* 9. booWho("a") should return false.
* 10. booWho("true") should return false.
* 11. booWho("false") should return false.

**Code:**

function booWho(value) {

  return typeof value === "boolean";

}

console.log(booWho(true));

console.log(booWho(false));

console.log(booWho([1, 2, 3]));

console.log(booWho([].slice));

console.log(booWho({ "a": 1 }));

console.log(booWho(1));

console.log(booWho(NaN));

console.log(booWho("a"));

console.log(booWho("true"));

console.log(booWho("false"));

**Build an Email Masker:**

In this question, you will mask the username part of an email address with asterisks. Masking is a term used to hide or replace sensitive information with asterisks or other characters.

For example, if the email address was myEmail@email.com, then the masked email address will be m\*\*\*\*\*l@email.com.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**1.Create a function named maskEmail that takes email as an argument.  
2.Inside the function, you should mask the email and append the domain name to it. Remember that you can use methods like slice, repeat, indexOf or even replace to help you.  
3.Outside the function, declare a variable named email to store the email address you want to mask.  
4.Call the maskEmail function with the email variable and output the result to the console.  
5.maskEmail("apple.pie@example.com") should return "a\*\*\*\*\*\*\*e@example.com".  
6.maskEmail("skillacademy@example.com") should return "s\*\*\*\*\*\*\*\*\*\*y@example.com".  
7.maskEmail("info@test.dev") should return "i\*\*o@test.dev".  
8.maskEmail("user@domain.org") should return "u\*\*r@domain.org".  
  
You should define a function named maskEmail.  
The maskEmail function should take a string, email as an argument.  
Outside the function, you should have an email variable.  
 You should assign a valid email address to your email variable.  
 maskEmail("apple.pie@example.com") should return "a\*\*\*\*\*\*\*e@example.com".  
 maskEmail("skillacademy@example.com") should return "s\*\*\*\*\*\*\*\*\*\*y@example.com".  
 maskEmail("info@test.dev") should return "i\*\*o@test.dev".  
maskEmail("user@domain.org") should return "u\*\*r@domain.org".  
Your maskEmail should produce the correct result.  
 You should log the output of calling maskEmail with email as argument.

**Code:**

function maskEmail(email){

  let sign = email.indexOf("@");

  let lc = email.slice(sign-1);

  let fc = email.charAt(0);

  let domain = email.slice(1,sign);

  let mask = "\*";

  let maskL = domain.length-1;

  let repeating = mask.repeat(maskL);

  let replaced = domain.replace(domain,repeating);

  return fc + replaced + lc;

}

let email = "apple.pie@example.com";

console.log(maskEmail(email));

email = "skillacademy@example.com";

console.log(maskEmail(email));

email = "info@test.dev";

console.log(maskEmail(email));

email = "user@domain.org";

console.log(maskEmail(email));

**Build a Loan Qualification Checker:**

In this question, you'll review JavaScript conditionals by building a loan qualification checker app.

The app will check whether the user is eligible for a duplex, car, and condo loan based on their annual income and credit score. A credit score is a number that represents how good you are at managing borrowed money.

To get started, create the following variables and values.

| **Variable Name** | **Value** |
| --- | --- |
| minIncomeForDuplex | 60000 |
| minCreditScoreForDuplex | 700 |
| minIncomeForCondo | 45000 |
| minCreditScoreForCondo | 680 |
| minIncomeForCar | 30000 |
| minCreditScoreForCar | 650 |

When the user is eligible for a loan, you'll want to display a message to them in the console. For that, you'll build out a function inside which you'll have some checks that'll return what loan the applicant is eligible for. Create an empty getLoanMessage function with an annualIncome and creditScore parameters.  
  
To check which loan a user is qualified for based on the annualIncome and creditScore, you have to use if/else if statement or a ternary right inside the getLoanMessage function. You'll then return the appropriate message in the block of each condition. Starting with the duplex loan, check if annualIncome is greater than or equal to minIncomeForDuplex AND if creditScore is greater than or equal to minCreditScoreForDuplex. If that condition is true, then the applicant is eligible for a duplex loan and the other loans. So, inside the check, return the string "You qualify for a duplex, condo, and car loan."  
  
If the applicant's annual income is greater than or equal to minIncomeForCondo, AND if their credit score is greater than or equal to minCreditScoreForCondo, then they qualify for a condo and car loan. Check if that's true in the getLoanMessage function. If it is, return the string "You qualify for a condo and car loan."  
  
Now, you should check if the applicant is qualified for a car loan only. If they are, return the string "You qualify for a car loan.".  
  
If the applicant's annual income and credit score fall below minIncomeForCar and minCreditScoreForCar, then they don't qualify for any loan. So, return the string "You don't qualify for any loans."  
  
Now, it is time to test out your getLoanMessage function. Use the table below to create 4 variables and their values:

| **Variable Name** | **Value** |
| --- | --- |
| duplexLoanMsg | getLoanMessage(85000, 850) |
| condoLoanMsg | getLoanMessage(65000, 690) |
| carLoanMsg | getLoanMessage(45000, 660) |
| noLoanMsg | getLoanMessage(25000, 550) |

After that, log each variable to the console to see the messages. With that, your loan qualification checker project is complete!

**Code:**

const minIncomeForDuplex = 60000;

const minCreditScoreForDuplex = 700;

const minIncomeForCondo = 45000;

const minCreditScoreForCondo = 680;

const minIncomeForCar = 30000;

const minCreditScoreForCar = 650;

function getLoanMessage(annualIncome, creditScore) {

  if(creditScore >= minCreditScoreForDuplex && annualIncome >= minIncomeForDuplex) {

    return "You qualify for a duplex, condo, and car loan."

  } else if (annualIncome >= minIncomeForCondo && creditScore >= minCreditScoreForCondo) {

    return "You qualify for a condo and car loan."

  } else if (annualIncome >= minIncomeForCar && creditScore >= minCreditScoreForCar) {

    return "You qualify for a car loan."

  } else {

    return "You don't qualify for any loans."

  }

}

let duplexLoanMsg = getLoanMessage(85000, 850);

let condoLoanMsg = getLoanMessage(65000, 690);

let carLoanMsg = getLoanMessage(45000, 660);

let noLoanMsg = getLoanMessage(25000, 550);

console.log(duplexLoanMsg);

console.log(condoLoanMsg);

console.log(carLoanMsg);

console.log(noLoanMsg);

**Build a Celsius to Fahrenheit Converter**

In this lab, you will write a function that converts the temperature from Celsius to Fahrenheit. The formula to convert from Celsius to Fahrenheit is:  
Example code:   
fahrenheit = celsius \* (9/5) + 32  
  
**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a function named convertCtoF.
2. The convertCtoF should take a single numeric argument, which is the temperature in Celsius.
3. convertCtoF should return a number.

* 1. You should create a function named convertCtoF.
* 2. convertCtoF should take a single parameter.
* 3. convertCtoF(0) should return a number
* 4. convertCtoF(-30) should return a value of -22
* 5. convertCtoF(-10) should return a value of 14
* 6. convertCtoF(0) should return a value of 32
* 7. convertCtoF(20) should return a value of 68
* 8. convertCtoF(30) should return a value of 86

**Code:**

function convertCtoF(celsius) {

  return celsius \* (9 / 5) + 32;

}

console.log(convertCtoF(0));    // 32

console.log(convertCtoF(-30));  // -22

console.log(convertCtoF(-10));  // 14

console.log(convertCtoF(20));   // 68

console.log(convertCtoF(30));   // 86

**Build a Leap Year Calculator:**

A leap year is a year that is divisible by 4, except for years that are divisible by 100 and not divisible by 400. For example, 2000 is a leap year, but 1900 is not. Also, a leap year has an extra day in February, which is the 29th day of the month.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. Define a function called isLeapYear that takes a number as an argument.
2. Outside the function, declare a variable year that stores the value of the year you want to check.
3. Inside the function, use an if/ else statement or a ternary operator to check if the year is a leap year.
4. To check if the year is a leap year, fulfill the following conditions:
   * If the year is divisible by 4, then it is a leap year.
   * Unless the year is also divisible by 100, then it is not a leap year.
   * Unless the year is also divisible by 400, then it is a leap year.
5. If the year is a leap year, return [year] is a leap year.. Otherwise, return [year] is not a leap year.. You will replace [year] with the parameter defined in the isLeapYear function.
6. You should call the isLeapYear function with year as the argument and assign the result to a variable named result.
7. You should output the result variable to the console using console.log().

* 1. You should define a function named isLeapYear.
* 2. The isLeapYear function should take a number as an argument.
* 3. You should declare a variable year and assign it a value to check if it is a leap year.
* 4. The year variable shouldn't be empty.
* 5. With 2024 as the value of the year variable, the result should be 2024 is a leap year..
* 6. With 2000 as the value of the year variable, the result should be 2000 is a leap year..
* 7. With 1900 as the value of the year variable, the result should be 1900 is not a leap year..
* 8. You should call the isLeapYear function and pass year as a parameter.
* 9. You should declare a result variable.
* 10. You should store the result of calling the isLeapYear function in a variable named result.
* 11. You should output the result to the console using console.log().

**Code:**

let year = 1900;

function isLeapYear(year) {

  if (year % 4 == 0 && year % 400 == 0)  {

   return`${year} is a leap year.`

  } else if (year % 4==0  && year % 100==0) {

    return`${year} is not a leap year.`

  } else if (year % 4==0  && year % 100!=0) {

    return `${year} is a leap year.`

  } else {

    return `${year} is not a leap year.`

  };

};

let result = isLeapYear(year);

console.log(result);

**Implement the Truncate a String Algorithm:**

In this question, you will practice truncating a string to a certain length.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a function truncateString that accepts two arguments, the first one a string, the second one a number.
2. If the length of the string is more than the given number, the string should be truncated to reduce the length so that it is equal the given number, and ... should be appended at the end of the truncated string.
3. If the length of the string is equal to or lower than the given number, the string should be returned unchanged.

* 1. truncateString("A-tisket a-tasket A green and yellow basket", 8) should return the string A-tisket....
* 2. truncateString("Peter Piper picked a peck of pickled peppers", 11) should return the string Peter Piper....
* 3. truncateString("A-tisket a-tasket A green and yellow basket", "A-tisket a-tasket A green and yellow basket".length) should return the string A-tisket a-tasket A green and yellow basket.
* 4. truncateString("A-tisket a-tasket A green and yellow basket", "A-tisket a-tasket A green and yellow basket".length + 2) should return the string A-tisket a-tasket A green and yellow basket.
* 5. truncateString("A-", 1) should return the string A....
* 6. truncateString("Absolutely Longer", 2) should return the string Ab....

**Code:**

function truncateString(str, num) {

// Clear out that junk in your trunk

if (str.length > num) {

return str.slice(0, num) + '...';

} else if (str.length > num && num <= 3) {

return str.slice(0, num) + '...';

} else {

return str;

}

}

truncateString("A-tisket a-tasket A green and yellow basket", "A-tisket a-tasket A green and yellow basket".length);

**Build a Confirm the Ending Tool**

In this lab, you will implement a function that checks if a string ends with the given target string.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a function named confirmEnding that takes two parameters: the string to check and the string to check against.
2. The function should return true if the first string ends with the second string, and false otherwise.
3. You should not use the .endsWith() method; instead, use one of the JavaScript substring methods to achieve this.

* 1. You should create a function named confirmEnding.
* 2. confirmEnding should take 2 parameters.
* 3. confirmEnding("Bastian", "n") should return true.
* 4. confirmEnding("Congratulation", "on") should return true.
* 5. confirmEnding("Connor", "n") should return false.
* 6. confirmEnding("Walking on water and developing software from a specification are easy if both are frozen", "specification") should return false.
* 7. confirmEnding("He has to give me a new name", "name") should return true.
* 8. confirmEnding("Open sesame", "same") should return true.
* 9. confirmEnding("Open sesame", "sage") should return false.
* 10. confirmEnding("Open sesame", "game") should return false.
* 11. confirmEnding("If you want to save our world, you must hurry. We dont know how much longer we can withstand the nothing", "mountain") should return false.
* 12. confirmEnding("Abstraction", "action") should return true.
* 13. Your code should not use the built-in method .endsWith() to solve the lab.

**Code:**

function confirmEnding(str, target) {

  return str.substring(str.length - target.length) === target;

}

console.log(confirmEnding("Bastian", "n")); // true

console.log(confirmEnding("Congratulation", "on")); // true

console.log(confirmEnding("Connor", "n")); // false

console.log(confirmEnding("Walking on water and developing software from a specification are easy if both are frozen", "specification")); // false

console.log(confirmEnding("He has to give me a new name", "name")); // true

console.log(confirmEnding("Open sesame", "same")); // true

console.log(confirmEnding("Open sesame", "sage")); // false

console.log(confirmEnding("Open sesame", "game")); // false

console.log(confirmEnding("If you want to save our world, you must hurry. We dont know how much longer we can withstand the nothing", "mountain")); // false

console.log(confirmEnding("Abstraction", "action")); // true

**Build a shopping list:**

1.In this question, you will continue to learn about arrays by building a grocery shopping list. Start by adding a console.log that logs the string "Grocery shopping list" to the console.  
2. In this step, create a variable called shoppingList and assign it an empty array.  
3. Start by using console.log to log the message "It will be nice to have some fruit to eat.".  
4. Using the push method, add the string "Apples" to the shoppingList array.  
5. Create a function called getShoppingListMsg that takes an array as a parameter and returns the string "Current Shopping List: " followed by the contents of the provided array. You can use template literals or string concatenation with the + operator to combine the string and the shoppingList array.  
6. Now it is time to see the message logged to the console. Add a console.log and call the getShoppingListMsg function inside of the console.log to see the message logged to the console.  
7. Now it is time to add another fruit to the list. Using the same array method as earlier, add the string "Grapes" to the end of the shoppingList array. Then, add a console.log and call the getShoppingListMsg function inside of the console.log to see the updated list logged to the console.  
8. Now it is time to start adding items to the top of the grocery list. Start by adding a console.log that logs the message "It looks like we need to get some cooking oil." to the console.  
9. Use the unshift() method to add the string "Vegetable Oil" to the beginning of the shoppingList array.  
10. Next, add a console.log and call the getShoppingListMsg function inside of the console.log to see the updated list logged to the console.  
11. In this step, use the push method to add the strings "Popcorn", "Beef Jerky", "Potato Chips" to the shoppingList array. The order is important, so make sure to add the items in the order they are listed.  
12. Now it is time to log the updated shoppingList array to the console. Add another console.log and call the getShoppingListMsg function inside of the console.log to see the updated list logged to the console.  
13. Start by adding a console.log that logs the message "This looks like too much junk food.".  
14. Use the pop method to remove the last item from the shoppingList array.  
15. Now it is time to log the updated shoppingList array to the console. Add a console.log and call the getShoppingListMsg function inside of the console.log to see the updated list logged to the console.  
16. Now it is time to add more items to the beginning of the grocery list. Start by adding a console.log statement that logs the message "It might be nice to get a dessert." Below that console statement, use the correct array method to add the string "Chocolate Cake" to the beginning of the shoppingList array. Finally, add a console.log and call the getShoppingListMsg function inside of the console.log to see the updated list logged to the console.  
17. Start by adding a console.log that logs the message "On second thought, maybe we should be more health conscious.".  
18. Use the shift method to remove the first item from the shoppingList array.  
19. Update the first item in the shoppingList array to be "Canola Oil".  
20. Call the getShoppingListMsg function with the shoppingList array as an argument inside console.log. And with this last step your grocery list is complete!

Code:

console.log("Grocery shopping list");

const shoppingList = [];

console.log("It will be nice to have some fruit to eat.");

shoppingList.push("Apples");

function getShoppingListMsg(arr) {

  return `Current Shopping List: ${arr}`;

}

console.log(getShoppingListMsg(shoppingList));

shoppingList.push("Grapes");

console.log(getShoppingListMsg(shoppingList));

console.log("It looks like we need to get some cooking oil.");

shoppingList.unshift("Vegetable Oil");

console.log(getShoppingListMsg(shoppingList));

shoppingList.push("Popcorn", "Beef Jerky", "Potato Chips");

console.log(getShoppingListMsg(shoppingList));

console.log("This looks like too much junk food.");

shoppingList.pop();

console.log(getShoppingListMsg(shoppingList));

console.log("It might be nice to get a dessert.");

shoppingList.unshift("Chocolate Cake");

console.log(getShoppingListMsg(shoppingList));

console.log("On second thought, maybe we should be more health conscious.");

shoppingList.shift();

shoppingList[0] = "Canola Oil";

console.log( getShoppingListMsg(shoppingList));

**Build a Lunch Picker Program:**

In this question, you'll build a program that helps in managing lunch options. You'll work with an array of lunches, add and remove items from the array, and randomly select a lunch option.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the question.

**User Stories:**

1. You should create a lunches variable and assign it an empty array that will be used to store lunch items.
2. You should create a function addLunchToEnd that takes an array as the first argument and a string as the second argument. The function should:
   * Add the string to the end of the array.
   * Log the string "[Lunch Item] added to the end of the lunch menu." to the console, where [Lunch Item] is the string passed to the function.
   * Return the updated array.
3. You should create a function addLunchToStart that takes an array as the first argument and a string as the second argument. The function should:
   * Add the string to the start of the array.
   * Log the string "[Lunch Item] added to the start of the lunch menu." to the console, where [Lunch Item] is the string passed to the function.
   * Return the updated array.
4. You should create a function removeLastLunch that takes an array as its argument. The function should:
   * Remove the last element from the array.
   * If the removal is successful, log the string "[Lunch Item] removed from the end of the lunch menu." to the console, where [Lunch Item] is the element removed from the array.
   * If the array is empty, log the string "No lunches to remove." to the console.
   * Return the updated array.
5. You should create a function removeFirstLunch that takes an array as its argument. The function should:
   * Remove the first element from the array.
   * If the removal is successful, log the string "[Lunch Item] removed from the start of the lunch menu." to the console, where [Lunch Item] is the element removed from the array.
   * If the array is empty, log the string "No lunches to remove." to the console.
   * Return the updated array.
6. You should create a function getRandomLunch that takes an array as its argument. The function should:
   * Select a random element from the array.
   * If successful, log the string "Randomly selected lunch: [Lunch Item]" to the console, where [Lunch Item] is a random element in the array.
   * If the array is empty, log the string "No lunches available." to the console.
7. You should create a function showLunchMenu that takes an array as its argument and:
   * If there are elements in the array, logs the string "Menu items: [Lunch Item], [Lunch Item]..." to the console, where each [Lunch item] is one of the elements in the array, in order.
   * If the array is empty, logs the string "The menu is empty." to the console.

* 1. You should declare a variable lunches and assign it an empty array to store the lunch items.
* 2. You should define a function addLunchToEnd.
* 3. The function addLunchToEnd should have two parameters.
* 4. addLunchToEnd(lunches, "Tacos") should log the string "Tacos added to the end of the lunch menu." to the console.
* 5. addLunchToEnd(["Pizza", "Tacos"], "Burger") should return ["Pizza", "Tacos", "Burger"].
* 6. You should define a function addLunchToStart.
* 7. The function addLunchToStart should have two parameters.
* 8. addLunchToStart(lunches, "Sushi") should log the string "Sushi added to the start of the lunch menu." to the console.
* 9. addLunchToStart(["Burger", "Sushi"], "Pizza") should return ["Pizza", "Burger", "Sushi"].
* 10. You should define a function removeLastLunch.
* 11. The function removeLastLunch should have one parameter.
* 12. When the input array is empty, the function removeLastLunch should log the string "No lunches to remove." to the console.
* 13. removeLastLunch(["Stew", "Soup", "Toast"]) should log the string "Toast removed from the end of the lunch menu." to the console.
* 14. removeLastLunch(["Sushi", "Pizza", "Noodles"]) should return ["Sushi", "Pizza"].
* 15. You should define a function removeFirstLunch.
* 16. The function removeFirstLunch should have a single parameter.
* 17. When the input array is empty, the function removeFirstLunch should log the string "No lunches to remove." to the console.
* 18. removeFirstLunch(["Salad", "Eggs", "Cheese"]) should log the string "Salad removed from the start of the lunch menu." to the console.
* 19. removeFirstLunch(["Sushi", "Pizza", "Burger"]) should return ["Pizza", "Burger"].
* 20. addLunchToEnd, addLunchToStart, removeLastLunch, and removeFirstLunch should return the same array passed as an argument after updating it.
* 21. You should define a function getRandomLunch.
* 22. The function getRandomLunch should have a single parameter.
* 23. When the input array is empty, the function getRandomLunch should log the string "No lunches available." to the console.
* 24. When the input array is not empty, the function getRandomLunch should log a string in the format "Randomly selected lunch: [Lunch Item]" to the console.
* 25. The getRandomLunch function should not change the array passed to it as argument.
* 26. You should define a function showLunchMenu.
* 27. The function showLunchMenu should have a single parameter.
* 28. When the input array is empty, the function showLunchMenu should log the string "The menu is empty." to the console.
* 29. showLunchMenu(["Greens", "Corns", "Beans"]) should log "Menu items: Greens, Corns, Beans" to the console.
* 30. showLunchMenu(["Pizza", "Burger", "Fries", "Salad"]) should log "Menu items: Pizza, Burger, Fries, Salad" to the console.

**Code:**

let lunches=[];

function addLunchToEnd(lunches,str){

    lunches.push(str);

  console.log(`${str} added to the end of the lunch menu.`);

  return lunches;

}

addLunchToEnd(lunches,["Pizza", "Tacos", "Burger"]);

function addLunchToStart(lunches,str){

  lunches.unshift(str);

  console.log(`${str} added to the start of the lunch menu.`);

  return lunches;

}

addLunchToStart(lunches,["Burger", "Sushi", "Pizza"]);

function removeLastLunch(lunches){

  if(lunches.length!==0){

        let popedLunch= lunches.pop();

console.log(`${popedLunch} removed from the end of the lunch menu.`);

  }

  else{

  console.log("No lunches to remove.");

  }

    return lunches;

}

removeLastLunch(["Stew", "Soup", "Toast"]);

function removeFirstLunch(lunches){

  if(lunches.length!=0){

  let removedItem=lunches.shift();

  console.log(`${removedItem} removed from the start of the lunch menu.`)

  }

else{

    console.log("No lunches to remove.");

  }

 return lunches;

}

removeFirstLunch(["Salad", "Eggs", "Cheese"]);

//get Random Lunch function:

function getRandomLunch(lunches){

  if(lunches.length==0){

    console.log('No lunches available.');

  }

  else{

      let randNumber=Math.floor(Math.random() \*3);

  let randElement=lunches[randNumber];

    console.log(`Randomly selected lunch: ${randElement}`);

  }

}

getRandomLunch(["Salad", "Eggs", "Cheese"]);

// funation show Lunch Menu

function showLunchMenu(lunches){

    if(lunches.length!=0){

      console.log(`Menu items: ${lunches.join(', ')}`);

  }

  else{

      console.log('The menu is empty.');

  }

}

//showLunchMenu([]);

showLunchMenu(["Greens", "Corns", "Beans"]);

**Build a string inverter:**

In this lab, you will build a simple string inverter that reverses the characters of a given string.

For example, "hello" should become "olleh".

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a function named reverseString that takes a string as an argument.
2. The function should return the reversed string.

* 1. You should have a function named reverseString.
* 2. reverseString should take a string as an argument.
* 3. reverseString("hello") should return a string.
* 4. reverseString("hello") should return the string olleh.
* 5. reverseString("Howdy") should return the string ydwoH.
* 6. reverseString("Greetings from Earth") should return the string htraE morf sgniteerG.

**Code:**

function reverseString(str) {

  return str.split("").reverse("").join("");

}

**Build a recipe tracker:**

**Start by creating an empty array named recipes. This is the array you'll push the recipe objects into later.  
Create an object named recipe1. Inside the recipe1 object, create a name property with the value "Spaghetti Carbonara". Also inside the recipe1 object, create an ingredients property with an array as the value. The array should have spaghetti, Parmesan cheese, pancetta, and black pepper inside it. Create another ratings property with array value. The array should have 4, 5, 4, and 5 inside it.**Add the following properties to the recipe1 object:

| **Key** | **Value** |
| --- | --- |
| cookingTime | 22 |
| totalIngredients | null |
| difficultyLevel | "" |
| averageRating | null |

The properties with null and empty string values will be updated later after you calculate them.

Create another recipe2 object with the following properties and values:

| **Key** | **Value** |
| --- | --- |
| name | Chicken Curry |
| ingredients | ["chicken breast", "coconut milk", "curry powder", "onion", "garlic"] |
| ratings | [4, 5, 5, 5] |
| cookingTime | 42 |
| totalIngredients | Null |
| difficultyLevel | '' |
| averageRating | null |

Access the name properties of both recipe1 and recipe2, and assign them to the variables recipe1Name and recipe2Name, respectively.

Next, access the cookingTime properties of both recipes and assign them to the variables recipe1CookingTime and recipe2CookingTime, respectively.

Make sure all the variables you created are logged to the console.

A third recipe3 object has been filled in for you. It has the same properties as recipe1 and recipe2.

You should now push the three objects into the recipes array. To do this, you can use the push() method.

Use the push() method to push all the recipe objects into the recipes array. Make sure to push recipe1, recipe2, and recipe3 in that order.

Also delete the recipe1Name, recipe2Name, recipe1CookingTime, and recipe2CookingTime variables, and the console.log statements which log those variables.

In the next few steps you will work on calculating the average rating, the total ingredients, and the difficulty level for each recipe in the recipes array.

Start by creating a getAverageRating function that takes a single argument, which is an array with ratings. Inside the function, calculate the average rating from the array passed to the function.

Your getAverageRating function must return a number.

Create a getTotalIngredients function that takes a single argument, representing an array with ingredients, and returns the number of ingredients from the array passed to the function.  
Create a getDifficultyLevel function that takes a number indicating the cooking time as a parameter.

If the cooking time is less than or equal to 30, the function should return "easy". If it is less than or equal to 60, the function should return "medium". Otherwise, the function should return "hard".

Create three new variables: recipe1AverageRating, recipe1TotalIngredients, and recipe1DifficultyLevel. Assign them the values by calling the corresponding function for each variable and passing in the appropriate recipe1 property.

Finally, log each variable to the console to see the results.

You can now fill in each item of the recipes array with values for the averageRating, totalIngredients, and difficultyLevel properties.

For now, access the averageRating, totalIngredients, and difficultyLevel of recipe1 and set them to the appropriate results of function calls and arguments.

Repeat the process for the averageRating, totalIngredients, and difficultyLevel properties of recipe2.  
The averageRating, totalIngredients, and difficultyLevel properties of recipe3 have been filled in for you.

Now, log the recipes array to the console to see all its items filled with the updated values.

With that, your recipes tracker project is complete.

Code:

const recipes = [];

const recipe1 = {

  name: 'Spaghetti Carbonara',

  ingredients: ['spaghetti', 'Parmesan cheese', 'pancetta', 'black pepper'],

  cookingTime: 22,

  totalIngredients: null,

  difficultyLevel: '',

  ratings: [4, 5, 4, 5],

  averageRating: null,

};

const recipe2 = {

  name: 'Chicken Curry',

  ingredients: ['chicken breast', 'coconut milk', 'curry powder', 'onion', 'garlic'],

  cookingTime: 42,

  totalIngredients: null,

  difficultyLevel: '',

  ratings: [4, 5, 5, 5],

  averageRating: null,

};

const recipe3 = {

  name: 'Vegetable Stir Fry',

  ingredients: ['broccoli', 'carrot', 'bell pepper'],

  cookingTime: 15,

  totalIngredients: null,

  difficultyLevel: '',

  ratings: [4, 3, 4, 5],

  averageRating: null,

};

recipes.push(recipe1, recipe2, recipe3);

function getAverageRating(ratings) {

  const total = ratings[0] + ratings[1] + ratings[2] + ratings[3];

  return total / ratings.length;

}

function getTotalIngredients(ingredients) {

  return ingredients.length;

}

function getDifficultyLevel(cookingTime) {

  if (cookingTime <= 30) {

    return 'easy';

  } else if (cookingTime <= 60) {

    return 'medium';

  } else {

    return 'hard';

  }

}

const recipe1AverageRating = getAverageRating(recipe1.ratings);

console.log(recipe1AverageRating);

const recipe1TotalIngredients = getTotalIngredients(recipe1.ingredients);

console.log(recipe1TotalIngredients);

const recipe1DifficultyLevel = getDifficultyLevel(recipe1.cookingTime);

console.log(recipe1DifficultyLevel);

recipe1.averageRating = getAverageRating(recipe1.ratings);

recipe1.totalIngredients = getTotalIngredients(recipe1.ingredients)

recipe1.difficultyLevel = getDifficultyLevel(recipe1.cookingTime)

recipe2.averageRating = getAverageRating(recipe2.ratings);

recipe2.totalIngredients = getTotalIngredients(recipe2.ingredients);

recipe2.difficultyLevel = getDifficultyLevel(recipe2.cookingTime);

recipe3.averageRating = getAverageRating(recipe3.ratings);

recipe3.totalIngredients = getTotalIngredients(recipe3.ingredients)

recipe3.difficultyLevel = getDifficultyLevel(recipe3.cookingTime)

console.log(recipes);

Build a Quiz Game:

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create an array named questions.
2. The questions array should contain at least five objects, each having the keys category, question, choices, and answer.
3. The category key should have the value of a string representing a question category.
4. The question key should have the value of a string representing a question.
5. The choices key should have the value of an array containing three strings, which are alternative answers to the question.
6. The answer key should have the value of a string, representing the correct answer to the question. Also, the value of answer should be included in the choices array.
7. You should have a function named getRandomQuestion that takes an array of questions as a parameter and returns a random question object from the array.
8. You should have a function named getRandomComputerChoice that takes the array of the available choices as a parameter, and returns a random answer to the selected question.
9. You should have a function named getResults that takes the question object as the first parameter and the computer's choice as the second parameter. The function should return The computer's choice is correct! if the answer is correct. Otherwise, it returns The computer's choice is wrong. The correct answer is: <correct-answer>, where <correct-answer> is the value of the correct answer to the chosen question.

* 1. You should create an array named questions.
* 2. The questions array should contain at least five objects, each having the keys category, question, choices, and answer.
* 3. The category key should have the value of a string representing a question category.
* 4. The question key should have the value of a string representing a question.
* 5. The choices key should have the value of an array containing three strings.
* 6. The answer key should have the value of a string.
* 7. The value of answer should be included in the choices array.
* 8. You should have a function named getRandomQuestion that takes an array of questions as a parameter and returns a random question object from the array.
* 9. You should have a function named getRandomComputerChoice that takes the array of the available choices as a parameter, and returns a random answer to the selected question.
* 10. You should have a function named getResults.
* 11. Your getResults function should take the question object as the first parameter and the computer's choice as the second parameter.
* 12. If the computer choice matches the answer, getResults should return The computer's choice is correct!
* 13. If the computer choice doesn't match the answer, getResults should return The computer's choice is wrong. The correct answer is: <correct-answer>, where <correct-answer> is the value of the correct answer to the chosen question.

Code:

const questions = [

  {

    category: "PCo",

    question: "fav color ?",

    choices: ['Red', 'Black', 'Both'],

    answer: 'Red'

  },

  {

    category: "PCa",

    question: "Fav car ?",

    choices: ['Honda', 'BMW', 'both'],

    answer: 'Honda'

  },

  {

    category: "PB",

    question: "Fav brand ?",

    choices: ['Amazon', 'Google', 'Both'],

    answer: 'Both'

  },

  {

    category: "PF",

    question: "Fav fruit ?",

    choices: ['Apple', 'Grapes', 'Mango'],

    answer: 'Mango'

  },

  {

    category: "PH",

    question: "fav hobby ?",

    choices: ['Singing', 'Coding', 'Both'],

    answer: 'Both'

  }

];

// 8. Returns a random question object from the array

function getRandomQuestion(questionsArray) {

  let randomIndex = Math.floor(Math.random() \* questionsArray.length);

  return questionsArray[randomIndex];

}

// 9. Takes choices array and returns random choice

function getRandomComputerChoice(choicesArray) {

  let randomIndex = Math.floor(Math.random() \* choicesArray.length);

  return choicesArray[randomIndex];

}

// 10, 11, 12, 13: Result logic

function getResults(questionObj, computerChoice) {

  if (computerChoice === questionObj.answer) {

    return "The computer's choice is correct!";

  } else {

    return `The computer's choice is wrong. The correct answer is: ${questionObj.answer}`;

  }

}

// Test the functions

const selectedQuestion = getRandomQuestion(questions);

console.log("Question:", selectedQuestion.question);

const computerAnswer = getRandomComputerChoice(selectedQuestion.choices);

console.log("Computer's Choice:", computerAnswer);

const result = getResults(selectedQuestion, computerAnswer);

console.log(result);

**Build a Record Collection**

You are creating a function that aids in the maintenance of a musical album collection. The collection is organized as an object that contains multiple albums which are also objects. Each album is represented in the collection with a unique id as the property name. Within each album object, there are various properties describing information about the album. Not all albums have complete information.

The updateRecords function takes 4 arguments represented by the following function parameters:

* records - an object containing several individual albums
* id - a number representing a specific album in the records object
* prop - a string representing the name of the album’s property to update
* value - a string containing the information used to update the album’s property

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. Your function must always return the entire records object.
2. If value is an empty string, delete the given prop property from the album.
3. If prop isn't tracks and value isn't an empty string, assign the value to that album's prop.
4. If prop is tracks and value isn't an empty string, but the album doesn't have a tracks property, create an empty array and add value to it.
5. If prop is tracks and value isn't an empty string, add value to the end of the album's existing tracks array.

**Note:** A copy of the recordCollection object is used for the tests. Your function should not directly refer to the recordCollection object, only the function parameter.

* 1. You should have a updateRecords function.
* 2. After updateRecords(recordCollection, 5439, "artist", "ABBA"), artist should be the string ABBA
* 3. After updateRecords(recordCollection, 5439, "tracks", "Take a Chance on Me"), tracks should have the string Take a Chance on Me as the last and only element.
* 4. After updateRecords(recordCollection, 2548, "artist", ""), artist should not be set
* 5. After updateRecords(recordCollection, 1245, "tracks", "Addicted to Love"), tracks should have the string Addicted to Love as the last element.
* 6. After updateRecords(recordCollection, 2468, "tracks", "Free"), tracks should have the string 1999 as the first element.
* 7. After updateRecords(recordCollection, 2548, "tracks", ""), tracks should not be set
* 8. After updateRecords(recordCollection, 1245, "albumTitle", "Riptide"), albumTitle should be the string Riptide

Code:

function updateRecords(records, id, prop, value) {

  if (value === "") {

    // If value is an empty string, delete the prop from the album

    delete records[id][prop];

  } else if (prop !== "tracks") {

    // If prop is not 'tracks', update or set the property with the given value

    records[id][prop] = value;

  } else {

    // If prop is 'tracks' and value is not empty

    if (!records[id].hasOwnProperty("tracks")) {

      // If tracks doesn't exist, create it as an array

      records[id]["tracks"] = [];

    }

    // Push the new value to the tracks array

    records[id]["tracks"].push(value);

  }

  // Always return the entire records object

  return records;

}

Build a sentence analyzer:

In this question, you will build a sentence analyzer that will take a sentence and get the count for the number of words, vowels, consonants, and punctuation marks.

To begin, create a getVowelCount function with a parameter called sentence. Your function should return the total number of vowels in a sentence.

Create a vowelCount variable and assign it the result of calling the getVowelCount function with the argument of "Apples are tasty fruits"

After that, log the following to the console: "Vowel Count: [vowel count goes here]". Replace [vowel count goes here] with the actual variable name. You can choose to use template strings or string concatenation with the + operator here.

It's time to count the consonants. Create a getConsonantCount function with a sentence parameter.

Inside the function, use a loop to count the number of consonants in the sentence that will be passed into the function when it is called. A consonant is any letter that is not one of the following characters: "aeiou".

Your getConsonantCount function must return a number.

Now it is time to test your getConsonantCount function.

Create a consonantCount variable and assign it the result of calling the getConsonantCount function with the argument of "Coding is fun"

After that, log the following to the console: "Consonant Count: [Consonant count goes here]". Replace [Consonant count goes here] with the actual variable name. You can choose to use template strings or string concatenation with the + operator here.

ou should count the number of punctuations now.

Create a getPunctuationCount function with a sentence parameter.

Inside the function, create a loop to count the number of punctuation characters in the sentence that will be passed into the function when it is called. For our purposes, a punctuation character is any character that is not a space (" ") or a letter.

Your getPunctuationCount function must return a number.

Now it is time to test your getPunctuationCount function.

Create a punctuationCount variable and assign it the result of calling the getPunctuationCount function with the argument of "WHAT?!?!?!?!?"

After that, log the following to the console: "Punctuation Count: [Punctuation count goes here]". Replace [Punctuation count goes here] with the actual variable name. You can choose to use template strings or string concatenation with the + operator here.

Finally, count the number of words by creating a getWordCount function with a sentence parameter. The function should return the total number of words in the sentence passed in when it is called.  
Test your getWordCount by creating a wordCount variable set to the calling of the getWordCount function with the sentence "I love freeCodeCamp".

After that, log the following to the console: "Word Count: [Word count goes here]". Replace [Word count goes here] with the actual variable name. You can choose to use template strings or string concatenation with the + operator here.

With that, your sentence analyzer project is done!

Code:

function getVowelCount(sentence) {

  const vowels = "aeiou";

  let count = 0;

  for (const char of sentence.toLowerCase()) {

    if (vowels.includes(char)) {

      count++;

    }

  }

  return count;

}

const vowelCount = getVowelCount("Apples are tasty fruits");

console.log(`Vowel Count: ${vowelCount}`);

function getConsonantCount(sentence) {

  const consonants = "bcdfghjklmnpqrstvwxyz";

  let count = 0;

  for (const char of sentence.toLowerCase()) {

    if (consonants.includes(char)) {

      count++;

    }

  }

  return count;

}

const consonantCount = getConsonantCount("Coding is fun");

console.log(`Consonant Count: ${consonantCount}`);

function getPunctuationCount(sentence) {

  const punctuations = ".,!?;:-()[]{}\"'–";

  let count = 0;

  for (const char of sentence) {

    if (punctuations.includes(char)) {

      count++;

    }

  }

  return count;

}

const punctuationCount = getPunctuationCount("WHAT?!?!?!?!?");

console.log(`Punctuation Count: ${punctuationCount}`);

function getWordCount(sentence) {

  if (sentence.trim() === '') {

    return 0;

  }

  const words = sentence.trim().split(/\s+/);

  return words.length;

}

let wordCount =  getWordCount("I love freeCodeCamp");

console.log("Word Count: " + wordCount);

Build a Factorial Calculator:

A factorial is the product of an integer and all the integers below it. For example, the factorial of 5 is 5 \* 4 \* 3 \* 2 \* 1 = 120. In this lab, you will create a factorial calculator that takes a number from the user and calculates the factorial of that number.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should declare a variable num and assign it a number of your choice. The assigned number should be between 1 and 20 inclusive.
2. Create a function named factorialCalculator that takes a number as an argument and returns the factorial of that number.
3. Inside the function, declare a result variable and assign it the value of 1. Using a loop, loop through all numbers from 1 to the input number(inclusive) and for each number, multiply the result variable by the current number and assign the result to the result variable. You can choose to use either a for loop, while loop or do...while loop here.
4. You should call the factorialCalculator function with num as the argument and assign the result to the variable factorial.
5. You should store the final output in the format Factorial of [num] is [factorial] and assign it to the variable resultMsg.
6. You should output the value of resultMsg to the console.

* 1. You should have a num variable.
* 2. You should assign a value to the num variable
* 3. The value of num should be between 1 and 20.
* 4. The num value should be a number.
* 5. You should have a function named factorialCalculator.
* 6. The factorialCalculator function should take a number as an argument.
* 7. The factorial of 5 should return 120.
* 8. The factorial of 7 should return 5040.
* 9. You should call your factorialCalculator function with the num variable as the argument.
* 10. You should define a factorial variable and assign the result of the factorialCalculator function to it.
* 11. Your factorialCalculator should produce the correct result.
* 12. Your resultMsg should contain a string in a format Factorial of [num] is [factorial].
* 13. You should output the value of resultMsg to the console.

Code:

let num = 7

function factorialCalculator(n){

let result = 1;

for (let i = 1; i <= n; i++){

  result \*= i;

}

return result;

}

let factorial = factorialCalculator(num);

let resultMsg = `Factorial of ${num} is ${factorial}`

console.log(resultMsg);

Implement the mutations algorithm:

**User Stories:**

1. Create a function named mutation that takes an array as its argument.
2. mutation should return true if the string in the first element of the array contains all of the letters of the string in the second element of the array, and false otherwise. For example:
   * mutation(["hello", "Hello"]), should return true because all of the letters in the second string are present in the first, ignoring case.
   * mutation(["hello", "hey"]) should return false because the string hello does not contain a y.
   * mutation(["Alien", "line"]), should return true because all of the letters in line are present in Alien.

* 1. mutation(["hello", "hey"]) should return false.
* 2. mutation(["hello", "Hello"]) should return true.
* 3. mutation(["zyxwvutsrqponmlkjihgfedcba", "qrstu"]) should return true.
* 4. mutation(["Mary", "Army"]) should return true.
* 5. mutation(["Mary", "Aarmy"]) should return true.
* 6. mutation(["Alien", "line"]) should return true.
* 7. mutation(["floor", "for"]) should return true.
* 8. mutation(["hello", "neo"]) should return false.
* 9. mutation(["voodoo", "no"]) should return false.
* 10. mutation(["ate", "date"]) should return false.
* 11. mutation(["Tiger", "Zebra"]) should return false.
* 12. mutation(["Noel", "Ole"]) should return true.

Code:

function mutation(arr) {

  let first = arr[0].toLowerCase();

  let second = arr[1].toLowerCase();

  for (let i = 0; i < second.length; i++) {

    if (!first.includes(second[i])) {

      return false;

    }

  }

  return true;

}

console.log(mutation(["hello", "hey"])); // false

console.log(mutation(["hello", "Hello"])); // true

console.log(mutation(["Alien", "line"])); // true

console.log(mutation(["Mary", "Army"])); // true

Implement the Chunky Monkey Algorithm:

**User Stories:**

1. Write a function named chunkArrayInGroups that takes an array as first argument and a number as second argument. The function should split the array into smaller arrays of length equal to the second argument and returns them as a two-dimensional array.

* 1. chunkArrayInGroups(["a", "b", "c", "d"], 2) should return [["a", "b"], ["c", "d"]].
* 2. chunkArrayInGroups([0, 1, 2, 3, 4, 5], 3) should return [[0, 1, 2], [3, 4, 5]].
* 3. chunkArrayInGroups([0, 1, 2, 3, 4, 5], 2) should return [[0, 1], [2, 3], [4, 5]].
* 4. chunkArrayInGroups([0, 1, 2, 3, 4, 5], 4) should return [[0, 1, 2, 3], [4, 5]].
* 5. chunkArrayInGroups([0, 1, 2, 3, 4, 5, 6], 3) should return [[0, 1, 2], [3, 4, 5], [6]].
* 6. chunkArrayInGroups([0, 1, 2, 3, 4, 5, 6, 7, 8], 4) should return [[0, 1, 2, 3], [4, 5, 6, 7], [8]].
* 7. chunkArrayInGroups([0, 1, 2, 3, 4, 5, 6, 7, 8], 2) should return [[0, 1], [2, 3], [4, 5], [6, 7], [8]].

Code:

function chunkArrayInGroups(arr, size) {

  let result = [];

  for (let i = 0; i < arr.length; i += size) {

    result.push(arr.slice(i, i + size));

  }

  return result;

}

console.log(chunkArrayInGroups(["a", "b", "c", "d"], 2));

// ➝ [["a", "b"], ["c", "d"]]

console.log(chunkArrayInGroups([0, 1, 2, 3, 4, 5], 4));

// ➝ [[0, 1, 2, 3], [4, 5]]

console.log(chunkArrayInGroups([0, 1, 2, 3, 4, 5, 6], 3));

// ➝ [[0, 1, 2], [3, 4, 5], [6]]

Implement the Slice and Splice Algorithm:

**User Stories:**

1. Create a frankenSplice function that accepts two arrays and an index.
2. Copy each element of the first array into the second array, in order, beginning at the given index, and return the resulting array.
3. The input arrays should remain the same after the function runs.

* 1. frankenSplice([1, 2, 3], [4, 5], 1) should return [4, 1, 2, 3, 5].
* 2. frankenSplice([1, 2], ["a", "b"], 1) should return ["a", 1, 2, "b"].
* 3. frankenSplice(["claw", "tentacle"], ["head", "shoulders", "knees", "toes"], 2) should return ["head", "shoulders", "claw", "tentacle", "knees", "toes"].
* 4. All elements from the first array should be added to the second array in their original order. frankenSplice([1, 2, 3, 4], [], 0) should return [1, 2, 3, 4].
* 5. The first array should remain the same after the function runs.
* 6. The second array should remain the same after the function runs.

Code:

function frankenSplice(arr1, arr2, index) {

  // Create a shallow copy of arr2 to avoid modifying the original

  let result = arr2.slice();

  // Use splice to insert elements of arr1 into the copied array at the specified index

  result.splice(index, 0, ...arr1);

  return result;

}

console.log(frankenSplice([1, 2, 3], [4, 5], 1));

// → [4, 1, 2, 3, 5]

console.log(frankenSplice(["claw", "tentacle"], ["head", "shoulders", "knees", "toes"], 2));

// → ["head", "shoulders", "claw", "tentacle", "knees", "toes"]

console.log(frankenSplice([1, 2], ["a", "b"], 1));

// → ["a", 1, 2, "b"]

Build a Pyramid Generator:

**User Stories:**

1. You should have a function named pyramid that takes three arguments.
2. The first argument should be a string representing the pattern character to repeat in your pyramid.
3. The second argument should be an integer representing the number of rows in the pyramid.
4. The third argument should be a Boolean value.
5. The pyramid function should return a string in which the pattern character is repeated and arranged to form a pyramid having the vertex facing upwards when the third argument is false.
6. When the third argument is true the pyramid should have the vertex facing downwards.
7. The vertex row should have a single pattern character, and each other row should have two pattern characters more than the previous one.
8. Each row should start with a number of spaces sufficient to put the center character of each row in the same column.
9. The pyramid should start and end with a newline character.

For example, calling pyramid("o", 4, false) should give this output:

Example Code

o

ooo

ooooo

ooooooo

* 1. You should have a function named pyramid.
* 2. Your pyramid function should have three parameters.
* 3. pyramid("o", 4, false) should return "\n o\n ooo\n ooooo\nooooooo\n".
* 4. pyramid("p", 5, true) should return "\nppppppppp\n ppppppp\n ppppp\n ppp\n p\n".

Code:

function pyramid(char, rows, inverted) {

  let result = "\n";

  for (let i = 0; i < rows; i++) {

    let level = inverted ? i : rows - 1 - i;

    let spaces = " ".repeat(level);

    let patternCount = 1 + 2 \* (rows - 1 - level);

    let pattern = char.repeat(patternCount);

    result += spaces + pattern + "\n";

  }

  return result;

}

console.log(pyramid("o", 4, false));

console.log(pyramid("p", 5, true));

Build a Gradebook App:

**User Stories:**

1. You should have a function named getAverage that takes an array of test scores as a parameter and returns the average score.
2. You should have a function named getGrade that takes a student score as a parameter and returns a string representing a letter grade based on the score. Here are the scores and their corresponding letter grades:

|  |  |
| --- | --- |
| Score Range | Grade |
| 100 | "A+" |
| 90 - 99 | "A" |
| 80 - 89 | "B" |
| 70 - 79 | "C" |
| 60 - 69 | "D" |
| 0 - 59 | "F" |

1. You should have a function named hasPassingGrade that takes a score as a parameter and returns either true or false depending on if the score corresponds to a passing grade.
2. The hasPassingGrade function should use the getGrade function to get the letter grade, and use it to determine if the grade is passing. A passing grade is anything different from "F".
3. You should have a function named studentMsg that takes an array of scores and a student score as the parameters. The function should return a string with the format:
   * "Class average: average-goes-here. Your grade: grade-goes-here. You passed the course.", if the student passed the course.
   * "Class average: average-goes-here. Your grade: grade-goes-here. You failed the course.", if the student failed the course.

Replace average-goes-here with the average of total scores and grade-goes-here with the student's grade. Use getAverage to get the average score and getGrade to get the student's grade.

* 1. You should have a function named getAverage.
* 2. Your getAverage function should return a number.
* 3. getAverage([92, 88, 12, 77, 57, 100, 67, 38, 97, 89]) should return 71.7.
* 4. getAverage([45, 87, 98, 100, 86, 94, 67, 88, 94, 95]) should return 85.4.
* 5. getAverage([38, 99, 87, 100, 100, 100, 100, 100, 100, 100]) should return 92.4.
* 6. getAverage([10, 20, 30, 40, 55, 65, 75, 83]) should return 47.25.
* 7. getAverage([10, 20, 30, 40, 50, 60, 70, 97]) should return 47.125.
* 8. Your getAverage function should return the average score of the test scores.
* 9. You should have a function named getGrade.
* 10. Your getGrade function should return "A+" if the score is 100.
* 11. Your getGrade function should return "A" if the score is between 90 and 99.
* 12. Your getGrade function should return "B" if the score is between 80 and 89.
* 13. Your getGrade function should return "C" if the score is between 70 and 79.
* 14. Your getGrade function should return "D" if the score is between 60 and 69.
* 15. Your getGrade function should return "F" if the score is between 0 and 59.
* 16. You should have a function named hasPassingGrade.
* 17. Your hasPassingGrade function should return a boolean value.
* 18. Your hasPassingGrade function should return true if the grade is an "A".
* 19. Your hasPassingGrade function should return false if the grade is an "F".
* 20. Your hasPassingGrade function should return true for all passing grades.
* 21. You should have a function named studentMsg.
* 22. studentMsg([92, 88, 12, 77, 57, 100, 67, 38, 97, 89], 37) should return the following message: "Class average: 71.7. Your grade: F. You failed the course.".
* 23. studentMsg([56, 23, 89, 42, 75, 11, 68, 34, 91, 19], 100) should return the following message: "Class average: 50.8. Your grade: A+. You passed the course.".
* 24. studentMsg([12, 22, 32, 42, 52, 62, 72, 92], 85) should return the following message: "Class average: 48.25. Your grade: B. You passed the course.".
* 25. studentMsg([15, 25, 35, 45, 55, 60, 70, 60], 75) should return the following message: "Class average: 45.625. Your grade: C. You passed the course.".
* 26. Your studentMsg function should return the correct message based on the student's score and the class average.

Code:

function getAverage(scores) {

  const total = scores.reduce((sum, score) => sum + score, 0);

  return total / scores.length;

}

function getGrade(score) {

  if (score === 100) return "A+";

  if (score >= 90) return "A";

  if (score >= 80) return "B";

  if (score >= 70) return "C";

  if (score >= 60) return "D";

  return "F";

}

function hasPassingGrade(score) {

  return getGrade(score) !== "F";

}

function studentMsg(scores, studentScore) {

  const average = getAverage(scores);

  const grade = getGrade(studentScore);

  const passed = hasPassingGrade(studentScore);

  const status = passed ? "You passed the course." : "You failed the course.";

  return `Class average: ${average}. Your grade: ${grade}. ${status}`;

}

console.log(studentMsg([92, 88, 12, 77, 57, 100, 67, 38, 97, 89], 37));

// ➝ "Class average: 71.7. Your grade: F. You failed the course."

Build an Inventory Management Program:

In this question, you will build an inventory management program that will allow you to add, update, find and remove products from the inventory. You will use an array of objects to represent your inventory, where each object will have name and quantity as the keys.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should declare an empty array named inventory that will store product objects having a key name with the value of a lowercase string, and a key quantity with the value of an integer.
2. You should create a function named findProductIndex that takes the product name as its argument and returns the index of the corresponding product object inside the inventory array. The function should always use the lowercase form of the product name to perform the search. If the product is not found, the function should return -1.
3. You should create a function named addProduct that takes a product object as its argument.
4. If the product is already present in the inventory, the addProduct function should update its quantity value and log to the console the product name followed by a space and quantity updated.
5. If the product is not present in the inventory, the addProduct function should push the product to the inventory array and log the product name to the console, followed by a space and added to inventory.
6. You should create a function named removeProduct that takes the product name and quantity as its arguments.
7. The removeProduct function should subtract the passed quantity from the corresponding product object inside the inventory and log the string Remaining <product-name> pieces: <product-quantity> to the console, where <product-name> should be replaced by the product name, and <product-quantity> should be replaced by the product quantity.
8. If the quantity after the subtraction is zero, removeProduct should remove the product object from the inventory. If the quantity in the inventory is not enough to perform the subtraction, the removeProduct function should log the string Not enough <product-name> available, remaining pieces: <product-quantity> to the console.
9. If the product to remove is not present in the inventory, the removeProduct function should log <product-name> not found to the console.

**Note:** To prevent conflicts, keep only the logging mentioned in the user stories when running tests.

* 1. You should declare an empty array named inventory.
* 2. You should have a function named findProductIndex.
* 3. findProductIndex("flour") should return the index of the object having name equal to "flour" inside the inventory array.
* 4. findProductIndex("FloUr") should return the index of the object having name equal to "flour" inside the inventory array.
* 5. findProductIndex("Flour") should return -1 when no object having name equal to "flour" is found inside the inventory array.
* 6. You should have a function named addProduct.
* 7. addProduct({name: "FLOUR", quantity: 5}) should add 5 to quantity value of the object having name equal to "flour" in the inventory array.
* 8. addProduct({name: "FLOUR", quantity: 5}) should log flour quantity updated when an object having name equal to "flour" is found in the inventory array.
* 9. addProduct({name: "FLOUR", quantity: 5}) should push {name: "flour", quantity: 5} to the inventory array when no object having name equal to "flour" is found in the inventory.
* 10. addProduct({name: "FLOUR", quantity: 5}) should log flour added to inventory when no object having name equal to "flour" is found in the inventory.
* 11. You should create a function named removeProduct.
* 12. removeProduct("FLOUR", 5) should log flour not found when no object having name equal to "flour" is found in the inventory array.
* 13. removeProduct("FLOUR", 5) should subtract 5 from the quantity value of the object having name equal to "flour" inside the inventory array.
* 14. removeProduct("FLOUR", 5) should log Remaining flour pieces: 15 to the console, when inventory is equal to [{name: "flour", quantity: 20}, {name: "rice", quantity: 5}].
* 15. If the quantity after the subtraction is zero, removeProduct should remove the product object from the inventory.
* 16. removeProduct("FLOUR", 10) should log Not enough flour available, remaining pieces: 5 when inventory is equal to [{name: "flour", quantity: 5}, {name: "rice", quantity: 5}].

Code:

// 1. Declare the inventory array

let inventory = [];

// 2. Function to find product index by lowercase name

function findProductIndex(productName) {

  const name = productName.toLowerCase();

  return inventory.findIndex(item => item.name === name);

}

// 3. Function to add or update a product

function addProduct(product) {

  const name = product.name.toLowerCase();

  const quantity = product.quantity;

  const index = findProductIndex(name);

  if (index !== -1) {

    inventory[index].quantity += quantity;

    console.log(`${name} quantity updated`);

  } else {

    inventory.push({ name, quantity });

    console.log(`${name} added to inventory`);

  }

}

// 4. Function to remove quantity or remove the product entirely

function removeProduct(productName, quantity) {

  const name = productName.toLowerCase();

  const index = findProductIndex(name);

  if (index === -1) {

    console.log(`${name} not found`);

    return;

  }

  const product = inventory[index];

  if (product.quantity < quantity) {

    console.log(`Not enough ${name} available, remaining pieces: ${product.quantity}`);

  } else if (product.quantity === quantity) {

    inventory.splice(index, 1);

  } else {

    product.quantity -= quantity;

    console.log(`Remaining ${name} pieces: ${product.quantity}`);

  }

}

addProduct({ name: "FLOUR", quantity: 10 });

// → "flour added to inventory"

addProduct({ name: "FlOur", quantity: 5 });

// → "flour quantity updated"

removeProduct("FLOur", 5);

// → "Remaining flour pieces: 10"

removeProduct("FlOUR", 20);

// → "Not enough flour available, remaining pieces: 10"

removeProduct("Rice", 5);

// → "rice not found"

Build a Password Generator App:

In this question, you'll practice using functions by building a random password generator.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a function called generatePassword that takes a parameter, indicating the length of generated password. You can name the parameter whatever you like.
2. Your function should return a string which represents a randomly generated password. You should use the following string and different Math methods to help you return a new string with random characters in it: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&\*().
3. You should define a variable called password and assign it the result of calling the generatePassword function with a numeric argument that represents the desired password length.
4. You should have a console.log that logs a single string made by concatenating the message Generated password: and the password variable separated by a space.

* 1. You should have a generatePassword function with a parameter. You can name the parameter whatever you like.
* 2. Your generatePassword function should return a string.
* 3. Your generatePassword function should return a new string that is the correct length.
* 4. Your function should return a randomly generated password with valid characters.
* 5. Your function should return a new random string each time it is called.
* 6. You should have a password variable.
* 7. Your password variable should be a string.
* 8. You should call the generatePassword function with a numeric argument and store the returned password in the password variable.
* 9. You should log a single string combining Generated password: and the password separated by a single space using + or a template literal.

Code:

// 1. Function to generate password

function generatePassword(length) {

  const characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&\*()";

  let result = "";

  for (let i = 0; i < length; i++) {

    // Pick a random character from the characters string

    const randomIndex = Math.floor(Math.random() \* characters.length);

    result += characters[randomIndex];

  }

  return result;

}

// 2. Call function and store result in password variable

const password = generatePassword(12); // You can choose any number here

// 3. Log result to the console

console.log("Generated password: " + password);

Design a Sum All Numbers Algorithm:

In this question, you will need to design a sum all numbers algorithm.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a function named sumAll that accepts an array of two numbers.
2. sumAll([n, m]) should return the sum of n and m plus the sum of all the numbers between them. The lowest number will not always come first. For example, sumAll([4,1]) should return 10 because sum of all the numbers between 1 and 4 (both inclusive) is 10.

* 1. You should have a function named sumAll.
* 2. sumAll([1, 4]) should return a number.
* 3. sumAll([1, 4]) should return 10.
* 4. sumAll([4, 1]) should return 10.
* 5. sumAll([5, 10]) should return 45.
* 6. sumAll([10, 5]) should return 45.

Code:

function sumAll(arr) {

  // Get the minimum and maximum from the array

  const min = Math.min(arr[0], arr[1]);

  const max = Math.max(arr[0], arr[1]);

  let sum = 0;

  for (let i = min; i <= max; i++) {

    sum += i;

  }

  return sum;

}

console.log(sumAll([1, 4]));   // ➝ 10

console.log(sumAll([4, 1]));   // ➝ 10

console.log(sumAll([5, 10]));  // ➝ 45

console.log(sumAll([10, 5]));  // ➝ 45

Build a Library Manager:

In this question, you will build a library manager app that will give you an opportunity to practice working with different higher order functions.

To get started, create a variable called library and assign it an empty array.

Inside the library array, create an object with the following properties and values:

|  |  |
| --- | --- |
| Property | Value |
| title | "Your Next Five Moves: Master the Art of Business Strategy" |
| author | "Patrick Bet-David and Greg Dinkin" |
| about | "A book on how to plan ahead" |
| pages | 320 |

Create another object inside the library array with the following properties and values:

| **Property** | **Value** |
| --- | --- |
| title | "Atomic Habits" |
| author | "James Clear" |
| about | "A practical book about discarding bad habits and building good ones" |
| pages | 320 |

Please fill more book objects.

Now would be a good time to start working on displaying the book information in the console. Over the next few steps, you will learn how to work with the map() method to achieve this goal.

Begin by logging the message "Books in the Library:\n" to the console. The newline character is added here because there should be a space between this message and the list of books.

Next, create a function called getBookInformation with one parameter. This parameter will represent the array of books when the function is called. You can name this parameter whatever you like.  
Inside the getBookInformation function, use the map() method on the catalog parameter to return a new array of just book titles.  
Now it is time to see what the result will look like in the console.

Below your getBookInformation function, add a console.log() and pass in a getBookInformation() function call with library for the argument.

When done correctly, you should see an array of book titles in the console.

Update your use of the map() method to instead return an array of strings in this format: TITLE by AUTHOR.

You will need to use either template literals or string concatenation with the + operator to achieve this result.

While the array of results is working, the final desired result should be a string listing all of the books with a title and author.  
The separator in this case is the \n which represents the newline character.

Chain the join() method with a \n for the separator to the map(). Now you should see a string in the console instead of the array of results.

For the next part of the question, you will focus on displaying a list of book summaries to the console.

Begin by logging the message "\nList of book summaries:\n" to the console. The newline character is added here because there should be a space before and after the message here.

To get a list of book summaries, you can use the about property of each book.

Create a getBookSummaries function with a single parameter, accepting an array with book objects. You can name the parameter whatever you like.

The getBookSummaries function should return an array of strings representing the summary for each book.

Just like in a prior step, the desired result would look nicer as a string instead of an array.

Chain the join() method to your existing map() method and pass in "\n" for the separator to the join().

Then, below your getBookSummaries function, add a console.log(). The console statement should call the getBookSummaries function with library for the argument.

Now, you should see a string for the result in the console.

For this next part of the question, you are going to learn how to display a list of books by author.

Begin by logging the message "\nList of books by Arvid Kahl:\n" to the console.

Create a getBooksByAuthor function with two parameters - an array with book objects and a string with the author.

The function must return an array that contains the books by a particular author.

Now, it is time to test out your function.

Add a console.log() below your getBooksByAuthor() function.

Inside the console.log(), call the getBooksByAuthor() function with library and "Arvid Kahl" for arguments.

Now, you should see all of the books for that particular author in the console.

It would be nice to test our your getBooksByAuthor function with another author.

Begin by logging the message "\nList of books by James Clear:\n" to the console.

Below that console.log(), add another console.log(). Inside that console.log(), call the getBooksByAuthor() function with library and "James Clear" for arguments.

Now, you should see all of the books for that particular author in the console.

For the last part of the question, you will learn how to get the total number of pages for all of the books in the library.

Start by logging to the message "\nTotal number of pages for all library books:\n" to the console.

For the last step of the question, you will review how to work with the reduce() method. This method is used to process an array and condense it into a single value.

Create a getTotalPages function with a single parameter, accepting an array with book objects. The function should return the total number of pages in the books from the array passed to the function.

Lastly, add a console.log(getTotalPages(library)) below your getTotalPages function to see the result.

With that, your library manager question is complete!

Code:

const library = [

  {

    title: 'Your Next Five Moves: Master the Art of Business Strategy',

    author: 'Patrick Bet-David and Greg Dinkin',

    about: 'A book on how to plan ahead',

    pages: 320,

  },

  {

    title: 'Atomic Habits',

    author: 'James Clear',

    about: 'A practical book about discarding bad habits and building good ones',

    pages: 320,

  },

  {

    title: 'Choose Your Enemies Wisely: Business Planning for the Audacious Few',

    author: 'Patrick Bet-David',

    about:

      "A book that emphasizes the importance of identifying and understanding one's adversaries to succeed in the business world",

    pages: 304,

  },

  {

    title: 'The Embedded Entrepreneur',

    author: 'Arvid Kahl',

    about: 'A book focusing on how to build an audience-driven business',

    pages: 308,

  },

  {

    title: 'How to Be a Coffee Bean: 111 Life-Changing Ways to Create Positive Change',

    author: 'Jon Gordon',

    about: 'A book about effective ways to lead a coffee bean lifestyle',

    pages: 256,

  },

  {

    title: 'The Creative Mindset: Mastering the Six Skills That Empower Innovation',

    author: 'Jeff DeGraff and Staney DeGraff',

    about: 'A book on how to develop creativity and  innovation skills',

    pages: 168,

  },

  {

    title: 'Rich Dad Poor Dad',

    author: 'Robert Kiyosaki and Sharon Lechter',

    about: 'A book about financial literacy, financial independence, and building wealth. ',

    pages: 336,

  },

  {

    title: 'Zero to Sold',

    author: 'Arvid Kahl',

    about: 'A book on how to bootstrap a business',

    pages: 500,

  },

];

console.log("Books in the Library:\n");

function getBookInformation(catalog) {

  return catalog.map(book => `${book.title} by ${book.author}`).join("\n");

}

console.log(getBookInformation(library));

console.log("\nList of book summaries:\n");

function getBookSummaries(catalog) {

  return catalog.map((book) => book.about).join("\n");

}

console.log(getBookSummaries(library));

console.log("\nList of books by Arvid Kahl:\n");

function getBooksByAuthor(catalog, author) {

  return catalog.filter((book) => book.author === author);

}

console.log(getBooksByAuthor(library, "Arvid Kahl"));

console.log("\nList of books by James Clear:\n");

console.log(getBooksByAuthor(library, "James Clear"));

console.log("\nTotal number of pages for all library books:\n");

function getTotalPages(books) {

return books.reduce((total, book) => total + book.pages, 0);

}

console.log(getTotalPages(library));

Build a Book Organizer:

**User Stories:**

1. You should have an array of objects named books where each object in the array should have a string title, another string authorName, and a number releaseYear.
2. Your books array should have a minimum of three objects.
3. You should have a callback function named sortByYear that accepts two books as parameter for sorting the array.
4. The sortByYear function should return -1 if the releaseYear of the first book is smaller than that of the second book.
5. The sortByYear function should return 1 if the releaseYear of the first book is bigger than that of the second book.
6. The sortByYear function should return 0 if both releaseYear values are equal.
7. You should filter out books written after a certain year such as 1950 from the books array and save the filtered array in a new array named filteredBooks.
8. You should sort the books in the filteredBooks array according to their releaseYear in ascending order. You learned in a prior lecture video that the sort() method will sort the array in place. This means the filteredBooks array will be mutated.

* 1. You should have a function sortByYear in your code.
* 2. Your sortByYear function should take two parameters.
* 3. Your sortByYear function should return -1 if the releaseYear of book1 object is smaller than that of the book2 object, 1 if the releaseYear of book1 object is larger than that of the book2 object, and 0 in all other scenarios.
* 4. You should have an array books in your code.
* 5. Your books array should have at least three book objects.
* 6. Your books array should contain objects each with a string title, another string authorName, and a number releaseYear.
* 7. You should have an array filteredBooks in your code.
* 8. The filteredBooks array should have some of the books filtered out from the books array and not be empty.
* 9. You should call the sort higher order function by passing the sortByYear callback function on the filteredBooks array.

Code:

// Step 1: Create the books array

const books = [

  { title: "To Kill a Mockingbird", authorName: "Harper Lee", releaseYear: 1960 },

  { title: "1984", authorName: "George Orwell", releaseYear: 1949 },

  { title: "The Great Gatsby", authorName: "F. Scott Fitzgerald", releaseYear: 1925 },

  { title: "Lord of the Flies", authorName: "William Golding", releaseYear: 1954 }

];

// Step 2: Define the sortByYear function

function sortByYear(book1, book2) {

  if (book1.releaseYear < book2.releaseYear) {

    return -1;

  } else if (book1.releaseYear > book2.releaseYear) {

    return 1;

  } else {

    return 0;

  }

}

// Step 3: Filter out books written after 1950

const filteredBooks = books.filter(book => book.releaseYear <= 1950);

// Step 4: Sort filteredBooks in ascending order by releaseYear

filteredBooks.sort(sortByYear);

// Step 5: Output result for verification

console.log("Filtered and Sorted Books:", filteredBooks);

Build a Story Telling App:

Begin by creating an h1 element and give it a text Want to hear a short story?.  
Create a main element with a class of story-container. Inside the .story-container element, create a div with a class of btn-container.  
Inside the .btn-container , create three buttons for each type of story. Give each button a class of btn and an id of scary-btn, funny-btn, and adventure-btn. The text for each button should be Scary Story, Funny Story, and Adventure Story respectively.  
Below the .btn-container element, you should have a p element with the id of result. This is where the story will be displayed.  
Next, you will start working on the JavaScript. For that, begin by linking the script.js file to your HTML.

Create a script element with its src attribute pointing to the script.js file.  
In your JavaScript file, select the .story-container element and store it in a variable called storyContainer.  
Select the #scary-btn, #funny-btn, and #adventure-btn buttons and store them in variables called scaryStoryBtn, funnyStoryBtn, and adventureStoryBtn.  
Select the #result paragraph and store it in a variable called resultParagraph. You will use this variable to display the selected story.  
To hold the different types of stories, create an object called storyObj. The object should have three properties: scary, funny, and adventure. Each property value should be an object with a story and a borderColor property.

Assign the following values to the individual objects:

* scary:
  + story: In the dark woods, a group of friends stumbled upon an old, abandoned cabin. They enter the cabin and awaken something malevolent that had been dormant for centuries.
  + borderColor: #ee4b2b
* funny:
  + story: During a camping trip, Mark decided to show off his culinary skills by cooking dinner over an open fire. However, his attempt caused him to burn the dinner as well as his eyebrows off.
  + borderColor: #f1be32
* adventure:
  + story: Lost in the heart of the Amazon rain forest, Sarah and Jake stumbled upon an ancient temple. They braved deadly traps and encountered strange wildlife, all while deciphering cryptic clues left behind by a mysterious civilization.
  + borderColor: #acd157

Create a function called displayStory. Inside the function, log "You clicked the button" to the console.  
Add an event listener for the click event to scaryStoryBtn and pass it the displayStory function. Then, click your button and see the output in the console.  
Now, you know that the button works. Remove the console.log statement from the displayStory function.  
To differentiate between the different types of stories, you will need to pass the genre of the selected story to the displayStory function.

Add a parameter called genre to the displayStory function.  
Inside the displayStory function, check if the storyObj object has a property that matches the genre parameter and set the #result paragraph's text content to the story of the selected genre.

Also, set the storyContainer's border color to the borderColor property of the selected genre.

Test this by calling the displayStory function with the scary genre. The border color should be #ee4b2b and the story should be "In the dark woods, a group of friends stumbled upon an old, abandoned cabin. They enter the cabin and awaken something malevolent that had been dormant for centuries.".  
Now that the displayStory function is working correctly, it is time to add the functionality to the buttons.

Before that, remove the displayStory("scary") function call.  
Modify the scaryStoryBtn event listener to call the displayStory function with the argument "scary". Also, add a click event listener to the funnyStoryBtn, and adventureStoryBtn buttons that call the displayStory function with the arguments "funny" and "adventure" respectively.

With this, you have successfully completed the workshop.

Code:

index.html🡪  
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Storytelling App</title>

    <link rel="stylesheet" href="./styles.css" />

</head>

<body>

    <h1>Want to hear a short story?</h1>

    <main class="story-container">

        <div class="btn-container">

            <button class="btn" id="scary-btn">Scary Story</button>

            <button class="btn" id="funny-btn">Funny Story</button>

            <button class="btn" id="adventure-btn">Adventure Story</button>

        </div>

        <p id="result"></p>

    </main>

    <script src="./script.js"></script>

</body>

</html>

script.js🡪  
const storyContainer = document.querySelector(".story-container");

const scaryStoryBtn = document.getElementById("scary-btn");

const funnyStoryBtn = document.getElementById("funny-btn");

const adventureStoryBtn = document.getElementById("adventure-btn");

const resultParagraph = document.getElementById("result");

const storyObj = {

  scary: {

    story: `In the dark woods, a group of friends stumbled upon an old, abandoned cabin. They enter the cabin and awaken something malevolent that had been dormant for centuries.`,

    borderColor: "#ee4b2b",

},

  funny: {

    story: `During a camping trip, Mark decided to show off his culinary skills by cooking dinner over an open fire. However, his attempt caused him to burn the dinner as well as his eyebrows off.`,

    borderColor: "#f1be32",

  },

  adventure: {

    story: `Lost in the heart of the Amazon rain forest, Sarah and Jake stumbled upon an ancient temple. They braved deadly traps and encountered strange wildlife, all while deciphering cryptic clues left behind by a mysterious civilization.`,

    borderColor: "#acd157"

  },

};

function displayStory(genre) {

  if (storyObj.hasOwnProperty(genre)) {

    resultParagraph.textContent = storyObj[genre].story;

    storyContainer.style.borderColor = storyObj[genre].borderColor;

  }

}

scaryStoryBtn.addEventListener("click", displayStory);

scaryStoryBtn.addEventListener("click", () => displayStory("scary"));

funnyStoryBtn.addEventListener("click", () => displayStory("funny"));

adventureStoryBtn.addEventListener("click", () => displayStory("adventure"));

styles.css🡪

\*,

\*::before,

\*::after {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

:root {

    --dark-grey: #1b1b32;

    --black: #000;

    --white: #fff;

    --golden-yellow: #fecc4c;

    --yellow: #ffcc4c;

    --gold: #feac32;

    --orange: #ffac33;

    --dark-orange: #f89808;

}

body {

    background-color: var(--dark-grey);

    color: var(--white);

}

h1,

#result {

    text-align: center;

}

h1 {

    margin: 10px 0 15px;

}

.story-container {

    margin: auto;

    padding: 10px;

    width: 80%;

    border-style: double;

    border-width: 14px;

    border-color: var(--white);

}

.btn-container {

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: center;

}

@media (min-width: 760px) {

    .btn-container {

        flex-direction: row;

    }

}

#result {

    margin-top: 15px;

    font-size: 1.2rem;

    line-height: 30px;

}

.btn {

    cursor: pointer;

    width: 200px;

    margin: 10px 0 10px 0.5rem;

    color: var(--black);

    background-color: var(--gold);

    background-image: linear-gradient(var(--golden-yellow), var(--orange));

    border-color: var(--gold);

    border-width: 3px;

}

.btn:hover {

    background-image: linear-gradient(var(--yellow), var(--dark-orange));

}

Build a Favorite Icon Toggler:

A white rectangular box with red heart icons

AI-generated content may be incorrect.

In this lab you will use JavaScript click events to toggle the appearance of a favorite icon. When the heart icon is clicked, the appearance of the heart changes from empty to filled, and vice versa.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have an unordered list with three items.
2. The unordered list should have the class item-list.
3. The three list items should contain the item name followed by a button element with the class favorite-icon.
4. The button element should contain the code &#9825; initially to represent an empty heart.
5. When a button element containing a heart is clicked, you should add a class named filled to the clicked button if it's not already present, and remove it, if it is.
6. You should have a class selector that targets the filled class and sets some CSS properties.
7. When a button element containing a heart is clicked, the heart symbol should toggle between &#9825; (empty heart) and &#10084; (filled heart), depending on its current state.

**Note:** Be sure to link your JavaScript file in your HTML. (Ex. <script src="script.js"></script>)

* 1. You should have an unordered list.
* 2. Your unordered list should have 3 items.
* 3. Your unordered list should have the class item-list.
* 4. Your individual list items should contain the item name.
* 5. Your individual list item should contain a button element with the class favorite-icon.
* 6. Initially, the button elements should contain the code &#9825; to represent an empty heart.
* 7. You should have a .filled selector that sets some CSS properties.
* 8. When the button element is clicked, and it contains the class filled, you should remove the class filled from the button element and change the innerHTML of the button element to &#9825;.
* 9. When the button element is clicked, and it doesn't contain the class filled, you should add the class filled to the button element and change the innerHTML of the button element to &#10084;.

Code:

index.html🡪

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <title>Favorite Icon Toggler</title>

    <link rel="stylesheet" href="styles.css" />

  </head>

  <body>

    <ul class="item-list">

      <li>Item 1 <button class="favorite-icon">&#9825;</button></li>

      <li>Item 2 <button class="favorite-icon">&#9825;</button></li>

      <li>Item 3 <button class="favorite-icon">&#9825;</button></li>

    </ul>

    <script src="script.js"></script>

  </body>

</html>

styles.css🡪  
/\* Normal heart (empty): nothing specific needed \*/

.filled {

  color: red;

  font-weight: bold;

}

script.js🡪  
const buttons = document.querySelectorAll(".favorite-icon");

buttons.forEach(button => {

  button.addEventListener("click", () => {

    if (button.classList.contains("filled")) {

      button.classList.remove("filled");

      button.innerHTML = "&#9825;"; // empty heart

    } else {

      button.classList.add("filled");

      button.innerHTML = "&#10084;"; // filled heart

    }

  });

});

Build a Music Instrument Filter:

Inside index.html under the body tag create a h1 tag with text Student Instruments.  
After that create a main tag . Inside the main tag create a div tag with class name products-container.

Inside the div tag with class named products-container create 10 more div tags with same class name as card, each div tag should have a h2 tag and a p tag . H2 tag will be the name of the music instruments : Flute, Clarinet, Oboe, Trumpet, Trombone, French Horn, Drum Set, Xylophone, Cymbals, Marimba and the p tag will be the cost : $500, $200, $4000, $200, $300, $4300, $500, $3000, $200, $3000.   
create a dropdown menu with the class of select-container.

Now add four options to your dropdown, All, Woodwinds, Brass, and Percussion, in this order. For each option, use the lowercase option to specify the id attribute and the value attribute.

Before starting to add the filter functionality to your page, link your script.js file to the HTML.

Inside scipt.js file:-

Declare an empty array named instrumentsArr.

instrumentsArr is going to hold an object for each of your instrument cards. Each object will have the keys category, instrument, and price.

Add a first object representing the flute to your array. For the values use woodwinds, Flute, and 500, respectively.

Add another 9 objects to your instrumentsArr to represent the remaining woodwinds instruments, Clarinet, Oboe, Trumpet, Trombone, French Horn, Drum Set, Xylophone, Cymbals, Marimba.

Now, declare two variables selectContainer and productsContainer and assign them your HTML dropdown and the element with the class products-container, respectively.

As you learned, the change event is triggered when the user modifies the value of certain input elements. You want to be able to update your page any time that a new value is picked from the dropdown menu. For that, add an event listener for the change event to selectContainer.

For now, inside the callback, log the string this is a test to the console. Then, select different options from your dropdown menu and check the console to test that the event listener works correctly.

Modify your console.log call to log the selected value from the dropdown.

To implement the filter functionality, you'll need a function. Declare an empty function named instrumentCards that takes a single parameter.

Within your new function, you need to filter the instruments depending on the selected category.Filter out items from instrumentsArr and make your function return an array containing the instrument objects with the same category of instrumentCategory. If instrumentCategory is equal to all, return the whole instrumentsArr array.  
Then, remove the console.log from the callback of your event listener and log the result of calling instrumentCards with the selected option from the dropdown menu as argument so you can test your function selecting different category options.

Currently, your instrumentCards function returns an array with instrument objects, so you'll need another couple of steps before you can display your instrument cards on the page. Modify your function so that it returns an array of strings containing the HTML code to display the instrument cards, each string corresponding to an object in the instruments array. The strings should have this format <div class="card"><h2>[instrument]</h2><p>$[price]</p></div>

Remove your console.log from the event listener and set the inner HTML of productsContainer to the result of calling instrumentCards with the selected category option. Then, select different options from your dropdown and see the result in the preview window.

When you select a category from the dropdown menu, the instrument cards are correctly filtered and displayed on the page, but you have to get rid of those commas in between the cards. Do it by joining the array returned by instrumentCards. With that, your music instrument filter is complete.

Code:

index.html🡪

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Music Instruments product page</title>

    <link rel="stylesheet" href="./styles.css" />

  </head>

  <body>

    <h1>Student Instruments</h1>

    <main>

      <select class="select-container">

        <option id="all" value="all">All</option>

        <option id="woodwinds" value="woodwinds">Woodwinds</option>

        <option id="brass" value="brass">Brass</option>

        <option id="percussion" value="percussion">Percussion</option>

      </select>

      <div class="products-container">

        <div class="card">

          <h2>Flute</h2>

          <p>$500</p>

        </div>

        <div class="card">

          <h2>Clarinet</h2>

          <p>$200</p>

        </div>

        <div class="card">

          <h2>Oboe</h2>

          <p>$4000</p>

        </div>

        <div class="card">

          <h2>Trumpet</h2>

          <p>$200</p>

        </div>

        <div class="card">

          <h2>Trombone</h2>

          <p>$300</p>

        </div>

        <div class="card">

          <h2>French Horn</h2>

          <p>$4300</p>

        </div>

        <div class="card">

          <h2>Drum Set</h2>

          <p>$500</p>

        </div>

        <div class="card">

          <h2>Xylophone</h2>

          <p>$3000</p>

        </div>

        <div class="card">

          <h2>Cymbals</h2>

          <p>$200</p>

        </div>

        <div class="card">

          <h2>Marimba</h2>

          <p>$3000</p>

        </div>

      </div>

    </main>

    <script src="./script.js"></script>

  </body>

</html>

script.js🡪

const instrumentsArr = [

  { category: "woodwinds", instrument: "Flute", price: 500 },

  { category: "woodwinds", instrument: "Clarinet", price: 200 },

  { category: "woodwinds", instrument: "Oboe", price: 4000 },

  { category: "brass", instrument: "Trumpet", price: 200 },

  { category: "brass", instrument: "Trombone", price: 300 },

  { category: "brass", instrument: "French Horn", price: 4300 },

  { category: "percussion", instrument: "Drum Set", price: 500 },

  { category: "percussion", instrument: "Xylophone", price: 3000 },

  { category: "percussion", instrument: "Cymbals", price: 200 },

  { category: "percussion", instrument: "Marimba", price: 3000 }

]

const selectContainer = document.querySelector("select");

const productsContainer = document.querySelector(".products-container");

function instrumentCards(instrumentCategory) {

  const instruments =

    instrumentCategory === "all"

      ? instrumentsArr

      : instrumentsArr.filter(

          ({ category }) => category === instrumentCategory

        );

  return instruments

    .map(({ instrument, price }) => {

      return `

          <div class="card">

            <h2>${instrument}</h2>

            <p>$${price}</p>

          </div>

        `;

    })

}

selectContainer.addEventListener("change", () => {

//console.log("this is a test"); // 1st code.

//console.log(selectContainer.value); //2nd code

//console.log(instrumentCards(selectContainer.value)); //3rd code

//productsContainer.innerHTML = instrumentCards(selectContainer.value); //4th code

productsContainer.innerHTML = instrumentCards(selectContainer.value).join("");

});

styles.css🡪

\*,

\*::before,

\*::after {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

:root {

  --dark-grey: #0a0a23;

  --white: #ffffff;

  --yellow: #f1be32;

}

body {

  background-color: var(--dark-grey);

  color: var(--white);

}

h1 {

  text-align: center;

  margin-top: 20px;

}

.select-container {

  display: block;

  margin: 25px auto;

  padding: 8px;

  border: 4px solid var(--white);

  border-radius: 4px;

  width: 200px;

}

.products-container {

  display: flex;

  flex-direction: column;

  flex-wrap: wrap;

  align-items: center;

  justify-content: center;

  gap: 20px;

}

@media (min-width: 760px) {

  .products-container {

    flex-direction: row;

  }

}

.card {

  background-color: var(--white);

  color: var(--dark-grey);

  border: 4px solid var(--yellow);

  border-radius: 5px;

  padding: 10px;

  width: 200px;

}

Build a Real Time Counter:

In this lab, you will build a real-time character counter by using JavaScript. The character counter will display the number of characters entered in a textarea element. The counter will update in real-time as the user types in the textarea.

Fulfill the user stories below and get all the tests to pass to complete the lab. **Do not copy this demo project**.

**User Stories:**

1. You should have a textarea element with the id of text-input.
2. There should be a p element with the id of char-count. It should initially contain the text Character Count: 0/50. This placeholder text should be replaced with the current count of characters in the textarea element.
3. When the #text-input element contains the text hello coder the #char-count element should contain the text "Character Count: 11/50"
4. When the #text-input element contains the text i am a programmer the #char-count element should contain the text "Character Count: 17/50"
5. When the #text-input element contains the text hello world the #char-count element should contain the text "Character Count: 11/50"
6. When the #text-input element contains the text I am learning a new language and it is called c++. the #char-count element should contain the text "Character Count: 50/50". **NOTE**: While the maxlength attribute would achieve the correct functionality, this lab requires a JavaScript only solution.
7. The user should not be able to enter more than 50 characters in the textarea element. When the character count reaches 50, the text Character Count: 50/50 should be displayed in red.

**Note:** Be sure to link your stylesheet and the JavaScript file in your HTML.

* 1. You should have a textarea element with the id of text-input.
* 2. You should have a p element with the id of char-count.
* 3. The #char-count element should initially contain the text Character Count: 0/50.
* 4. When the #text-input element contains the text hello coder the #char-count element should contain the text "Character Count: 11/50"
* 5. When the character count is 50, the text should be displayed in red.
* 6. If character count is greater than or equal to 50, the user shouldn't be able to enter more characters.

Code:

index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Character Counter</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <textarea id="text-input" placeholder="Type here..."></textarea>

  <p id="char-count">Character Count: 0/50</p>

  <script src="script.js"></script>

</body>

</html>

script.js🡪

    const textInput = document.getElementById("text-input");

const charCount = document.getElementById("char-count");

const maxLength = 50;

textInput.addEventListener("input", () => {

  let text = textInput.value;

  // Trim input if it exceeds max length

  if (text.length > maxLength) {

    text = text.slice(0, maxLength);

    textInput.value = text;

  }

  // Update the character count display

  charCount.textContent = `Character Count: ${text.length}/50`;

  // Add or remove red class

  if (text.length >= maxLength) {

    charCount.classList.add("red");

  } else {

    charCount.classList.remove("red");

  }

});

styles.css🡪

#char-count {

  font-size: 1.2rem;

  margin-top: 10px;

}

.red {

  color: red;

}

Build a Lightbox Viewer:

A lightbox is used on websites to showcase multimedia content in a more engaging way through use of a popup or modal window over the page's main content.

In this lab, you will create a lightbox gallery that displays full-size images when a thumbnail is clicked. For each image, two versions are provided: a thumbnail and a full-size image. The full-size image is the same as the thumbnail, but without the -thumbnail suffix.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a div with a class of gallery within your body.
2. Within the .gallery element, you should have three image thumbnails, each with a class of gallery-item. You should use the following links for thumbnail images:
   * https://cdn.freecodecamp.org/curriculum/labs/stonehenge-thumbnail.jpg
   * https://cdn.freecodecamp.org/curriculum/labs/storm-thumbnail.jpg
   * https://cdn.freecodecamp.org/curriculum/labs/trees-thumbnail.jpg
3. You should have a div with a class of lightbox within your body.
4. You should have a button with an id of close-btn within your .lightbox element. You can use &times; as its text if you want.
5. You should have a img with an id of lightbox-image within your .lightbox element.
6. Your .lightbox element should have a fixed position so that the preview opens on top of the current images.
7. Your .lightbox element should cover the entire viewport by setting the height and width to 100% of the container. You should ensure that the .lightbox element starts at the top left corner of the container.
8. .lightbox should have a background color. Initially, its display property should be set to none to hide it.
9. When you click one of your .gallery-item elements, the .lightbox element's display property should be set to flex to make the .lightbox element and the two elements within it visible.
10. When you click one of your .gallery-item elements, the #lightbox-image element's src should be set to a full-size version of the image clicked by removing -thumbnail from the image's src attribute. The full-size images are located at the following links:
    * https://cdn.freecodecamp.org/curriculum/labs/stonehenge.jpg
    * https://cdn.freecodecamp.org/curriculum/labs/storm.jpg
    * https://cdn.freecodecamp.org/curriculum/labs/trees.jpg
11. When your .lightbox element is visible and you click the #close-btn or the .lightbox element, the .lightbox element's display should be set back to none.

**Note:** Be sure to link your stylesheet and the JavaScript file in your HTML.

* 1. You should have a div with the class of gallery inside your body element.
* 2. Within the .gallery element, you should have three img elements with the class of gallery-item.
* 3. Within the .gallery element, you should have an img element with the src set to https://cdn.freecodecamp.org/curriculum/labs/stonehenge-thumbnail.jpg.
* 4. Within the .gallery element, you should an img element with the src set to https://cdn.freecodecamp.org/curriculum/labs/storm-thumbnail.jpg.
* 5. Within the .gallery element, you should an img element with the src set to https://cdn.freecodecamp.org/curriculum/labs/trees-thumbnail.jpg.
* 6. You should have a div element with the class of lightbox inside your body element.
* 7. Within your .lightbox element, you should have a button element with the id set to close-btn.
* 8. Within your .lightbox element, you should have an img element with the id set to lightbox-image.
* 9. Your .lightbox element should have fixed positioning.
* 10. Your .lightbox element should be aligned with top left corner of the container.
* 11. Your .lightbox element should have a background color.
* 12. Your .lightbox element should be hidden initially.
* 13. When you click one of your .gallery-item elements, the .lightbox element's display property should be set to flex to make .lightbox and the two elements within it visible.
* 14. When you click one of your .gallery-item elements, the #lightbox-image element's src should be set to a full-size version of the image clicked by removing -thumbnail from the image's src attribute.
* 15. When your .lightbox element is visible and you click the #close-btn button, the .lightbox element's display should be set back to none.
* 16. When your .lightbox element is visible and you click the .lightbox element, the .lightbox element's display should be set back to none.

Code:

index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <title>Lightbox Gallery</title>

  <link rel="stylesheet" href="styles.css" />

</head>

<body>

  <div class="gallery">

    <img class="gallery-item" src="https://cdn.freecodecamp.org/curriculum/labs/stonehenge-thumbnail.jpg" alt="Stonehenge" />

    <img class="gallery-item" src="https://cdn.freecodecamp.org/curriculum/labs/storm-thumbnail.jpg" alt="Storm" />

    <img class="gallery-item" src="https://cdn.freecodecamp.org/curriculum/labs/trees-thumbnail.jpg" alt="Trees" />

  </div>

  <div class="lightbox" id="lightbox">

    <button id="close-btn">&times;</button>

    <img id="lightbox-image" src="" alt="Full-size" />

  </div>

  <script src="script.js"></script>

</body>

</html>

script.js🡪

const galleryItems = document.querySelectorAll('.gallery-item');

const lightbox = document.querySelector('.lightbox');

const lightboxImage = document.getElementById('lightbox-image');

const closeBtn = document.getElementById('close-btn');

galleryItems.forEach(item => {

  item.addEventListener('click', () => {

    const fullSizeSrc = item.src.replace('-thumbnail', '');

    lightboxImage.src = fullSizeSrc;

    lightbox.style.display = 'flex';

  });

});

closeBtn.addEventListener('click', () => {

  lightbox.style.display = 'none';

});

lightbox.addEventListener('click', (e) => {

  if (e.target === lightbox) {

    lightbox.style.display = 'none';

  }

});

styles.css🡪

.gallery {

  display: flex;

  gap: 10px;

  margin: 20px;

}

.gallery-item {

  width: 150px;

  cursor: pointer;

}

/\* Lightbox overlay \*/

.lightbox {

  position: fixed;        /\* Fixed positioning to stay over content \*/

  top: 0;                 /\* Start at top \*/

  left: 0;                /\* Start at left \*/

  width: 100vw;           /\* Full viewport width \*/

  height: 100vh;          /\* Full viewport height \*/

  background-color: rgba(0, 0, 0, 0.8); /\* Overlay \*/

  display: none;          /\* Hidden by default \*/

  align-items: center;

  justify-content: center;

  flex-direction: column;

  z-index: 1000;

}

#lightbox-image {

  max-width: 90%;

  max-height: 80%;

}

#close-btn {

  position: absolute;

  top: 20px;

  right: 30px;

  background: none;

  border: none;

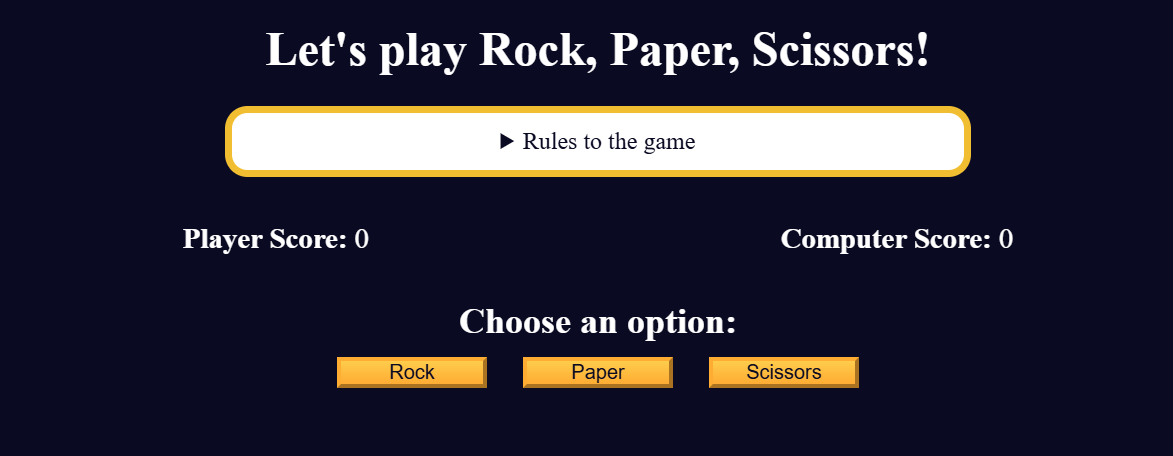
  font-size: 2rem;

  color: white;

  cursor: pointer;

}

Build a Rock, Paper, Scissor game :

Html & css🡪  


Clicking the Rules to the game it should show like this 🡪  
A white rectangular sign with black text

AI-generated content may be incorrect.

index.html🡪

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Rock, Paper, Scissors game</title>

    <link rel="stylesheet" href="./styles.css" />

  </head>

  <body>

    <h1>Let's play Rock, Paper, Scissors!</h1>

    <main>

      <details class="rules-container">

        <summary>Rules to the game</summary>

        <p>You will be playing against the computer.</p>

        <p>You can choose between Rock, Paper, and Scissors.</p>

        <p>The first one to three points wins.</p>

        <p>Here are the rules to getting a point in the game:</p>

        <ul>

          <li>Rock beats Scissors</li>

          <li>Scissors beats Paper</li>

          <li>Paper beats Rock</li>

        </ul>

        <p>

          If the player and computer choose the same option (Ex. Paper and

          Paper), then no one gets the point.

        </p>

      </details>

      <div class="score-container">

        <strong

          >Player Score: <span class="score" id="player-score">0</span></strong

        >

        <strong

          >Computer Score:

          <span class="score" id="computer-score">0</span></strong

        >

      </div>

      <section class="options-container">

        <h2>Choose an option:</h2>

        <div class="btn-container">

          <button id="rock-btn" class="btn">Rock</button>

          <button id="paper-btn" class="btn">Paper</button>

          <button id="scissors-btn" class="btn">Scissors</button>

        </div>

      </section>

      <div class="results-container">

        <p id="results-msg"></p>

        <p id="winner-msg"></p>

        <button class="btn" id="reset-game-btn">Play again?</button>

      </div>

    </main>

    <script src="./script.js"></script>

  </body>

</html>

styles.css

\*,

\*::before,

\*::after {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

:root {

  --very-dark-blue: #0a0a23;

  --white: #ffffff;

  --yellow: #f1be32;

  --golden-yellow: #feac32;

}

body {

  background-color: var(--very-dark-blue);

  text-align: center;

  color: var(--white);

}

h1 {

  margin: 15px 0 20px;

}

.btn {

  cursor: pointer;

  width: 100px;

  margin: 10px;

  color: var(--very-dark-blue);

  background-color: var(--golden-yellow);

  background-image: linear-gradient(#fecc4c, #ffac33);

  border-color: var(--golden-yellow);

  border-width: 3px;

}

.btn:hover {

  background-image: linear-gradient(#ffcc4c, #f89808);

}

.rules-container {

  padding: 10px 0;

  margin: auto;

  border-radius: 15px;

  border: 5px solid var(--yellow);

  background-color: var(--white);

  color: var(--very-dark-blue);

}

.rules-container ul {

  list-style-type: none;

}

.rules-container p {

  margin: 10px 0;

}

@media (min-width: 760px) {

  .rules-container {

    width: 60%;

  }

}

.score-container {

  display: flex;

  justify-content: space-around;

  margin: 30px 0;

  font-size: 1.2rem;

}

.score {

  font-weight: 500;

}

.results-container {

  font-size: 1.3rem;

  margin: 15px 0;

}

#winner-msg {

  margin-top: 25px;

}

#reset-game-btn {

  display: none;

  margin: 20px auto;

}

Start by creating a variable called options and assign it an array with the following strings: "Rock", "Paper", and "Scissors".  
The next step is to build out the functionality that will generate a random choice for the computer.

Create a function called getRandomComputerResult that returns a random choice from the options array.

Hint: Don't forget that in the earlier JavaScript fundamentals section, you learned about using Math.random() and Math.floor().

In this step, you will focus on determining if the player has won the round.

Create a hasPlayerWonTheRound function with two parameters called player and computer.

The function should return true if the player has won the round, and false if the player has lost or tied the round.

Here are the criteria for the player to win a round:

* If the player chooses "Rock" and the computer chooses "Scissors"
* If the player chooses "Scissors" and the computer chooses "Paper"
* If the player chooses "Paper" and the computer chooses "Rock"

The next step is to create the variables responsible for keeping track of the player and computer scores.

Create a variable called playerScore and initialize it with the value 0.

Then, create a variable called computerScore and initialize it with the value 0.

You will need to use let to declare these variables because their values will change throughout the game.

Now it's time to get the results of the round. Complete the getRoundResults function.

If the player wins the round, update the playerScore by 1 and return the message "Player wins! [player's choice] beats [computer's choice]".

If the computer and player choose the same option, return the message "It's a tie! Both chose [player's choice]".

If the computer wins the round, update the computerScore by 1 and return the message "Computer wins! [computer's choice] beats [player's choice]".

[computer's choice] should be replaced with computerResult while [player's choice] should be replaced with the userOption.

Start by creating a variable called playerScoreSpanElement and assign it the value of the element with the id player-score.

Then create a variable called computerScoreSpanElement and assign it the value of the element with the id computer-score.

Finally, create a variable called roundResultsMsg and assign it the value of the element with the id results-msg.

Now it is time to update the scores and the round results message.

Create a showResults function with a parameter called userOption.

Inside your showResults function, the roundResultsMsg should be updated with the result of the round.

Then, the playerScoreSpanElement and computerScoreSpanElement should also be updated to show the updated scores of the player and computer.

Remember, that the order matters here. You will need to first update the roundResultsMsg, then the playerScoreSpanElement, and finally the computerScoreSpanElement because the roundResultsMsg will be used to determine the scores.

Now it is time to test out your showResults function.

Start by creating a variable called rockBtn and assign it the value of the button with the id of rock-btn.

Then, create a variable called paperBtn and assign it the value of the button with the id of paper-btn.

Finally, create a variable called scissorsBtn and assign it the value of the button with the id of scissors-btn.

Start by creating an event listener for the rockBtn button. When the rockBtn button is clicked, you should call the showResults function with the argument "Rock".

Then, create an event listener for the paperBtn button. When the paperBtn button is clicked, you should call the showResults function with the argument "Paper".

Finally, create an event listener for the scissorsBtn button. When the scissorsBtn button is clicked, you should call the showResults function with the argument "Scissors".

Now, you should be able to click on any of the buttons and see the results for each round.

Start by creating a variable called winnerMsgElement that will store the element with the id of winner-msg.

Then, create a variable called optionsContainer that will store the element with the class of options-container.

Finally, create a variable called resetGameBtn that will store the element with the id of reset-game-btn.

If you try to play the game, you will see that you can play for an infinite amount of rounds. But the rules state that the first one to three points wins. You want to check if there's a winner, and display a message.

In your showResults function, if the player has reached three points, update the winnerMsgElement to "Player has won the game!". If the computer has reached three points, update the winnerMsgElement to "Computer has won the game!".

If there is a winner, show the resetGameBtn button by setting its display to block and hide the optionsContainer by setting its display to none.

Now, try to play the game and see if the winner message is displayed when a player reaches three points.

If the player or computer has won the game, there should be an option to reset the game and play again.

Add an event listener to the resetGameBtn button. Your event listener should take in a "click" event and a reference to the resetGame function.

For the final step of the workshop, you will need to build out the reset game functionality.

Create a resetGame function that accomplishes the following:

* Resets the player and computer scores to 0.
* Updates the playerScoreSpanElement and computerScoreSpanElement to display the new scores.
* Hides the resetGameBtn button.
* Shows the optionsContainer so the player can play again.
* Clears the content for the winnerMsgElement and roundResultsMsg elements.

Try testing out the game by playing a few rounds until one of the players reaches 3 points. Then, click the "Play again?" button to see if the game resets correctly.

And with this final step, you have completed the Rock, Paper, Scissors game!

script.js🡪

const options = ["Rock", "Paper", "Scissors"];

function getRandomComputerResult() {

  const randomIndex = Math.floor(Math.random() \* options.length);

  return options[randomIndex];

}

function hasPlayerWonTheRound(player, computer) {

  return (

    (player === "Rock" && computer === "Scissors") ||

    (player === "Scissors" && computer === "Paper") ||

    (player === "Paper" && computer === "Rock")

  );

}

let playerScore = 0;

let computerScore = 0;

function getRoundResults(userOption) {

  const computerResult = getRandomComputerResult();

  if (hasPlayerWonTheRound(userOption, computerResult)) {

    playerScore++;

    return `Player wins! ${userOption} beats ${computerResult}`;

  } else if (computerResult === userOption) {

    return `It's a tie! Both chose ${userOption}`;

  } else {

    computerScore++;

    return `Computer wins! ${computerResult} beats ${userOption}`;

  }

}

const playerScoreSpanElement = document.getElementById("player-score");

const computerScoreSpanElement = document.getElementById("computer-score");

const roundResultsMsg = document.getElementById("results-msg");

const winnerMsgElement = document.getElementById("winner-msg");

const optionsContainer = document.querySelector(".options-container");

const resetGameBtn = document.getElementById("reset-game-btn");

function showResults(userOption) {

  roundResultsMsg.innerText = getRoundResults(userOption);

  computerScoreSpanElement.innerText = computerScore;

  playerScoreSpanElement.innerText = playerScore;

  if (playerScore === 3 || computerScore === 3) {

    winnerMsgElement.innerText = `${

      playerScore === 3 ? "Player" : "Computer"

    } has won the game!`;

    resetGameBtn.style.display = "block";

    optionsContainer.style.display = "none";

  }

};

function resetGame() {

  // Reset scores

  playerScore = 0;

  computerScore = 0;

  // Update score display

  playerScoreSpanElement.innerText = playerScore;

  computerScoreSpanElement.innerText = computerScore;

  // Hide reset button and show options

  resetGameBtn.style.display = "none";

  optionsContainer.style.display = "flex"; // or "block", depending on your layout

  // Clear messages

  winnerMsgElement.innerText = "";

  roundResultsMsg.innerText = "";

}

resetGameBtn.addEventListener("click", resetGame);

const rockBtn = document.getElementById("rock-btn");

const paperBtn = document.getElementById("paper-btn");

const scissorsBtn = document.getElementById("scissors-btn");

rockBtn.addEventListener("click", function () {

  showResults("Rock");

});

paperBtn.addEventListener("click", function () {

  showResults("Paper");

});

scissorsBtn.addEventListener("click", function () {

  showResults("Scissors");

});

Build a Set of Football Team Cards:

In this lab, you will build a set of football team cards. The user should be able to use the dropdown menu and filter between the different players based on their positions.

**User Stories:**

1. You should create a footballTeam object with the following properties: team, year, headCoach, players.
2. The team property should contain the name of the team as a string.
3. The year property should contain the year as a number.
4. The headCoach property should contain the name of the head coach as a string.
5. The players property should be an array containing at least four elements.
6. Each element in the players array should be an object with properties name, position, isCaptain.
7. The name property should contain the name of the player as a string.
8. The position property should have one of the following values: "forward", "midfielder", "defender", or "goalkeeper".
9. The isCaptain property should have value of a boolean. One of the players should have their isCaptain property set to true.
10. You should display the coach, team and year values on the page. These values should be displayed in the HTML elements with the id values of head-coach, team and year.
11. You should display the players data on the page inside the #player-cards element, each player should be displayed in a div with class of player-card, and nested in it, an h2 containing the name of the player, and (Captain) in case of the player being captain, and a p containing Position: and the position of the player.

<div class="player-card">

<h2>Sergio Batista</h2>

<p>Position: midfielder</p>

</div>

<div class="player-card">

<h2>(Captain) Diego Maradona</h2>

<p>Position: midfielder</p>

</div>

1. When the dropdown menu is used to select one of the positions, only players of that position should be shown. If the "All Players" option is selected, then all of the players should display on the page.

1. You should have a footballTeam variable.

2. The footballTeam variable should be an object with four properties: team, year, headCoach and players.

3. The team property should be a string.

4. The year property should be a number.

5. The headCoach property should be a string.

6. The players property should be an array of at least four objects, each object should have the keys name, position, isCaptain.

7. The name property should have value of a string.

8. The position property should have one of values "forward", "midfielder", "defender", or "goalkeeper".

9. The isCaptain property should have value of a boolean, and there should be only one captain.

10. You should display the coach, team and year values from the footballTeam object in the HTML elements with the id values of head-coach, team and year.

11. When the option All Players is selected, all players should be shown within #player-cards.

12. When the option Position Forward is selected, only forward players should be shown within #player-cards.

13. When the option Position Midfielder is selected, only midfielder players should be shown within #player-cards.

14. When the option Position Defender is selected, only defender players should be shown within #player-cards.

15. When the option Position Goalkeeper is selected, only goalkeeper players should be shown.

Code:

index.html🡪  
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>

      Build a Set of Football Team Cards

    </title>

    <link rel="stylesheet" href="styles.css" />

  </head>

  <body>

    <h1 class="title">Team stats</h1>

    <main>

      <div class="team-stats">

        <p>Team: <span id="team"></span></p>

        <p>Year: <span id="year"></span></p>

        <p>Head coach: <span id="head-coach"></span></p>

      </div>

      <label class="options-label" for="players">Filter Teammates:</label>

      <select name="players" id="players">

        <option value="all">All Players</option>

        <option value="forward">Position Forward</option>

        <option value="midfielder">Position Midfielder</option>

        <option value="defender">Position Defender</option>

        <option value="goalkeeper">Position Goalkeeper</option>

      </select>

      <div class="cards" id="player-cards"></div>

    </main>

    <footer>&copy; freeCodeCamp</footer>

    <script src="./script.js"></script>

  </body>

</html>

script.js🡪  
// 1. Define the football team object

const footballTeam = {

  team: "Argentina",

  year: 1986,

  headCoach: "Carlos Bilardo",

  players: [

    { name: "Diego Maradona", position: "midfielder", isCaptain: true },

    { name: "Jorge Valdano", position: "forward", isCaptain: false },

    { name: "Oscar Ruggeri", position: "defender", isCaptain: false },

    { name: "Nery Pumpido", position: "goalkeeper", isCaptain: false }

  ]

};

// 2. Display team info on the page

document.getElementById("team").textContent = footballTeam.team;

document.getElementById("year").textContent = footballTeam.year;

document.getElementById("head-coach").textContent = footballTeam.headCoach;

// 3. Function to render players based on position filter

function renderPlayers(position) {

  const playerCardsContainer = document.getElementById("player-cards");

  playerCardsContainer.innerHTML = ""; // Clear previous cards

  const filteredPlayers = footballTeam.players.filter(player => {

    return position === "all" || player.position === position;

  });

  filteredPlayers.forEach(player => {

    const playerCard = document.createElement("div");

    playerCard.className = "player-card";

    const playerName = document.createElement("h2");

    playerName.textContent = player.isCaptain

      ? `(Captain) ${player.name}`

      : player.name;

    const playerPosition = document.createElement("p");

    playerPosition.textContent = `Position: ${player.position}`;

    playerCard.appendChild(playerName);

    playerCard.appendChild(playerPosition);

    playerCardsContainer.appendChild(playerCard);

  });

}

// 4. Initial render

renderPlayers("all");

// 5. Event listener for dropdown change

document.getElementById("players").addEventListener("change", function () {

  const selectedPosition = this.value;

  renderPlayers(selectedPosition);

});

styles.css🡪

\*,

\*::before,

\*::after {

  box-sizing: border-box;

  margin: 0;

  padding: 0;

}

:root {

  --dark-grey: #0a0a23;

  --light-grey: #f5f6f7;

  --white: #ffffff;

  --black: #000;

}

body {

  background-color: var(--dark-grey);

  text-align: center;

  padding: 10px;

}

.title,

.options-label,

.team-stats,

footer {

  color: var(--white);

}

.title {

  margin: 1.3rem 0;

}

.team-stats {

  display: flex;

  justify-content: space-around;

  flex-wrap: wrap;

  font-size: 1.3rem;

  margin: 1.2rem 0;

}

.options-label {

  font-size: 1.2rem;

}

.cards {

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

  align-items: center;

}

.player-card {

  background-color: var(--light-grey);

  padding: 1.3rem;

  margin: 1.2rem;

  width: 300px;

  border-radius: 15px;

}

@media (max-width: 768px) {

  .team-stats {

    flex-direction: column;

  }

}

Build a Palindrome Checker:

A *palindrome* is a word or phrase that can be read the same way forwards and backwards, ignoring punctuation, case, and spacing.

**Note:** You'll need to remove **all non-alphanumeric characters** (punctuation, spaces and symbols) and turn everything into the same case (lower or upper case) in order to check for palindromes.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have an input element with an id of "text-input".
2. You should have a button element with an id of "check-btn".
3. You should have a div, span or p element with an id of "result".
4. When you click on the #check-btn element without entering a value into the #text-input element, an alert should appear with the text Please input a value.
5. When the #text-input element only contains the letter A and the #check-btn element is clicked, the #result element should contain the text A is a palindrome.
6. When the #text-input element contains the text eye and the #check-btn element is clicked, the #result element should contain the text eye is a palindrome.
7. When the #text-input element contains the text \_eye and the #check-btn element is clicked, the #result element should contain the text \_eye is a palindrome.
8. When the #text-input element contains the text race car and the #check-btn element is clicked, the #result element should contain the text race car is a palindrome.
9. When the #text-input element contains the text not a palindrome and the #check-btn element is clicked, the #result element should contain the text not a palindrome is not a palindrome.
10. When the #text-input element contains the text A man, a plan, a canal. Panama and the #check-btn element is clicked, the #result element should contain the text A man, a plan, a canal. Panama is a palindrome.
11. When the #text-input element contains the text never odd or even and the #check-btn element is clicked, the #result element should contain the text never odd or even is a palindrome.
12. When the #text-input element contains the text nope and the #check-btn element is clicked, the #result element should contain the text nope is not a palindrome.
13. When the #text-input element contains the text almostomla and the #check-btn element is clicked, the #result element should contain the text almostomla is not a palindrome.
14. When the #text-input element contains the text My age is 0, 0 si ega ym. and the #check-btn element is clicked, the #result element should contain the text My age is 0, 0 si ega ym. is a palindrome.
15. When the #text-input element contains the text 1 eye for of 1 eye. and the #check-btn element is clicked, the #result element should contain the text 1 eye for of 1 eye. is not a palindrome.
16. When the #text-input element contains the text 0\_0 (: /-\ :) 0-0 and the #check-btn element is clicked, the #result element should contain the text 0\_0 (: /-\ :) 0-0 is a palindrome.
17. When the #text-input element contains the text five|\\_/|four and the #check-btn element is clicked, the #result element should contain the text five|\\_/|four is not a palindrome.

* 1. You should have an input element with an id of "text-input".
* 2. You should have a button element with an id of "check-btn".
* 3. You should have a div, span, or p element with an id of "result".
* 4. When you click on the #check-btn element without entering a value into the #text-input element, an alert should appear with the text Please input a value.
* 5. When the #text-input element only contains the letter A and the #check-btn element is clicked, the #result element should contain the text A is a palindrome.
* 6. When the #text-input element contains the text eye and the #check-btn element is clicked, the #result element should contain the text eye is a palindrome.
* 7. When the #text-input element contains the text \_eye and the #check-btn element is clicked, the #result element should contain the text \_eye is a palindrome.
* 8. When the #text-input element contains the text race car and the #check-btn element is clicked, the #result element should contain the text race car is a palindrome.
* 9. When the #text-input element contains the text not a palindrome and the #check-btn element is clicked, the #result element should contain the text not a palindrome is not a palindrome.
* 10. When the #text-input element contains the text A man, a plan, a canal. Panama and the #check-btn element is clicked, the #result element should contain the text A man, a plan, a canal. Panama is a palindrome.
* 11. When the #text-input element contains the text never odd or even and the #check-btn element is clicked, the #result element should contain the text never odd or even is a palindrome.
* 12. When the #text-input element contains the text nope and the #check-btn element is clicked, the #result element should contain the text nope is not a palindrome.
* 13. When the #text-input element contains the text almostomla and the #check-btn element is clicked, the #result element should contain the text almostomla is not a palindrome.
* 14. When the #text-input element contains the text My age is 0, 0 si ega ym. and the #check-btn element is clicked, the #result element should contain the text My age is 0, 0 si ega ym. is a palindrome.
* 15. When the #text-input element contains the text 1 eye for of 1 eye. and the #check-btn element is clicked, the #result element should contain the text 1 eye for of 1 eye. is not a palindrome.
* 16. When the #text-input element contains the text 0\_0 (: /-\ :) 0-0 and the #check-btn element is clicked, the #result element should contain the text 0\_0 (: /-\ :) 0-0 is a palindrome.
* 17. When the #text-input element contains the text five|\\_/|four and the #check-btn element is clicked, the #result element should contain the text five|\\_/|four is not a palindrome.
* 18. When the #text-input element contains an alphanumeric palindrome, the #result element should correctly identify it as a palindrome.
* 19. When the #text-input element contains a random sequence of alphanumeric characters that is not a palindrome, the #result element should say it is not a palindrome.

Code:

index.html🡪  
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Palindrome Checker</title>

  <link rel="stylesheet" href="style.css">

</head>

<body>

  <h1>Palindrome Checker</h1>

  <input id="text-input" type="text" placeholder="Enter text..." />

  <button id="check-btn">Check</button>

  <p id="result"></p>

  <script src="script.js"></script>

</body>

</html>

script.js🡪  
const input = document.getElementById("text-input");

const button = document.getElementById("check-btn");

const result = document.getElementById("result");

function isPalindrome(str) {

  const cleaned = str.replace(/[^A-Za-z0-9]/g, "").toLowerCase();

  const reversed = cleaned.split("").reverse().join("");

  return cleaned === reversed;

}

button.addEventListener("click", () => {

  const text = input.value;

  if (text.trim() === "") {

    alert("Please input a value");

    return;

  }

  const message = isPalindrome(text)

    ? `${text} is a palindrome.`

    : `${text} is not a palindrome.`;

  result.textContent = message;

});

styles.css🡪

body {

  font-family: Arial, sans-serif;

  text-align: center;

  padding: 50px;

}

input {

  padding: 10px;

  width: 250px;

  font-size: 16px;

}

button {

  padding: 10px 20px;

  font-size: 16px;

  margin-left: 10px;

  cursor: pointer;

}

#result {

  margin-top: 20px;

  font-size: 18px;

  font-weight: bold;

}

Build a Markdown to HTML Converter:

Markdown is a markup language used to add formatting elements to plain-text documents. For this lab, all the HTML and CSS has been provided to you. You will use JavaScript to complete the Markdown to HTML Converter app so that it can handle the conversion of basic Markdown constructs into HTML elements.

**Note:** The final result won't be a comprehensive Markdown to HTML converter, but you can add extra functionalities to it if you would like.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a function named convertMarkdown that takes no parameters.
2. The convertMarkdown function should use regular expressions to convert the markdown input from #markdown-input into HTML and should return a string containing the HTML code.
3. The convertMarkdown function should convert headings of level one, two, and three into the corresponding h1, h2, and h3 elements. A heading in markdown is indicated by as many # character as its level followed by a space and the heading text. # characters should either be placed at the beginning of the line or be preceded by space characters.
4. The convertMarkdown function should convert bold text into strong elements. Bold text in markdown is indicated either by a pair of double asterisks or a pair of double underscores enclosing the text.
5. The convertMarkdown function should convert italic text into em elements. Italic text in markdown is indicated either by a pair of asterisks or a pair of underscores enclosing the text.
6. The convertMarkdown function should convert images into img elements. An image in markdown is indicated by ![alt-text](image-source), where alt-text is the value of the alt attribute and image-source is the value of the src attribute.
7. The convertMarkdown function should convert links into anchor elements. A link in markdown is indicated by [link text](URL), where link text is the text to enclosed within the anchor tags and URL is the value of href attribute.
8. The convertMarkdown function should convert quotes into blockquote elements. A quote in markdown is indicated by a > followed by a space and the quote text. The > character should be either placed at the beginning of the line or be preceded by space characters.
9. When you input text inside #markdown-input, the raw HTML code returned by convertMarkdown should be displayed inside #html-output.
10. When you input text inside #markdown-input, the HTML code returned by convertMarkdown should be rendered inside #preview.

Here's a table containing all the markdown that convertMarkdown should be able to handle and the expected HTML after conversion:

| **Markdown** | **HTML** |
| --- | --- |
| # heading 1 | <h1>heading 1</h1> |
| ## heading 2 | <h2>heading 2</h2> |
| ### heading 3 | <h3>heading 3</h3> |
| \*\*bold text\*\* or \_\_bold text\_\_ | <strong>bold text</strong> |
| \*italic text\* or \_italic text\_ | <em>italic text</em> |
| ![alt-text](image-source) | <img alt="alt-text" src="image-source"> |
| [link text](URL) | <a href="URL">link text</a> |
| > quote | <blockquote>quote</blockquote> |

**Note:** Be sure to link your JavaScript file in your HTML.

* 1. You should have a function named convertMarkdown.
* 2. When the value of #markdown-input is # title 1, convertMarkdown() should return <h1>title 1</h1>.
* 3. When the value of #markdown-input is # title 1, <h1>title 1</h1> should be displayed inside #html-output.
* 4. When the value of #markdown-input is # title 1, an h1 element with the text of title 1 should be appended as a child of #preview.
* 5. When the value of #markdown-input is some text # title 1, convertMarkdown() should not convert # title 1 into an h1 element.
* 6. When the value of #markdown-input is # title 1 followed by # alternate title on a new line, convertMarkdown() should return <h1>title 1</h1><h1>alternate title</h1>.
* 7. When the value of #markdown-input is ## title 2, convertMarkdown() should return <h2>title 2</h2>.
* 8. When the value of #markdown-input is ## title 2, <h2>title 2</h2> should be displayed inside #html-output.
* 9. When the value of #markdown-input is ## title 2, an h2 element with the text of title 2 should be appended as a child of #preview.
* 10. When the value of #markdown-input is some text ## title 2, convertMarkdown() should not convert ## title 2 into an h2 element.
* 11. When the value of #markdown-input is ## title 2 followed by ## title 2 alt on a new line, convertMarkdown() should return <h2>title 2</h2><h2>title 2 alt</h2>.
* 12. When the value of #markdown-input is ### title 3, convertMarkdown() should return <h3>title 3</h3>.
* 13. When the value of #markdown-input is ### title 3, <h3>title 3</h3> should be displayed inside #html-output.
* 14. When the value of #markdown-input is ### title 3, an h3 element with the text of title 3 should be appended as a child of #preview.
* 15. When the value of #markdown-input is some text ### title 3, convertMarkdown() should not convert ### title 3 into an h3 element.
* 16. When the value of #markdown-input is ### title 3 followed by ### third title on a new line, convertMarkdown() should return <h3>title 3</h3><h3>third title</h3>.
* 17. When the value of #markdown-input is \*\*this is bold\*\*, convertMarkdown() should return <strong>this is bold</strong>.
* 18. When the value of #markdown-input is \*\*this is bold\*\*, <strong>this is bold</strong> should be displayed inside #html-output.
* 19. When the value of #markdown-input is \*\*this is bold\*\*, a strong element with the text of this is bold should be appended as a child of #preview.
* 20. When the value of #markdown-input is \*\*this is bold\*\* followed by \*\*this is also bold\*\* on a new line, convertMarkdown() should return <strong>this is bold</strong><strong>this is also bold</strong>.
* 21. When the value of #markdown-input is \_\_this is bold\_\_, <strong>this is bold</strong> should be displayed inside #html-output.
* 22. When the value of #markdown-input is \_\_this is bold\_\_, a strong element with the text of this is bold should be appended as a child of #preview.
* 23. When the value of #markdown-input is \_\_this is bold\_\_ followed by \_\_this is also bold\_\_ on a new line, convertMarkdown() should return <strong>this is bold</strong><strong>this is also bold</strong>.
* 24. When the value of #markdown-input is \*this is italic\*, convertMarkdown() should return <em>this is italic</em>.
* 25. When the value of #markdown-input is \*this is italic\*, <em>this is italic</em> should be displayed inside #html-output.
* 26. When the value of #markdown-input is \*this is italic\*, an em element with the text of this is italic should be appended as a child of #preview.
* 27. When the value of #markdown-input is \*this is italic\* followed by \*this is also italic\* on a new line, convertMarkdown() should return <em>this is italic</em><em>this is also italic</em>.
* 28. When the value of #markdown-input is \_this is italic\_, convertMarkdown() should return <em>this is italic</em>.
* 29. When the value of #markdown-input is \_this is italic\_, <em>this is italic</em> should be displayed inside #html-output.
* 30. When the value of #markdown-input is \_this is italic\_, an em element with the text of this is italic should be appended as a child of #preview.
* 31. When the value of #markdown-input is \_this is italic\_ followed by \_this is also italic\_ on a new line, convertMarkdown() should return <em>this is italic</em><em>this is also italic</em>.
* 32. When the value of #markdown-input is either # \*\*title 1\*\* or # \_\_title 1\_\_, convertMarkdown() should return <h1><strong>title 1</strong></h1>.
* 33. When the value of #markdown-input is either # \*\*title 1\*\* or # \_\_title 1\_\_, <h1><strong>title 1</strong></h1> should be displayed inside #html-output.
* 34. When the value of #markdown-input is either # \*\*title 1\*\* or # \_\_title 1\_\_, you set the inner HTML of #preview to <h1><strong>title 1</strong></h1>.
* 35. When the value of #markdown-input is ![alt-text](image-source), convertMarkdown() should return <img alt="alt-text" src="image-source">.
* 36. When the value of #markdown-input is ![alt-text](image-source), <img alt="alt-text" src="image-source"> should be displayed inside #html-output.
* 37. When the value of #markdown-input is ![alt-text](image-source), <img alt="alt-text" src="image-source"> should be appended as a child of #preview.
* 38. When the value of #markdown-input is ![alt-text](image-source) followed by ![alt-text-2](image-source-2) on a new line, convertMarkdown() should return <img alt="alt-text" src="image-source"><img alt="alt-text-2" src="image-source-2">.
* 39. When the value of #markdown-input is [link text](URL), convertMarkdown() should return <a href="URL">link text</a>.
* 40. When the value of #markdown-input is [link text](URL), <a href="URL">link text</a> should be displayed inside #html-output.
* 41. When the value of #markdown-input is [link text](URL), <a href="URL">link text</a> should be appended as a child of #preview.
* 42. When the value of #markdown-input is [link text](URL) followed by [link text 2](URL2) on a new line, convertMarkdown() should return <a href="URL">link text</a><a href="URL2">link text 2</a>.
* 43. When the value of #markdown-input is > this is a quote, convertMarkdown() should return <blockquote>this is a quote</blockquote>.
* 44. When the value of #markdown-input is > this is a quote, <blockquote>this is a quote</blockquote> should be displayed inside #html-output.
* 45. When the value of #markdown-input is > this is a quote, <blockquote>this is a quote</blockquote> should be appended as a child of #preview.
* 46. When the value of #markdown-input is > this is a quote followed by > this is another quote on a new line, convertMarkdown() should return <blockquote>this is a quote</blockquote><blockquote>this is another quote</blockquote>.
* 47. When the value of #markdown-input is some text > not a quote anymore, convertMarkdown() should not convert > not a quote anymore into a blockquote element.
* 48. When the value of #markdown-input is > \*\*this is a \*quote\*\*\*, convertMarkdown() should return <blockquote><strong>this is a <em>quote</em></strong></blockquote>.
* 49. When the value of #markdown-input is > \*\*this is a \*quote\*\*\*, <blockquote><strong>this is a <em>quote</em></strong></blockquote> should be displayed inside #html-output.
* 50. When the value of #markdown-input is > \*\*this is a \*quote\*\*\*, you should set the inner HTML of #preview to <blockquote><strong>this is a <em>quote</em></strong></blockquote>.
* 51. You should have only one script element in your HTML.

Code:

index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Markdown to HTML Converter</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1>Markdown to HTML Converter</h1>

    <div id="container">

        <div class="container">

            <h2>Markdown Input:</h2>

            <textarea id="markdown-input" placeholder="Enter your markdown here..."></textarea>

        </div>

        <div class="container">

            <h2>Raw HTML Output:</h2>

            <div id="html-output"></div>

        </div>

        <div class="container">

            <h2>HTML Preview:</h2>

            <div id="preview"></div>

        </div>

    </div>

      <script src="script.js"></script>

</body>

</html>

script.js🡪

function convertMarkdown() {

  const input = document.getElementById("markdown-input").value;

  const lines = input.split("\n");

  let output = "";

  for (let line of lines) {

    // Trim leading spaces for checking valid markdown syntax

    const trimmedLine = line.trimStart();

    // Skip if line doesn't start correctly for heading/blockquote

    let htmlLine = line;

    // Headings

    if (/^#{1,3} /.test(trimmedLine)) {

      const match = trimmedLine.match(/^(#{1,3}) (.\*)/);

      if (match) {

        const level = match[1].length;

        const content = match[2];

        htmlLine = `<h${level}>${content}</h${level}>`;

      }

    }

    // Blockquote

    else if (/^> /.test(trimmedLine)) {

      htmlLine = `<blockquote>${trimmedLine.slice(2)}</blockquote>`;

    }

    // Image

    else if (/^!\[.\*?\]\(.\*?\)$/.test(trimmedLine)) {

      htmlLine = trimmedLine.replace(/!\[(.\*?)\]\((.\*?)\)/g, `<img alt="$1" src="$2">`);

    }

    // Link

    else if (/^\[.\*?\]\(.\*?\)$/.test(trimmedLine)) {

      htmlLine = trimmedLine.replace(/\[(.\*?)\]\((.\*?)\)/g, `<a href="$2">$1</a>`);

    }

    // Bold (must handle strong before em to avoid nesting issues)

    htmlLine = htmlLine.replace(/(\\*\\*|\_\_)(.\*?)\1/g, `<strong>$2</strong>`);

    // Italic

    htmlLine = htmlLine.replace(/(\\*|\_)(.\*?)\1/g, `<em>$2</em>`);

    // Image or link in any line

    htmlLine = htmlLine.replace(/!\[(.\*?)\]\((.\*?)\)/g, `<img alt="$1" src="$2">`);

    htmlLine = htmlLine.replace(/\[(.\*?)\]\((.\*?)\)/g, `<a href="$2">$1</a>`);

    output += htmlLine;

  }

  // Update HTML Output and Preview

  document.getElementById("html-output").6L = output;

  return output;

}

// Update on input change

document.getElementById("markdown-input").addEventListener("input", convertMarkdown);

styles.css🡪

\* {

     box-sizing: border-box;

}

 body {

     font-family: Arial, sans-serif;

     padding: 20px;

}

 #markdown-input {

     width: 100%;

     height: 100px;

}

 #html-output, #preview {

     height: 100px;

     display: inline-block;

     width: 100%;

     border: 1px solid #ccc;

     padding: 10px;

     margin: auto;

     white-space: pre-wrap;

     background-color: #f9f9f9;

}

 @media (min-width: 600px) {

     #markdown-input, #html-output, #preview {

         height: 200px;

         margin: 0;

    }

     #container {

         display: flex;

         justify-content: space-evenly;

         gap: 10px;

    }

}

Build a RegEx Sandbox [JS need to do]:

For this lab, you start with the CSS and HTML already written for you. You will use JavaScript to enable the regex sandbox to test a regular expression against a string and highlight the results.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should access the #pattern, #test-string, #test-btn, and #result elements and save them in the variables regexPattern, stringToTest, testButton, and testResult, respectively.
2. You should access the #i and #g flag checkboxes and save them in the variables caseInsensitiveFlag and globalFlag, respectively.
3. You should have a function named getFlags that returns a string containing the flags from checked flag checkboxes, or an empty string if none of them have been checked.
4. When you click the testButton button, if the regex pattern matches the test string, the matched text should be highlighted. To highlight the matches, replace each match with the same text surrounded by a span element with the class of highlight. Note that the CSS for that has been already added for you.
5. When there's a match, the matched text should be displayed inside #result. In case of multiple matches, each matched text should be separated from the next one by a comma and a space.
6. When there's no match, the text no match should be displayed inside #result.

* 1. You should access the #pattern element and save it in the regexPattern variable.
* 2. You should access the #test-string element and save it in the stringToTest variable.
* 3. You should access the #test-btn element and save it in the testButton variable.
* 4. You should access the #result element and save it in the testResult variable.
* 5. You should access the #i flag checkbox and save it in the caseInsensitiveFlag.
* 6. You should access the #g flag checkbox and save it in the globalFlag variable.
* 7. You should have a getFlags function.
* 8. The getFlags function should return "i" if the #i checkbox is checked.
* 9. The getFlags function should return "g" if the #g checkbox is checked.
* 10. The getFlags function should return either "ig" or "gi" if both checkboxes are checked.
* 11. The getFlags function should return an empty string if both checkboxes are not checked.
* 12. When the inner HTML of stringToTest is Gu1n34 P1g5, the value of regexPattern is \d+, and no flag is checked, stringToTest.innerHTML should become Gu<span class="highlight">1</span>n34 P1g5 by clicking the testButton button.
* 13. When the inner HTML of stringToTest is Gu1n34 P1g5, the value of regexPattern is \d+, and the global flag is checked, stringToTest.innerHTML should become Gu<span class="highlight">1</span>n<span class="highlight">34</span> P<span class="highlight">1</span>g<span class="highlight">5</span> by clicking the testButton button.
* 14. When the inner HTML of stringToTest is Gu1n34 P1g5, the value of regexPattern is G, and both the global and case insensitive flags are checked, stringToTest.innerHTML should become <span class="highlight">G</span>u1n34 P1<span class="highlight">g</span>5 by clicking the testButton button.
* 15. When you click the testButton button, if the regex pattern matches the test string, the matched text should be surrounded by a span element with the class of highlight.
* 16. When there's no match, the test string shouldn't be modified.
* 17. When the inner HTML of stringToTest is Gu1n34 P1g5, the value of regexPattern is \d+, and no flag is checked, the inner text of #result should become 1 by clicking the testButton button.
* 18. When the inner HTML of stringToTest is Gu1n34 P1g5, the value of regexPattern is \d+, and the global flag is checked, the inner text of #result should become 1, 34, 1, 5 by clicking the testButton button.
* 19. When there's a match, the matched text should be displayed inside #result.
* 20. When there's no match, the text no match should be displayed inside #result.

index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="styles.css">

    <title>Regex Sandbox</title>

</head>

<body>

    <h1>Regex Sandbox</h1>

    <main>

        <div id="regex-container">

            <label for="pattern">Regex Pattern:

                <div id="pattern-container">/<input type="text" id="pattern" name="pattern"

                        placeholder="Enter your regex pattern">/</div>

            </label>

            <div id="flags-container">

                <p>Flags: </p>

                <label for="i">

                    <input type="checkbox" name="flags" id="i"> i

                </label>

                <label for="g">

                    <input type="checkbox" name="flags" id="g"> g

                </label>

            </div>

        </div>

        <div id="test-container">

            <p>Test String:</p>

            <div id="test-string" placeholder="Enter your test string" contenteditable="true"></div>

        </div>

        <button class="btn" id="test-btn" type="button">Test Regex</button>

        <div id="result-container">

            <h2>Result:</h2>

            <p id="result">

            </p>

        </div>

    </main>

    <script src="script.js"></script>

</body>

</html>

styles.css🡪

\*,

\*::before,

\*::after {

    box-sizing: border-box;

    margin: 0;

    padding: 0;

}

:root {

    --dark-grey: #1b1b32;

    --light-grey: #f5f6f7;

    --golden-yellow: #fecc4c;

    --yellow: #ffcc4c;

    --gold: #feac32;

    --orange: #ffac33;

    --dark-orange: #f89808;

    --border: 0.2rem solid darkgrey;

    --padding: 0.3rem;

}

body {

    background-color: var(--dark-grey);

    color: var(--light-grey);

    font-size: 20px;

    font-family: "Lato", Helvetica, Arial, sans-serif;

    padding: 5px;

}

h1 {

    margin: 5rem auto 2rem;

    text-align: center;

}

p {

    padding: var(--padding);

}

#regex-container {

    max-width: 680px;

    margin: 20px auto;

    display: flex;

    justify-content: center;

    align-items: center;

    border: var(--border);

}

#regex-container>label {

    padding: var(--padding);

    flex: 1 1 auto;

}

#pattern-container {

    display: inline-block;

    color: var(--dark-grey);

    background-color: var(--light-grey);

    margin: 5px;

    border: var(--border);

}

#pattern {

    margin: 0.2rem;

    border: 0;

    font-size: 1rem;

    width: calc(100% - 1.2rem);

}

#pattern:focus {

    outline: none;

}

#flags-container {

    display: flex;

    align-items: center;

    flex: 1 1 auto;

}

#flags-container>label {

    padding: var(--padding);

    margin-right: 0.3rem;

}

#test-container {

    max-width: 680px;

    margin: 20px auto;

    display: flex;

    flex-direction: column;

    flex: 0 0 auto;

    border: var(--border);

}

#test-string {

    background-color: var(--light-grey);

    min-height: 5rem;

    color: var(--dark-grey);

    border-top: var(--border);

    font-size: 1.2rem;

}

[contenteditable=true]:empty:before {

    content: attr(placeholder);

    pointer-events: none;

    color: var(--dark-grey);

}

::placeholder {

    color: var(--dark-grey);

}

button {

    display: block;

    cursor: pointer;

    width: 8rem;

    margin: 0.2rem auto;

    color: var(--dark-grey);

    background-color: var(--gold);

    background-image: linear-gradient(var(--golden-yellow), var(--orange));

    border-color: var(--gold);

    border-width: 0.2rem;

    font-size: 1.1rem;

}

.btn:hover {

    background-image: linear-gradient(var(--yellow), var(--dark-orange));

}

#result-container {

    max-width: 680px;

    margin: 20px auto;

    display: flex;

    justify-content: center;

    align-items: center;

}

h2 {

    align-self: flex-start;

    margin: 0.4rem 0.2rem 0.2rem;

    flex: 0 1 auto;

}

#result {

    color: var(--dark-grey);

    background-color: var(--light-grey);

    font-size: 1.5rem;

    flex: 1 1 auto;

    margin: 0.2rem;

    border: var(--border);

    min-height: 3rem;

}

.highlight {

    background-color: lightgreen;

}

script.js🡪  
  
Build a Customer Complaint Form:

Make an HTML for this page:-  
index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Customer Complaint Form</title>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1>Complaint Form</h1>

    <form id="form">

        <fieldset id="personal-info">

            <div>

                <label for="full-name">Full Name:</label>

                <input type="text" id="full-name" name="full-name" placeholder="John Doe">

            </div>

            <div>

                <label for="email">Email Address:</label>

                <input type="email" id="email" name="email" placeholder="example@domain.com">

            </div>

        </fieldset>

        <hr>

        <fieldset id="product-info">

            <div>

                <label for="order-no">Order No:</label>

                <input type="text" id="order-no" name="order-no" placeholder="2024######">

            </div>

            <div>

                <label for="product-code">Product Code:</label>

                <input type="text" id="product-code" name="product-code" placeholder="XX##-X###-XX#">

            </div>

            <div>

                <label for="quantity">Quantity:</label>

                <input type="number" id="quantity" name="quantity" min="1">

            </div>

        </fieldset>

        <fieldset id="complaints-group">

            <legend>Complaint Reason:</legend>

            <div>

                <input type="checkbox" id="damaged-product" name="complaint" value="damaged-product">

                <label for="damaged-product">Damaged Product</label>

            </div>

            <div>

                <input type="checkbox" id="nonconforming-product" name="complaint" value="nonconforming-product">

                <label for="nonconforming-product">Nonconforming Product</label>

            </div>

            <div>

                <input type="checkbox" id="delayed-dispatch" name="complaint" value="delayed-dispatch">

                <label for="delayed-dispatch">Delayed Dispatch</label>

            </div>

            <div>

                <input type="checkbox" id="other-complaint" name="complaint" value="other">

                <label for="other-complaint">Other</label>

            </div>

        </fieldset>

        <div id="complaint-description-container">

            <legend>Description of Complaint Reason</legend>

            <textarea placeholder="Describe the reason of your complaint in at least 20 characters"

                name="complaint-textarea" id="complaint-description"></textarea>

        </div>

        <fieldset id="solutions-group">

            <legend>Desired Solution</legend>

            <input type="radio" name="solutions" id="refund" value="refund">

            <label for="refund">Refund</label>

            <input type="radio" name="solutions" id="exchange" value="exchange">

            <label for="exchange">Exchange</label>

            <input type="radio" name="solutions" id="other-solution" value="other">

            <label for="other-solution">Other</label>

        </fieldset>

        <div id="solution-description-container">

            <legend>Description of Desired Solution</legend>

            <textarea placeholder="Describe the desired solution to your issue in at least 20 characters"

                name="solution-textarea" id="solution-description"></textarea>

        </div>

        <div id="btn-container">

            <button type="submit" id="submit-btn">Submit</button>

            <span id="message-box" aria-live="polite"></span>

        </div>

    </form>

    <script src="script.js"></script>

</body>

</html>

styles.css🡪

\* {

    box-sizing: border-box;

}

h1 {

    text-align: center;

}

#form {

    max-width: 70%;

    margin: auto;

    border-radius: 10px;

    box-shadow: rgba(0, 0, 0, 0.35) 0px 5px 15px;

    padding: 10px;

}

input {

    border-color: rgb(118, 118, 118);

}

#personal-info input, #product-info input {

    width: 100%;

    margin-bottom: 10px;

}

fieldset {

    margin-bottom: 10px;

    border-radius: 5px;

    border-color: rgb(118, 118, 118);

}

textarea {

    width: 100%;

    border-color: rgb(118, 118, 118);

}

#btn-container {

    display: flex;

    justify-content: space-between;

    align-items: center;

}

#submit-btn, #clear-btn {

    margin: 10px 15px 0;

}

For this lab, you have been provided with all the HTML and CSS. You will use JavaScript to validate the complaint form.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. When the form is submitted, you should ensure that:
   * #full-name is not empty.
   * #email is a valid email address format.
   * #order-no is a sequence of ten numbers starting with 2024.
   * #product-code follows the pattern XX##-X###-XX#, where X represents either a lowercase letter or an uppercase letter and # represents a number.
   * #quantity is a positive integer.
   * at least one checkbox from #complaints-group is checked.
   * #complaint-description contains at least twenty characters if the Other checkbox is checked.
   * a radio button from #solutions-group is selected.
   * #solution-description contains at least twenty characters if the Other radio button is selected.
2. You should have a function named validateForm that returns an object containing the following keys: full-name, email, order-no, product-code, quantity, complaints-group, complaint-description, solutions-group, and solution-description. The value of each key should be true if the corresponding form field is correctly filled and false otherwise.
3. You should have a function named isValid that takes the object returned by validateForm as argument and returns true if every form field is correctly filled and false otherwise.
4. If a change event is triggered on a form field and it has a valid value, you should set its border color to green. In case of checkbox and radio button groups, you should set the border color of the parent fieldset.
5. If a change event is triggered on a form field and it has an invalid value, you should set its border color to red. In case of checkbox and radio button groups, you should set the border color of the parent fieldset.
6. When you try to submit the form you should call isValid to validate the form.
7. When you try to submit the form, if the form has any invalid field, each invalid field should be highlighted by setting the border color of each invalid input, textarea or fieldset (in case of checkbox and radio button groups) to red.

* 1. You should have a function named validateForm.
* 2. validateForm should return an object.
* 3. validateForm()["full-name"] should be false when #full-name is empty, and true otherwise.
* 4. When a change event is triggered on #full-name, you should set its border color to green if it contains a valid value, and red otherwise.
* 5. validateForm()["email"] should be true when #email contains a valid email address, and false otherwise.
* 6. When a change event is triggered on #email, you should set its border color to green if it contains a valid email address, and red otherwise.
* 7. validateForm()["order-no"] should be true when #order-no contains a valid value, and false otherwise.
* 8. When a change event is triggered on #order-no, you should set its border color to green if it contains a valid value, and red otherwise.
* 9. validateForm()["product-code"] should be true when #product-code contains a valid value, and false otherwise.
* 10. When a change event is triggered on #product-code, you should set its border color to green if it contains a valid value, and red otherwise.
* 11. validateForm()["quantity"] should be true when #quantity contains a valid value, and false otherwise.
* 12. When a change event is triggered on #quantity, you should set its border color to green if it contains a valid value, and red otherwise.
* 13. When at least one checkbox from #complaints-group is checked, validateForm()["complaints-group"] should be true.
* 14. When none of the checkboxes from #complaints-group is checked, validateForm()["complaints-group"] should be false.
* 15. Once one checkbox from #complaints-group is checked, you should set #complaints-group's border color to green.
* 16. When all of the checkboxes from #complaints-group are changed to the unchecked state, you should set #complaints-group's border color to red.
* 17. When #other-complaint is checked and #complaint-description contains at least twenty characters, validateForm()["complaint-description"] should be true.
* 18. When #other-complaint is checked and #complaint-description contains less than twenty characters, validateForm()["complaint-description"] should be false.
* 19. When #other-complaint is checked and the value of #complaint-description is changed to a valid value, you should set its border color to green.
* 20. When #other-complaint is checked and the value of #complaint-description is changed to an invalid value, you should set its border color to red.
* 21. When a radio button from #solutions-group is checked, validateForm()["solutions-group"] should be true.
* 22. When none of the radio buttons from #solutions-group is checked, validateForm()["solutions-group"] should be false.
* 23. Once a radio button from #solutions-group is checked, you should set #solutions-group's border color to green.
* 24. When the form is submitted and all of the radio buttons from #solutions-group are in the unchecked state, you should set #solutions-group's border color to red.
* 25. When #other-solution is checked and #solution-description contains at least twenty characters, validateForm()["solution-description"] should be true.
* 26. When #other-solution is checked and #solution-description contains less than twenty characters, validateForm()["solution-description"] should be false.
* 27. When #other-solution is checked and the value of #solution-description is changed to a valid value, you should set its border color to green.
* 28. When #other-solution is checked and the value of #solution-description is changed to an invalid value, you should set its border color to red.
* 29. You should have a function named isValid.
* 30. Your isValid function should take the validateForm() as its argument and return true when all the form fields have been filled correctly.
* 31. Your isValid function should take the validateForm() as its argument and return false when not all the form fields have been filled correctly.
* 32. You should call isValid when you try to submit the form.

Code:

script.js🡪  
// Utility: set border color

function setBorderColor(el, isValid) {

  if (!el) return;

  el.style.borderColor = isValid ? "green" : "red";

}

// Main validation function

function validateForm() {

  const result = {};

  const fullName = document.getElementById("full-name");

  const email = document.getElementById("email");

  const orderNo = document.getElementById("order-no");

  const productCode = document.getElementById("product-code");

  const quantity = document.getElementById("quantity");

  const complaintsGroup = document.getElementById("complaints-group");

  const complaintCheckboxes = complaintsGroup.querySelectorAll("input[type='checkbox']");

  const otherComplaint = document.getElementById("other-complaint");

  const complaintDescription = document.getElementById("complaint-description");

  const solutionsGroup = document.getElementById("solutions-group");

  const solutionRadios = solutionsGroup.querySelectorAll("input[type='radio']");

  const otherSolution = document.getElementById("other-solution");

  const solutionDescription = document.getElementById("solution-description");

  // Validation Rules

  result["full-name"] = fullName.value.trim() !== "";

  result["email"] = /^[^@\s]+@[^@\s]+\.[^@\s]+$/.test(email.value);

  result["order-no"] = /^2024\d{6}$/.test(orderNo.value);

  result["product-code"] = /^[a-zA-Z]{2}\d{2}-[a-zA-Z]\d{3}-[a-zA-Z]{2}\d$/.test(productCode.value);

  result["quantity"] = /^[1-9]\d\*$/.test(quantity.value);

  result["complaints-group"] = Array.from(complaintCheckboxes).some(cb => cb.checked);

  if (otherComplaint.checked) {

    result["complaint-description"] = complaintDescription.value.trim().length >= 20;

  } else {

    result["complaint-description"] = true;

  }

  result["solutions-group"] = Array.from(solutionRadios).some(r => r.checked);

  if (otherSolution.checked) {

    result["solution-description"] = solutionDescription.value.trim().length >= 20;

  } else {

    result["solution-description"] = true;

  }

  return result;

}

// Check if all values are valid

function isValid(validationResult) {

  return Object.values(validationResult).every(Boolean);

}

// Attach event listeners to each field

function setupValidationListeners() {

  const fieldMap = {

    "full-name": document.getElementById("full-name"),

    "email": document.getElementById("email"),

    "order-no": document.getElementById("order-no"),

    "product-code": document.getElementById("product-code"),

    "quantity": document.getElementById("quantity"),

    "complaint-description": document.getElementById("complaint-description"),

    "solution-description": document.getElementById("solution-description"),

  };

  Object.entries(fieldMap).forEach(([key, el]) => {

    el.addEventListener("change", () => {

      const res = validateForm();

      setBorderColor(el, res[key]);

    });

  });

  // Complaints checkboxes

  const complaintsGroup = document.getElementById("complaints-group");

  const complaintCheckboxes = complaintsGroup.querySelectorAll("input[type='checkbox']");

  complaintCheckboxes.forEach(cb => {

    cb.addEventListener("change", () => {

      const res = validateForm();

      setBorderColor(complaintsGroup, res["complaints-group"]);

    });

  });

  // Solutions radios

  const solutionsGroup = document.getElementById("solutions-group");

  const solutionRadios = solutionsGroup.querySelectorAll("input[type='radio']");

  solutionRadios.forEach(r => {

    r.addEventListener("change", () => {

      const res = validateForm();

      setBorderColor(solutionsGroup, res["solutions-group"]);

    });

  });

  // Conditional complaint description

  document.getElementById("complaint-description").addEventListener("change", () => {

    const res = validateForm();

    setBorderColor(

      document.getElementById("complaint-description"),

      res["complaint-description"]

    );

  });

  // Conditional solution description

  document.getElementById("solution-description").addEventListener("change", () => {

    const res = validateForm();

    setBorderColor(

      document.getElementById("solution-description"),

      res["solution-description"]

    );

  });

}

// Form submit handler

document.getElementById("form").addEventListener("submit", function (e) {

  e.preventDefault();

  const res = validateForm();

  // Set all field borders

  setBorderColor(document.getElementById("full-name"), res["full-name"]);

  setBorderColor(document.getElementById("email"), res["email"]);

  setBorderColor(document.getElementById("order-no"), res["order-no"]);

  setBorderColor(document.getElementById("product-code"), res["product-code"]);

  setBorderColor(document.getElementById("quantity"), res["quantity"]);

  setBorderColor(document.getElementById("complaints-group"), res["complaints-group"]);

  if (document.getElementById("other-complaint").checked) {

    setBorderColor(document.getElementById("complaint-description"), res["complaint-description"]);

  }

  setBorderColor(document.getElementById("solutions-group"), res["solutions-group"]);

  if (document.getElementById("other-solution").checked) {

    setBorderColor(document.getElementById("solution-description"), res["solution-description"]);

  }

  // Show message

  const msg = document.getElementById("message-box");

  if (isValid(res)) {

    msg.textContent = "Form submitted successfully!";

    msg.style.color = "green";

  } else {

    msg.textContent = "Please correct the errors and try again.";

    msg.style.color = "red";

  }

});

// Setup listeners on load

setupValidationListeners();

Build a Date Conversion Program:

In this lab, you will code a date conversion program that converts a given date to different formats. For example, the current date Fri Sep 27 2024 16:04:43 GMT+0500 (Pakistan Standard Time) would be formatted in the following 2 ways:

* (MM/DD/YYYY): 9/27/2024.
* (Month Day, Year): September 27, 2024.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a variable named currentDate and assign it the current date and time using the Date object.
2. You should create a variable named currentDateFormat and assign it the string Current Date and Time: [current date]. Replace [current date] with the result of the currentDate variable.
3. You should log the value of currentDateFormat to the console.
4. You should create a function named formatDateMMDDYYYY that takes the date object as a parameter. You can name this parameter whatever you like.
5. Your formatDateMMDDYYYY function should convert the current date to month/day/year format and return the string Formatted Date (MM/DD/YYYY): [month]/[day]/[year].
6. You should create a function named formatDateLong that takes the date object as a parameter.
7. Your formatDateLong function should convert the current date to Month Day, Year format and return the string Formatted Date (Month Day, Year): [formatted date].

**Note:** For the tests to pass, make sure that you use en-US for the locale when formatting the dates.

* 1. You should have a variable named currentDate that holds the current date and time using the Date object.
* 2. You should have a variable named currentDateFormat that holds the current date in the format Current Date and Time: <ddd> <MMM> <dd> <yyyy> <HH>:<mm>:<ss> <TIMEZONE>.
* 3. You should log the value of currentDateFormat to the console.
* 4. You should have a function formatDateMMDDYYYY
* 5. The function formatDateMMDDYYYY should take a single parameter.
* 6. When the date object holds Fri Sep 27 2024 16:16:11 GMT+0500 (Pakistan Standard Time), the function formatDateMMDDYYYY should return Formatted Date (MM/DD/YYYY): 9/27/2024.
* 7. You should have a function formatDateLong
* 8. The function formatDateLong should take a single a parameter.
* 9. When the date object holds Fri Sep 27 2024 16:16:11 GMT+0500 (Pakistan Standard Time), the function formatDateLong should return Formatted Date (Month Day, Year): September 27, 2024.

Code:

// 1. Create a variable holding the current date and time

const currentDate = new Date();

// 2. Create a string with current date and time in the default format

const currentDateFormat = `Current Date and Time: ${currentDate}`;

// 3. Log the formatted date to the console

console.log(currentDateFormat);

// 4. Function to return formatted date as MM/DD/YYYY

function formatDateMMDDYYYY(date) {

  const month = date.getMonth() + 1; // getMonth is 0-indexed

  const day = date.getDate();

  const year = date.getFullYear();

  return `Formatted Date (MM/DD/YYYY): ${month}/${day}/${year}`;

}

// 5. Function to return formatted date as "Month Day, Year"

function formatDateLong(date) {

  const formatted = date.toLocaleDateString('en-US', {

    year: 'numeric',

    month: 'long',

    day: 'numeric'

  });

  return `Formatted Date (Month Day, Year): ${formatted}`;

}

// 6. Test both functions

console.log(formatDateMMDDYYYY(currentDate));

console.log(formatDateLong(currentDate));

Build a Drum Machine:

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

1. You should have a div element with an id of drum-machine that contains all other elements.
2. Inside the #drum-machine element you should have another div with an id of pad-bank.
3. Inside the #drum-machine element you should have a p element with an id of display.
4. Inside your #pad-bank element you should have nine clickable drum pad elements each with a class of drum-pad, a unique id that describes the audio clip the drum pad will be set up to trigger, and an inner text that corresponds to one of the following keys on the keyboard: Q, W, E, A, S, D, Z, X, C. The drum pads MUST be in this order.
5. Each .drum-pad should have an audio element which has a class of clip, a src attribute that points to an audio clip, and an id corresponding to the inner text of its parent .drum-pad element (e.g. id="Q", id="W", id="E" etc.).
6. When you click on a .drum-pad element, the audio clip contained in its child audio element should be triggered.
7. When you press the trigger key associated with each .drum-pad, the audio clip contained in its child audio element should be triggered (e.g. pressing the Q key should trigger the drum pad which contains the string Q, pressing the W key should trigger the drum pad which contains the string W, etc.).
8. When a .drum-pad is triggered, you should display a string describing the associated audio clip as the inner text of the #display element (each string must be unique).

Some audio samples you can use for your drum machine can be found at https://cdn.freecodecamp.org/curriculum/drum/<fileName>, where <fileName> is as follows:

| **drum name** | **fileName** |
| --- | --- |
| Heater 1 | Heater-1.mp3 |
| Heater 2 | Heater-2.mp3 |
| Heater 3 | Heater-3.mp3 |
| Heater 4 | Heater-4\_1.mp3 |
| Clap | Heater-6.mp3 |
| Open-HH | Dsc\_Oh.mp3 |
| Kick-n'-Hat | Kick\_n\_Hat.mp3 |
| Kick | RP4\_KICK\_1.mp3 |
| Closed-HH | Cev\_H2.mp3 |

* 1. You should have a div element with an id of drum-machine that contains all other elements.
* 2. Inside the #drum-machine element you should have another div with an id of pad-bank.
* 3. Inside the #drum-machine element you should have a p element with an id of display.
* 4. Inside your #pad-bank element you should have nine clickable drum pad elements each with a class of drum-pad.
* 5. Each .drum-pad should have one of the following letters as innerText, in order: Q, W, E, A, S, D, Z, X, C.
* 6. Each .drum-pad should have an audio element which has a class of clip, a src attribute that points to an audio clip, and an id corresponding to the inner text of its parent .drum-pad element (e.g. id="Q", id="W", id="E" etc.).
* 7. When you click on a .drum-pad element, the audio clip contained in its child audio element should be triggered.
* 8. When you press one of the keys Q, W, E, A, S, D, Z, X, C on your keyboard, the corresponding audio element should play the corresponding sound.
* 9. When a .drum-pad is triggered, you should display a string describing the associated audio clip as the inner text of the #display element (each string must be unique).

index.html🡪  
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <title>Drum Machine</title>

  <link rel="stylesheet" href="style.css" />

</head>

<body>

  <div id="drum-machine">

    <p id="display">Play a Sound</p>

    <div id="pad-bank">

      <!-- Drum pads are created dynamically -->

    </div>

  </div>

  <script src="script.js"></script>

</body>

</html>

styles.css🡪

#drum-machine {

  max-width: 400px;

  margin: 50px auto;

  padding: 20px;

  background-color: #222;

  color: #fff;

  font-family: Arial, sans-serif;

  border-radius: 10px;

  text-align: center;

}

#display {

  margin: 20px 0;

  padding: 10px;

  background-color: #444;

  border-radius: 5px;

}

#pad-bank {

  display: grid;

  grid-template-columns: repeat(3, 1fr);

  gap: 10px;

}

.drum-pad {

  background-color: #555;

  border: none;

  padding: 30px;

  border-radius: 5px;

  cursor: pointer;

  font-size: 24px;

  color: white;

  transition: background-color 0.2s ease;

}

.drum-pad:hover {

  background-color: #888;

}

script.js🡪

const pads = [

  { key: 'Q', sound: 'Heater 1', src: 'https://cdn.freecodecamp.org/curriculum/drum/Heater-1.mp3' },

  { key: 'W', sound: 'Heater 2', src: 'https://cdn.freecodecamp.org/curriculum/drum/Heater-2.mp3' },

  { key: 'E', sound: 'Heater 3', src: 'https://cdn.freecodecamp.org/curriculum/drum/Heater-3.mp3' },

  { key: 'A', sound: 'Heater 4', src: 'https://cdn.freecodecamp.org/curriculum/drum/Heater-4\_1.mp3' },

  { key: 'S', sound: 'Clap', src: 'https://cdn.freecodecamp.org/curriculum/drum/Heater-6.mp3' },

  { key: 'D', sound: 'Open-HH', src: 'https://cdn.freecodecamp.org/curriculum/drum/Dsc\_Oh.mp3' },

  { key: 'Z', sound: "Kick-n'-Hat", src: 'https://cdn.freecodecamp.org/curriculum/drum/Kick\_n\_Hat.mp3' },

  { key: 'X', sound: 'Kick', src: 'https://cdn.freecodecamp.org/curriculum/drum/RP4\_KICK\_1.mp3' },

  { key: 'C', sound: 'Closed-HH', src: 'https://cdn.freecodecamp.org/curriculum/drum/Cev\_H2.mp3' }

];

const padBank = document.getElementById('pad-bank');

const display = document.getElementById('display');

// Create drum pads

pads.forEach(pad => {

  const button = document.createElement('button');

  button.classList.add('drum-pad');

  button.id = pad.sound;

  button.innerText = pad.key;

  button.onclick = () => playSound(pad.key);

  const audio = document.createElement('audio');

  audio.classList.add('clip');

  audio.src = pad.src;

  audio.id = pad.key;

  button.appendChild(audio);

  padBank.appendChild(button);

});

// Play sound

function playSound(key) {

  const audio = document.getElementById(key);

  if (audio) {

    audio.currentTime = 0;

    audio.play();

    const pad = pads.find(p => p.key === key);

    display.innerText = pad.sound;

  }

}

// Handle keyboard press

document.addEventListener('keydown', (e) => {

  const key = e.key.toUpperCase();

  if (pads.some(p => p.key === key)) {

    playSound(key);

  }

});

Build a Voting System:

In this lab, you will build a voting system that uses Map to create a poll and Set to prevent duplicate voting.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should initialize a poll variable to a new Map object.
2. You should have a function addOption that accepts a parameter option.
3. In the addOption function:
   * If the option does not already exist in the poll, it should be added to the poll with an empty Set as its value to track voters. You should also return the message Option "<option>" added to the poll.
   * If the option already exists, it should return the message Option "<option>" already exists..
   * If you try to add an empty option, the function should return the message Option cannot be empty..
4. You should have a function vote that accepts two parameters, option (the option to vote for) and voterId (a unique ID for the voter).
5. In the vote function:
   * If the option does not exist in the poll, the function should return the message Option "<option>" does not exist..
   * If the option exists, the function should check if the voterId has already voted for this option.
   * If the voter has already voted, the function should display the message Voter <voterId> has already voted for "<option>".
   * If the voter has not voted, voterId should be added to the Set of voters for this option. The function should return the message Voter <voterId> voted for "<option>".
6. You should have at least three options in your poll.
7. Your poll should have at least three votes.
8. You should have a function displayResults that returns the poll results in the following format:

* 1. You should have a poll variable initialized to a new Map object.
* 2. You should define a function addOption that accepts a parameter option.
* 3. You should define a function vote that accepts two parameters, option and voterId.
* 4. You should define a function displayResults to display the poll results.
* 5. You should have at least three options in your poll.
* 6. Your poll should have at least three votes.
* 7. You should ensure each voting option maps to a Set object.
* 8. addOption("Egypt") should return Option "Egypt" added to the poll.
* 9. Adding an empty option should return "Option cannot be empty."
* 10. When Turkey is already added, addOption("Turkey") should return Option "Turkey" already exists.
* 11. When Malaysia exists in the voting options, vote("Malaysia", "traveler1") should return Voter traveler1 voted for "Malaysia".
* 12. vote should update the Set of voters for an option.
* 13. When traveler1 tries to vote for Algeria again, vote("Algeria", "traveler1") should return Voter traveler1 has already voted for "Algeria".
* 14. Duplicate votes should not increase the size of the Set.
* 15. When Nigeria is not in the voting options, vote("Nigeria", "traveler2") should return Option "Nigeria" does not exist.
* 16. A unique option should be able to receive multiple votes.
* 17. displayResults() should return the results in the correct format.

Example code:

**Poll Results:  
OptionA: N votes**

**OptionB: N votes**

**.**

**.**

**/\***

**sample output**

**Poll Results:**

**Turkey: 2 votes**

**Morocco: 1 votes**

**\*/**

Code:

// 1. Initialize poll as a Map

const poll = new Map();

// 2. addOption function

function addOption(option) {

  if (!option || option.trim() === "") {

    return "Option cannot be empty.";

  }

  if (poll.has(option)) {

    return `Option "${option}" already exists.`;

  }

  poll.set(option, new Set());

  return `Option "${option}" added to the poll.`;

}

// 3. vote function

function vote(option, voterId) {

  if (!poll.has(option)) {

    return `Option "${option}" does not exist.`;

  }

  const voters = poll.get(option);

  if (voters.has(voterId)) {

    return `Voter ${voterId} has already voted for "${option}".`;

  }

  voters.add(voterId);

  return `Voter ${voterId} voted for "${option}".`;

}

// 4. displayResults function

function displayResults() {

  let results = "Poll Results:\n";

  for (const [option, voters] of poll.entries()) {

    results += `${option}: ${voters.size} votes\n`;

  }

  return results.trim(); // removes trailing newline

}

// Add at least 3 options

console.log(addOption("Turkey"));     // Option "Turkey" added to the poll.

console.log(addOption("Malaysia"));   // Option "Malaysia" added to the poll.

console.log(addOption("Algeria"));    // Option "Algeria" added to the poll.

// Add duplicate option

console.log(addOption("Turkey"));     // Option "Turkey" already exists.

// Add empty option

console.log(addOption(""));           // Option cannot be empty.

// Cast at least 3 votes

console.log(vote("Malaysia", "traveler1"));  // Voter traveler1 voted for "Malaysia"

console.log(vote("Algeria", "traveler1"));   // Voter traveler1 voted for "Algeria"

console.log(vote("Algeria", "traveler1"));   // Voter traveler1 has already voted for "Algeria"

console.log(vote("Malaysia", "traveler2"));  // Voter traveler2 voted for "Malaysia"

console.log(vote("Nigeria", "traveler3"));   // Option "Nigeria" does not exist.

console.log(vote("Malaysia", "traveler3"));  // Voter traveler3 voted for "Malaysia"

// Display final results

console.log("\nPoll Results:");

console.log(displayResults());

Build a Bookmark Manager App:

For this lab, all the HTML and CSS has been provided to you. You will use JavaScript to complete the Bookmark Manager app so that it can store a collection of bookmarks in the local storage and read data from it.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a getBookmarks function that returns the bookmarks array stored in the local storage. If the bookmarks key has not been set yet, or it doesn't contain a valid array with bookmarks, the getBookmarks function should return an empty array.
2. The bookmarks key stored in the local storage should be an array of bookmark objects. Each bookmark object should have three keys: name, category, and url.
3. You should have a function named displayOrCloseForm that toggles the hidden class on #main-section and #form-section.
4. When you click #add-bookmark-button, you should update the inner text of .category-name to be the value of the selected option from #category-dropdown and call displayOrCloseForm to display the form section and hide the main section.
5. When you click #close-form-button, you should run your function to hide the form section and display the main section.
6. When you click #add-bookmark-button-form, you should update the bookmarks key stored in the local storage by adding a bookmark object to the end of the array. The object should have name set to the value of the #name input, category set to the value of the selected option from the category dropdown, and url set to the value of the #url input.
7. Once the bookmarks key is updated, you should reset the value of #name and #url to an empty string before running your function to hide the form section and show the main section.
8. You should have a function named displayOrHideCategory that toggles the hidden class on #main-section and #bookmark-list-section.
9. When you click #view-category-button, you should update the inner text of .category-name to be the value of the selected option from #category-dropdown, modify the inner HTML of #category-list according to the user stories below, and call the displayOrHideCategory function.
10. If none of the bookmarks in local storage have the category, you should set the inner HTML of the #category-list to a p element with the text No Bookmarks Found.
11. If one or more bookmarks in local storage have the selected category, add a radio button for each bookmark to the innerHTML of the #category-list. Give each radio button an id and value set to the bookmark name, and a name that's the same for all the buttons.
12. Each radio button should have a corresponding label containing an anchor element with the bookmark name and the href attribute set to the bookmark URL.
13. When you click the #close-list-button, you should run your function to hide the #bookmark-list-section and display the main section.
14. When you click the #delete-bookmark-button, you should delete the bookmark corresponding to the selected radio button and appropriate category from the local storage and update the displayed bookmark list.

* 1. You should have a getBookmarks function.
* 2. Your getBookmarks function should return an array.
* 3. Your getBookmarks function should return the bookmarks key from localStorage.
* 4. When the bookmarks key is not set in localStorage or is an empty array, the getBookmarks function should return an empty array.
* 5. When the bookmarks key in the localStorage does not contain a valid array of bookmark objects, the getBookmarks function should return an empty array.
* 6. You should have a function named displayOrCloseForm.
* 7. Your displayOrCloseForm function should toggle the hidden class on #main-section and #form-section.
* 8. When you click #add-bookmark-button, you should update the inner text of .category-name to be the value of the selected option from #category-dropdown.
* 9. When you click #add-bookmark-button, you should call displayOrCloseForm to display the form section and hide the main section.
* 10. When you click #close-form-button, you should call displayOrCloseForm to hide the form section and display the main section.
* 11. When you click #add-bookmark-button-form, you should update the bookmarks key stored in the local storage by adding an object to the end of the array. The added object should have name set to the value of the #name input, category set to the value of the selected option from the category dropdown, and url set to the value of the #url input.
* 12. When you click #add-bookmark-button-form, you should reset the value of #name and #url to an empty string.
* 13. When you click #add-bookmark-button-form, you should run displayOrCloseForm to hide the form section and show the main section.
* 14. You should have a function named displayOrHideCategory.
* 15. Your displayOrHideCategory function should toggle the hidden class on #main-section and #bookmark-list-section.
* 16. When you click #view-category-button, you should update the inner text of .category-name to be the value of the selected option from #category-dropdown.
* 17. When you click #view-category-button, you should add a p element with the text No Bookmarks Found to #category-list's inner HTML if none of the bookmarks in local storage have the selected category.
* 18. When you click the #view-category-button, you should modify the #category-list element's inner HTML by adding a radio button. The radio button should have the id and value attributes set to the bookmark name for each bookmark in the selected category. Additionally, each radio button should have the same name attribute.
* 19. Each radio button added to #category-list's inner HTML should have a corresponding label containing an anchor element with the bookmark name and the href attribute set to the bookmark URL.
* 20. Each label element should contain an anchor element with the bookmark name as text, and the href attribute set to the bookmark URL.
* 21. When you click #close-list-button, you should hide #bookmark-list-section and display the main section.
* 22. When you click the #close-list-button and then open any category, the #category-list should contain only data relevant for the selected category, without duplicating entries.
* 23. When you click the #delete-bookmark-button, you should delete the bookmark corresponding to the selected radio button and appropriate category from the local storage and update the displayed bookmark list.

index.html🡪

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=0.0">

    <title>Bookmark Manager</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <main>

        <section id="main-section">

            <h1>Bookmark Manager</h1>

            <div id="dropdown">

                <label for="category-dropdown">Select a category:</label>

                <select id="category-dropdown" name="options">

                    <option value="news" selected>News</option>

                    <option value="entertainment">Entertainment</option>

                    <option value="work">Work</option>

                    <option value="miscellaneous">Miscellaneous</option>

                </select>

            </div>

            <div id="buttons">

                <button type="button" id="view-category-button">View Category</button>

                <button type="button" id="add-bookmark-button">Add Bookmark</button>

            </div>

        </section>

        <section id="form-section" class="hidden">

            <form>

                <h2 class="category-name"></h2>

                <div>

                    <label for="name">Name:</label>

                    <input type="text" id="name">

                </div>

                <div>

                    <label for="url">URL:</label>

                    <input type="text" id="url">

                </div>

                <div>

                    <button type="button" id="close-form-button">Go Back</button>

                    <button type="button" id="add-bookmark-button-form">Add Bookmark</button>

                </div>

            </form>

        </section>

        <section id="bookmark-list-section" class="hidden">

            <h2 class="category-name"></h2>

            <div id="category-list">

            </div>

            <div>

                <button type="button" id="close-list-button">Go Back</button>

                <button type="button" id="delete-bookmark-button">Delete Bookmark</button>

            </div>

        </section>

    </main>

    <script src="script.js"></script>

</body>

</html>

styles.css🡪

:root {

  --light-grey: #f5f6f7;

  --dark-grey: #0a0a23;

  --yellow: #f1be32;

  --golden-yellow: #feac32;

}

\*,

\*::before,

\*::after {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

main {

  display: flex;

  justify-content: center;

}

body {

  background-color: var(--dark-grey);

}

.hidden {

  display: none;

}

section {

  display: flex;

  flex-direction: column;

  justify-content: center;

}

select,

input,

label {

  margin-left: 10px;

}

div {

  padding: 30px;

  display: flex;

  justify-content: center;

}

.close-form-button {

  background: none;

  border: none;

  cursor: pointer;

}

h1, h2 {

  margin-top: 20px;

  text-align: center;

}

#category-list {

  text-align: center;

  display: flex;

  flex-direction: column;

  justify-content: flex-start;

  background-color: var(--light-grey);

  align-self: center;

  width: 80%;

  margin-top: 15px;

  border-radius: 10px;

}

#category-list,

h1,

h2,

label {

  color: var(--light-grey);

}

#category-list p {

  color: var(--dark-grey);

}

button {

  cursor: pointer;

  padding: 5px;

  width: 100px;

  margin: 10px;

  color: var(--dark-grey);

  background-color: var(--golden-yellow);

  background-image: linear-gradient(#fecc4c, #ffac33);

  border-color: var(--golden-yellow);

  border-width: 3px;

}

button:hover {

  background-image: linear-gradient(#ffcc4c, #f89808);

}

section {

  margin-top: 60px;

  border: 2px solid var(--golden-yellow);

  width: fit-content;

  border-radius: 10px;

}

script.js🡪  
// 1. getBookmarks function

function getBookmarks() {

  try {

    const bookmarksJSON = localStorage.getItem("bookmarks");

    const bookmarks = JSON.parse(bookmarksJSON);

    if (

      Array.isArray(bookmarks) &&

      bookmarks.every(b =>

        b &&

        typeof b === 'object' &&

        typeof b.name === 'string' &&

        typeof b.category === 'string' &&

        typeof b.url === 'string'

      )

    ) {

      return bookmarks;

    } else {

      return [];

    }

  } catch (error) {

    return [];

  }

}

// 6. displayOrCloseForm toggles form section and main section

function displayOrCloseForm() {

  document.getElementById("main-section").classList.toggle("hidden");

  document.getElementById("form-section").classList.toggle("hidden");

}

// 14. displayOrHideCategory toggles list section and main section

function displayOrHideCategory() {

  document.getElementById("main-section").classList.toggle("hidden");

  document.getElementById("bookmark-list-section").classList.toggle("hidden");

}

// Event: Add Bookmark Button (open form)

document.getElementById("add-bookmark-button").addEventListener("click", () => {

  const category = document.getElementById("category-dropdown").value;

  document.querySelectorAll(".category-name").forEach(el => el.innerText = category);

  displayOrCloseForm();

});

// Event: Close Form Button (go back to main)

document.getElementById("close-form-button").addEventListener("click", () => {

  displayOrCloseForm();

});

// Event: Add Bookmark from Form

document.getElementById("add-bookmark-button-form").addEventListener("click", () => {

  const name = document.getElementById("name").value.trim();

  const url = document.getElementById("url").value.trim();

  const category = document.getElementById("category-dropdown").value;

  if (!name || !url) return;

  const bookmarks = getBookmarks();

  bookmarks.push({ name, category, url });

  localStorage.setItem("bookmarks", JSON.stringify(bookmarks));

  document.getElementById("name").value = "";

  document.getElementById("url").value = "";

  displayOrCloseForm();

});

// Event: View Category

document.getElementById("view-category-button").addEventListener("click", () => {

  const category = document.getElementById("category-dropdown").value;

  document.querySelectorAll(".category-name").forEach(el => el.innerText = category);

  const bookmarks = getBookmarks();

  const categoryBookmarks = bookmarks.filter(b => b.category === category);

  const list = document.getElementById("category-list");

  if (categoryBookmarks.length === 0) {

    list.innerHTML = `<p>No Bookmarks Found</p>`;

  } else {

    list.innerHTML = categoryBookmarks

      .map(b =>

        `<div>

          <input type="radio" id="${b.name}" name="bookmark" value="${b.name}">

          <label for="${b.name}">

            <a href="${b.url}" target="\_blank">${b.name}</a>

          </label>

        </div>`

      ).join("");

  }

  displayOrHideCategory();

});

// Event: Close List Button

document.getElementById("close-list-button").addEventListener("click", () => {

  displayOrHideCategory();

  document.getElementById("category-list").innerHTML = "";

});

// Event: Delete Bookmark

document.getElementById("delete-bookmark-button").addEventListener("click", () => {

  const selectedRadio = document.querySelector('input[name="bookmark"]:checked');

  const category = document.querySelector(".category-name").innerText;

  if (!selectedRadio) return;

  const nameToDelete = selectedRadio.value;

  let bookmarks = getBookmarks();

  bookmarks = bookmarks.filter(b => !(b.name === nameToDelete && b.category === category));

  localStorage.setItem("bookmarks", JSON.stringify(bookmarks));

  // Re-render the list

  const categoryBookmarks = bookmarks.filter(b => b.category === category);

  const list = document.getElementById("category-list");

  if (categoryBookmarks.length === 0) {

    list.innerHTML = `<p>No Bookmarks Found</p>`;

  } else {

    list.innerHTML = categoryBookmarks

      .map(b =>

        `<div>

          <input type="radio" id="${b.name}" name="bookmark" value="${b.name}">

          <label for="${b.name}">

            <a href="${b.url}" target="\_blank">${b.name}</a>

          </label>

        </div>`

      ).join("");

  }

});

Build a Project Idea Board:

**User Stories:**

1. You should define an object constant named projectStatus with the three keys: PENDING, SUCCESS, and FAILURE. Each status should be assigned an object with a description key with the value Pending Execution, Executed Successfully, and Execution Failed, respectively.
2. You should define a class named ProjectIdea with a constructor that takes title and description as parameters. Initialize the title and description properties with the provided parameters. The class should also have a property named status that is set to the value projectStatus.PENDING by default.
3. You should define a method named updateProjectStatus inside the ProjectIdea class. This method should accept a newStatus parameter and update the status property to the given value.
4. You should define a ProjectIdeaBoard class with a constructor that accepts a title and initializes an empty array named ideas to hold instances of the ProjectIdea class.
5. You should define a method named pin inside the ProjectIdeaBoard class that accepts an instance of the ProjectIdea class and pushes the given instance to the ideas array.
6. You should define a method named unpin inside the ProjectIdeaBoard class. This method should accept an instance of the ProjectIdea class and removes it from the ideas array.
7. You should define a method named count that returns the number of project ideas in the given ProjectIdeaBoard array.
8. You should define a method named formatToString that returns the name of the projects ideas, their description and status in the format:

Example code:

**<ProjectIdeaBoard title> has <ProjectIdeaBoard count> idea(s)**

**<ProjectIdea title> (<ProjectIdea status description>) - <ProjectIdea description>**

**<ProjectIdea title> (<ProjectIdea status description>) - <ProjectIdea description>**

* 1. You should define an object constant named projectStatus with the correct project statuses and descriptions.
* 2. You should have a ProjectIdea class.
* 3. Your ProjectIdea class should initialize the title property and description property based on the parameters passed. It should also set the status to projectStatus.PENDING.
* 4. Your ProjectIdea class should have a updateProjectStatus method.
* 5. You should have a ProjectIdeaBoard class.
* 6. Your ProjectIdeaBoard should initialize the title property based on the parameter passed and initialize an empty array named ideas to hold instances of the ProjectIdea class.
* 7. Your ProjectIdeaBoard class should have a pin method.
* 8. Your ProjectIdeaBoard class should have an unpin method.
* 9. Your ProjectIdeaBoard class should have a count method.
* 10. Your ProjectIdeaBoard class should have a formatToString method.
* 11. new ProjectIdea("Smart Window Locks", "An automation project allowing users to lock, unlock windows automatically based on weather conditions.") should return { title: 'Smart Window Locks', description: 'An automation project allowing users to lock, unlock windows automatically based on weather conditions.', status: { description: 'Pending Execution' } }.
* 12. Calling updateProjectStatus(projectStatus.SUCCESS) on new ProjectIdea("Fitness Tracker App", "An app that tracks user workouts, diet, and sleep patterns.") should update the status to { description: 'Executed Successfully' }.
* 13. Calling updateProjectStatus(projectStatus.FAILURE) on new ProjectIdea("Breakfast Chef Robot", "A robot that can follow a given list of instructions and prepare breakfast for the user and let them know through their phone.") should update the status to { description: 'Execution Failed' }.
* 14. Calling updateProjectStatus(projectStatus.SUCCESS) on new ProjectIdea("Online Used Video Games Store", "An online platform where users can buy second hand physical copies of video games from other users.") should update the status to { description: 'Executed Successfully' }.
* 15. You should be able to pin a ProjectIdea object to your ProjectIdeaBoard using the pin method.
* 16. You should be able to unpin a ProjectIdea object to your ProjectIdeaBoard using the unpin method.
* 17. When new ProjectIdeaBoard("Empty Board") is empty, emptyBoard.formatToString() should return Empty Board has 0 idea(s)\n.
* 18. When you pin new ProjectIdea("Smart Home System", "An integrated system to control lighting, temperature, and security devices remotely.") to new ProjectIdeaBoard("Tech Projects Board"), techProjects.formatToString() should return Tech Projects Board has 1 idea(s)\nSmart Home System (Pending Execution) - An integrated system to control lighting, temperature, and security devices remotely.\n.

**Code:**

// 1. Define the projectStatus constant

const projectStatus = {

  PENDING: { description: "Pending Execution" },

  SUCCESS: { description: "Executed Successfully" },

  FAILURE: { description: "Execution Failed" }

};

// 2. Define the ProjectIdea class

class ProjectIdea {

  constructor(title, description) {

    this.title = title;

    this.description = description;

    this.status = projectStatus.PENDING; // default status

  }

  // 3. Method to update project status

  updateProjectStatus(newStatus) {

    this.status = newStatus;

  }

}

// 4. Define the ProjectIdeaBoard class

class ProjectIdeaBoard {

  constructor(title) {

    this.title = title;

    this.ideas = [];

  }

  // 5. Method to pin a ProjectIdea instance

  pin(projectIdea) {

    this.ideas.push(projectIdea);

  }

  // 6. Method to unpin a ProjectIdea instance

  unpin(projectIdea) {

    const index = this.ideas.indexOf(projectIdea);

    if (index !== -1) {

      this.ideas.splice(index, 1);

    }

  }

  // 7. Method to count the ideas

  count() {

    return this.ideas.length;

  }

  // 8. Method to format the board details as a string

  formatToString() {

    let result = `${this.title} has ${this.count()} idea(s)\n`;

    this.ideas.forEach(idea => {

      result += `${idea.title} (${idea.status.description}) - ${idea.description}\n`;

    });

    return result;

  }

}

const idea1 = new ProjectIdea("Smart Window Locks", "An automation project allowing users to lock, unlock windows automatically based on weather conditions.");

const idea2 = new ProjectIdea("Fitness Tracker App", "An app that tracks user workouts, diet, and sleep patterns.");

idea2.updateProjectStatus(projectStatus.SUCCESS);

const idea3 = new ProjectIdea("Breakfast Chef Robot", "A robot that can follow a given list of instructions and prepare breakfast for the user and let them know through their phone.");

idea3.updateProjectStatus(projectStatus.FAILURE);

const board = new ProjectIdeaBoard("Tech Projects Board");

board.pin(idea1);

board.pin(idea2);

board.pin(idea3);

console.log(board.formatToString());

board.unpin(idea2);

console.log(board.formatToString());

const emptyBoard = new ProjectIdeaBoard("Empty Board");

console.log(emptyBoard.formatToString());

**Build a Bank Account Management Program:**

**User Stories:**

1. You should define a class named BankAccount with a constructor. The constructor should set the default balance to 0 and initialize an empty array named transactions to store transaction records as objects.
2. Each transaction stored in the transactions array should be an object with two properties: type and amount. The type property should be either deposit or withdraw, and the amount property should be the amount deposited or withdrawn.
3. You should define a method named deposit that takes the deposit amount as a parameter. When the deposit amount is greater than 0, it should:
   * Push a new object to the transactions array with a type of deposit and the amount deposited.
   * Update the balance.
   * Return "Successfully deposited $[amount]. New balance: $[balance]".
4. If the amount is less than or equal to 0, the deposit method should return "Deposit amount must be greater than zero.".
5. You should define a method named withdraw that takes an amount as a parameter. This method should update the current balance according to withdrawals. When the amount to be withdrawn is greater than 0 and less than or equal to the current balance, it should:
   * Push a new object to the transactions array with a type of withdraw and the amount withdrawn.
   * Update the balance.
   * Return "Successfully withdrew $[amount]. New balance: $[balance]".
6. If the amount to be withdrawn is less than or equal to 0 or greater than the current balance, the withdraw method should return "Insufficient balance or invalid amount.".
7. You should define a method named checkBalance that returns the current balance in the format "Current balance: $[balance]".
8. You should define a method named listAllDeposits that iterates through the transactions array and returns all deposits in the format "Deposits: amount,amount,...".
9. You should define a method named listAllWithdrawals that iterates through the transactions array and returns all withdrawals in the format "Withdrawals: amount,amount,...".
10. You should create a new instance of BankAccount named myAccount.
11. Your myAccount bank account should have at least five transactions.
12. Your myAccount bank account should have at least two deposits.
13. Your myAccount bank account should have at least two withdrawals.
14. Your myAccount bank account should have a balance greater than $100.

* 1. You should define a class named BankAccount.
* 2. The BankAccount object should initially have a balance of 0 and an empty array transactions to store transaction records.
* 3. You should have a deposit method that takes the deposit amount as a parameter.
* 4. You should have a withdraw method that takes the withdrawal amount as a parameter.
* 5. You should have a checkBalance method that checks the current balance.
* 6. You should have a listAllDeposits method that lists all deposits.
* 7. You should have a listAllWithdrawals method that lists all withdrawals.
* 8. BankAccount.deposit(100) should return "Successfully deposited $100. New balance: $100".
* 9. BankAccount.deposit(-50) should return "Deposit amount must be greater than zero.".
* 10. BankAccount.deposit(0) should return "Deposit amount must be greater than zero.".
* 11. When the account balance is 100, BankAccount.withdraw(150) should return "Insufficient balance or invalid amount.".
* 12. BankAccount.withdraw(-50) should return "Insufficient balance or invalid amount.".
* 13. BankAccount.withdraw(0) should return "Insufficient balance or invalid amount.".
* 14. When the account balance is 200, BankAccount.withdraw(150) should return "Successfully withdrew $150. New balance: $50".
* 15. When the account balance is 200, BankAccount.checkBalance() should return "Current balance: $200".
* 16. When you deposit 10, 35, 90, the listAllDeposits method should return "Deposits: 10,35,90".
* 17. When you withdraw 20, 50, 100, the listAllWithdrawals method should return "Withdrawals: 20,50,100".
* 18. You should have an instance of BankAccount named myAccount.
* 19. Your myAccount bank account should have at least five transactions.
* 20. Your myAccount bank account should have at least two deposits.
* 21. Your myAccount bank account should have at least two withdrawals.
* 22. Your myAccount bank account should have a balance greater than $100.

Code:

class BankAccount {

  constructor() {

    this.balance = 0;

    this.transactions = [];

  }

  deposit(amount) {

    if (amount > 0) {

      this.transactions.push({ type: 'deposit', amount: amount });

      this.balance += amount;

      return `Successfully deposited $${amount}. New balance: $${this.balance}`;

    } else {

      return "Deposit amount must be greater than zero.";

    }

  }

  withdraw(amount) {

    if (amount > 0 && amount <= this.balance) {

      this.transactions.push({ type: 'withdraw', amount: amount });

      this.balance -= amount;

      return `Successfully withdrew $${amount}. New balance: $${this.balance}`;

    } else {

      return "Insufficient balance or invalid amount.";

    }

  }

  checkBalance() {

    return `Current balance: $${this.balance}`;

  }

  listAllDeposits() {

    const deposits = this.transactions

      .filter(tx => tx.type === 'deposit')

      .map(tx => tx.amount);

    return `Deposits: ${deposits.join(",")}`;

  }

  listAllWithdrawals() {

    const withdrawals = this.transactions

      .filter(tx => tx.type === 'withdraw')

      .map(tx => tx.amount);

    return `Withdrawals: ${withdrawals.join(",")}`;

  }

}

// Create instance of BankAccount

const myAccount = new BankAccount();

// Perform at least 5 transactions

console.log(myAccount.deposit(200));   // Deposit #1

console.log(myAccount.deposit(150));   // Deposit #2

console.log(myAccount.withdraw(50));   // Withdraw #1

console.log(myAccount.withdraw(75));   // Withdraw #2

console.log(myAccount.deposit(30));    // Deposit #3

// Checking balance (should be > 100)

console.log(myAccount.checkBalance());

// List deposits and withdrawals

console.log(myAccount.listAllDeposits());    // Should show all deposit amounts

console.log(myAccount.listAllWithdrawals()); // Should show all withdrawal amounts

Build a Permutation Generator:

In this lab, you will build a permutation generator that will take a string and return all possible permutations of the characters in the string. For example, the possible permutations of the string cat are cat, cta, act, atc, tac, and tca.

**Objective:** Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should create a function named permuteString.
2. The permuteString function should take three parameters- a string, a prefix value and an empty array for storing and returning the results. The prefix value would accumulate characters to form a permutation.
3. Inside the function, you should check if the length of the passed string is 0. If it is, push the current prefix to the results and return the results.
4. Iterate over each character in the input string and for each iteration, remove the current character from the string and call the permuteString function recursively with updated arguments to build the remaining permutations.
5. You should return the final results array.
6. You should ensure that the permutations are unique by removing duplicates.

* 1. You should have a function permuteString.
* 2. You should use recursion in your permuteString function.
* 3. permuteString("far") should return [ "far", "fra", "afr", "arf", "rfa", "raf" ].
* 4. permuteString("fcc") should return [ "fcc", "cfc", "ccf" ].
* 5. permuteString("p") should return [ "p" ].
* 6. permuteString("") should return [""].
* 7. permuteString("walk") should return ["walk", "wakl", "wlak", "wlka", "wkla", "wkal", "awlk", "awkl", "alwk", "alkw", "aklw", "akwl", "lawk", "lakw", "lwak", "lwka", "lkaw", "lkwa", "kawl", "kalw", "kwal", "kwla", "klaw", "klwa"]. `.
* 8. permuteString should return the correct results.

Code:

function permuteString(str, prefix = "", results = []) {

  if (str.length === 0) {

    if (!results.includes(prefix)) {

      results.push(prefix);

    }

    return results;

  }

  for (let i = 0; i < str.length; i++) {

    const rem = str.slice(0, i) + str.slice(i + 1); // remove current char

    permuteString(rem, prefix + str[i], results);   // recursive call

  }

  return results;

}

console.log(permuteString("cat"));

// ["cat", "cta", "act", "atc", "tca", "tac"]

console.log(permuteString("fcc"));

// ["fcc", "cfc", "ccf"]

console.log(permuteString("walk"));

// 24 permutations

Build a recipe-ingredient converter:

index.html🡪  
  
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Recipe Ingredient Converter</title>

    <link rel="stylesheet" href="styles.css" />

  </head>

  <body>

    <main class="container">

      <h1>Recipe Ingredient Converter</h1>

      <form id="recipe-form">

        <div class="input-container">

          <label for="ingredient">Ingredient:</label>

          <input type="text" id="ingredient" required />

        </div>

        <div class="input-container">

          <label for="quantity">Quantity:</label>

          <input type="number" id="quantity" step="any" required />

        </div>

        <div class="input-container">

          <label for="unit">Unit:</label>

          <select id="unit">

            <option value="cup">Cup</option>

            <option value="gram">Gram</option>

            <option value="ounce">Ounce</option>

            <option value="teaspoon">Teaspoon</option>

          </select>

        </div>

        <div class="input-container">

          <label for="servings">Number of Servings:</label>

          <input type="number" id="servings" step="any" value="1" />

        </div>

        <button type="submit">Convert</button>

      </form>

      <section>

        <h2>Converted Ingredients</h2>

        <ul id="result-list"></ul>

      </section>

    </main>

    <script src="script.js"></script>

  </body>

</html>

styles.css🡪  
\*,

\*::before,

\*::after {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

:root {

  --dark-grey: #0a0a23;

  --white: #fff;

  --light-grey: #ddd;

  --golden-yellow: #fecc4c;

  --yellow: #ffcc4c;

  --gold: #feac32;

  --orange: #ffac33;

  --dark-orange: #f89808;

}

body {

  font-family: Arial, sans-serif;

  background-color: var(--dark-grey);

  display: flex;

  justify-content: center;

  align-items: center;

  height: 100vh;

}

h1 {

  margin: 10px 0 15px;

}

.container {

  text-align: center;

  background: var(--white);

  padding: 20px;

  border-radius: 8px;

  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

  width: 100%;

  max-width: 600px;

}

input,

select {

  padding: 10px;

  font-size: 16px;

  margin: 5px 0;

  border: 1px solid var(--light-grey);

  border-radius: 4px;

}

button {

  font-size: 1.2rem;

  cursor: pointer;

  width: 200px;

  margin: 20px 0 30px;

  color: var(--dark-grey);

  background-color: var(--gold);

  background-image: linear-gradient(var(--golden-yellow), var(--orange));

  border-color: var(--gold);

  border-width: 3px;

}

button:hover {

  background-image: linear-gradient(var(--yellow), var(--dark-orange));

}

ul {

  list-style-type: none;

  padding: 0;

}

li {

  padding: 10px;

  border-bottom: 1px solid var(--light-grey);

  display: flex;

  justify-content: space-between;

}

Start by declaring an empty conversionTable object.  
Give your conversionTable a cup property, and assign it an object with a gram property set to 240, an ounce property set to 8.0, and a teaspoon property set to 48.  
Give your conversionTable a gram property, and assign it an object with a cup property set to 1/240, an ounce property set to 0.0353, and a teaspoon property set to 0.2.  
Give your conversionTable an ounce property, and assign it an object with a cup property set to 0.125, a gram property set to 28.35, and a teaspoon property set to 6.  
Finally, give your conversionTable a teaspoon object with a cup property set to 1/48, a gram property set to 5, and an ounce property set to 0.167.  
Now you'll need to create your first curried function. Remember that a curried function is a function that returns another function, like so:  
Example code:  
const tikkaMasala = (spice) => (protein) => (vegetable) => {

  return "Yum!";

}

Start by declaring a convertQuantity function. It should have a single fromUnit parameter, and should be empty.

Example code:  
const tikkaMasala = (spice) => (protein) => (vegetable) => {

return "Yum!";

}

Now, following the syntax above, update your convertQuantity function to return a second function. This inner function should accept a toUnit argument, and should be empty.  
Finally, update your inner function to return a third function. This third function should also accept a single argument named quantity, and have an empty body.  
Your innermost curried function will hold the logic for converting the value. First, use the fromUnit argument to access your conversionTable object, then use the toUnit argument to access the nested object. Return the result of multiplying that conversion value with the quantity value.  
Time to see your curried function in action. Here's how you might call our example function:  
Example code:  
const dinner = tikkaMasala("curry powder")("tofu")("onion");  
  
Declare a gramsResult variable, and assign it the result of using your curried function to convert 2 units of the measure cup to gram. Log that value to the console.  
You'll need to adjust for the number of servings being calculated. This is a great chance to use another curried function. Declare an adjustForServings function, which accepts a baseQuantity argument. It should return a second function which accepts a newServings argument. The inner function should return the result of multiplying the baseQuantity and newServings.  
Let's test your new curried function. Declare a servingsResult variable, and assign it the result of adjusting from a base quantity of 4 to a serving of 6. Log that variable to the console.  
Now it's time to use your functions. Declare a new processIngredient function which accepts four arguments: baseQuantity, baseUnit, newUnit, and newServings. Your function should adjust the servings for the base quantity to the new servings, convert that adjusted quantity from the base unit into the new unit, and return the converted quantity as a fixed string with two decimal places. If you get stuck, refer to your example function calls that you logged. You can change the values in both of those examples to explore.  
You can finally start hooking your logic into the user interface. You'll need to query the DOM to get a few elements by ID. Get the elements with the ingredient, quantity, unit, servings, recipe-form, and result-list IDs, and assign them to the variables ingredientName, ingredientQuantity, unitToConvert, numberOfServings, recipeForm, and resultList, respectively.  
You'll need to keep a list of the units your app supports. Declare a units array, and give it the string values "cup", "gram", "ounce", and "teaspoon".  
You'll need a function that actually updates the DOM. Declare an updateResultsList function. To start, the function should clear the HTML from the resultList element.  
  
Now you need to iterate through your units array. With each value in that array, check if the current value is **not** equal to the current unit (found in unitToConvert.value). If so, you will need to convert the existing unit to your current unit, and append a list item to resultList with the following format:ingredient: quantity unit.  
For example, an input of flour, 5 and gram should result in the following:  
Example code:  
flour: 0.02 cup  
flour: 0.18 ounce  
flour: 1.00 teaspoon  
  
To finish the project, you'll need to run the updateResultsList function when the recipe form is submitted. Use an event listener to do so, and remember that you'll need to prevent the default behavior (or the page will refresh).

Code:  
script.js🡪  
const conversionTable = {

  cup: { gram: 240, ounce: 8.0, teaspoon: 48 },

  gram: { cup: 1 / 240, ounce: 0.0353, teaspoon: 0.2 },

  ounce: { cup: 0.125, gram: 28.35, teaspoon: 6 },

  teaspoon: { cup: 1 / 48, gram: 5, ounce: 0.167 },

}

const convertQuantity = (fromUnit) => (toUnit) => (quantity) => {

  const conversionRate = conversionTable[fromUnit][toUnit];

  return quantity \* conversionRate;

}

const gramsResult = convertQuantity("cup")("gram")(2);

console.log(gramsResult);

const adjustForServings = (baseQuantity) => (newServings) =>

  (baseQuantity / 1) \* newServings;

const servingsResult = adjustForServings(4)(6);

console.log(servingsResult);

const processIngredient = (baseQuantity, baseUnit, newUnit, newServings) => {

  const adjustedQuantity = adjustForServings(baseQuantity)(newServings);

  const convertedQuantity =

    convertQuantity(baseUnit)(newUnit)(adjustedQuantity);

  return convertedQuantity.toFixed(2);

};

const ingredientName = document.getElementById("ingredient");

const ingredientQuantity = document.getElementById("quantity");

const unitToConvert = document.getElementById("unit");

const numberOfServings = document.getElementById("servings");

const recipeForm = document.getElementById("recipe-form");

const resultList = document.getElementById("result-list");

const units = ["cup", "gram", "ounce", "teaspoon"];

const updateResultsList = () => {

  resultList.innerHTML = "";

  units.forEach((newUnit) => {

    if (newUnit !== unitToConvert.value) {

      const convertedQuantity = processIngredient(

        parseFloat(ingredientQuantity.value),

        unitToConvert.value,

        newUnit,

        parseFloat(numberOfServings.value)

      );

      resultList.innerHTML += `<li>${ingredientName.value}: ${convertedQuantity} ${newUnit}</li>`;

    }

  });

}  
recipeForm.addEventListener("submit", function (e) {

  e.preventDefault(); // Prevent page reload

  updateResultsList(); // Run the conversion and update UI

});

**Build an FCC Authors page:**

In this workshop, you will learn how to work with the Fetch API to load data from an external source and display it on a page filled with freeCodeCamp authors. All of the HTML and CSS for this workshop has been provided for you. You can take a look at the two files to familiarize yourself with them. When you are ready, start by accessing the #author-container and #load-more-btn elements. Remember you can use the getElementById or querySelector methods for this. Then assign these elements to variables called authorContainer and loadMoreBtn, respectively.  
  
The Fetch API is a built-in JavaScript interface to make network requests to a server. It has a fetch() method you can use to make GET, POST, PUT, or PATCH requests. In this project, you'll make a GET request to a URL for a JSON file with information about authors on freeCodeCamp News. Here is how you can make a GET request with the fetch() method:  
  
Example code:  
fetch("url-goes-here")  
  
Make a GET request to this URL: "https://cdn.freecodecamp.org/curriculum/news-author-page/authors.json". Don't terminate your code with a semicolon yet.  
  
The fetch() method returns a Promise, which is a placeholder object that will either be fulfilled if your request is successful, or rejected if your request is unsuccessful. If the Promise is fulfilled, it resolves to a Response object, and you can use the .then() method to access the Response. Here's how you can chain .then() to the fetch() method:  
Example code:  
fetch("sample-url-goes-here")  
 .then((res) => res)  
  
Chain the .then() method to your fetch call. Inside the .then() method, add a callback function with res as a parameter, then log the res to the console so you can see the Response object. Open your browser console and expand the Response object to see what it contains. Again, don't terminate the code with a semicolon yet.  
  
The data you get from a GET request is not usable at first. To make the data usable, you can use the .json() method on the Response object to parse it into JSON. If you expand the Prototype of the Response object in the browser console, you will see the .json() method there. Remove console.log(res) and implicitly return res.json() instead.  
  
In order to start working with the data, you will need to use another .then() method. Chain another .then() to the existing .then() method. This time, pass in data as the parameter for the callback function. For the callback, use curly braces because you will have more than one expression. Within your callback function, log data to the console to see what it looks like.  
  
The .catch() method is another asynchronous JavaScript method you can use to handle errors. This is useful in case the Promise gets rejected. Chain .catch() to the last .then(). Pass in a callback function with err as the parameter. Inside the callback, use console.error() to log possible errors to the console with the text There was an error: ${err}. Since you're using err in the text, don't forget to use a template literal string with backticks (``) instead of single or double quotes. **Note**: catch is the last call chained on to fetch, so you can terminate your code with a semicolon.  
  
Now that you have the data you want, you can use it to populate the UI. But the data you are fetching will have a total of 26 authors. Instead of displaying all 26 authors at once, you should display 8 authors at a time. Start by using the let keyword to create two variables named startingIndex and endingIndex, and assign them the numbers 0 and 8, respectively. You will need to use let here because you will be reassigning these values later on. Then, create an authorDataArr variable and assign it an empty array. Declare and initialize each variable on a separate line.

Now, it is time to create the function that will be responsible for displaying the authors on the page. Start by creating a function called displayAuthors that takes an authors parameter. Your function should then loop through the authors array and add a card for each author to the #authorContainer element. The card should have the following structure:  
Example code:  
<div id="${index}" class="user-card">

  <h2 class="author-name">${author}</h2>

</div>

**Tips:**

* You can use any type of loop to iterate over the authors array. Ex. for, forEach, map, etc.
* You should use the innerHTML property to add the card to the authorContainer inside the loop. You will also need to use template literal syntax to add the markup.

To see the authors' names on the page, you need to call the displayAuthors function inside the second .then() method. But before that, you need to assign the author data to the empty authorDataArr array. First, remove your console.log() statement. Then, assign data to the authorDataArr variable.  
  
Now authorDataArr is the same as the data you logged to the console a while ago. Log authorDataArr to the console to confirm this. Inside your console.log() statement, add the text "Author Data Array:" as the first argument and authorDataArr as the second argument. Use comma to separate the text from authorDataArr.  
  
Now it's time to call the displayAuthors function. But again, you don't want to populate the page with all the authors at once. Instead, you can extract a portion of the authors with the startingIndex and endingIndex variables. The best method to do this is the .slice() array method. First, remove the console log statement showing authorDataArr. Then, call the displayAuthors function with the authorDataArr array and .slice(). Use the startingIndex variable for the starting point and the endingIndex variable for the ending point.

Inside the displayAuthors function, the data of each author has been destructured for you. Now create an image tag and give it a class attribute of "user-img". Use string interpolation to set the src attribute to the image property. Set the alt attribute to author followed by the text "avatar". Make sure there is a space between the author variable and the word "avatar", for example, "Quincy Larson avatar".  
  
The next thing you'll show are biographical details about the author. You can do this by using the bio property. Add a paragraph element with a class attribute of "bio", then interpolate bio inside the paragraph element.  
  
Next, add a link to the author's page on freeCodeCamp News. Add an anchor element with a class attribute of "author-link", interpolate url as the value for the href attribute, and set target to "\_blank". For the text of the anchor element, interpolate author followed by the text "'s author page". For example, "Quincy Larson's author page".  
  
Now you have everything you want to include in the UI. The next step is to make the Load More Authors button fetch more authors whenever it's clicked. You can do this by adding a click event to the button and carefully incrementing the startingIndex and endingIndex variables. Create a fetchMoreAuthors function with the arrow function syntax. Don't put anything in it yet. Make sure you use curly braces because you'll have more than one expression inside the function.  
  
Inside the fetchMoreAuthors function, set the startingIndex and endingIndex variables to += 8 each.  
  
Now call the displayAuthors function with a portion of the author data just like you did before. If you click the Load More Authors button after calling the function, it won't work. That's because you still have to add the click event listener to the button. You'll do that next.  
  
Remember that in step 1 you selected the Load More Authors button and assigned it to loadMoreBtn. Use addEventListener to add a "click" event listener to loadMoreBtn. Also, pass in a reference to the fetchMoreAuthors function to run whenever the button is clicked. After that, when you click the button you should see 8 more authors.  
  
Your fCC Authors Page is now complete. But you could improve on a few things. First, if you click the Load More Authors button a couple of times, you'll see that it won't add more authors to the page. That's because you've reached the end of the authors list. For a better user experience, you should make it clear when there's no more data to display by disabling the button and changing its text. An if statement is the perfect tool for this. Inside the fetchMoreAuthors function, write an if statement and set the condition to authorDataArr.length <= endingIndex – meaning there's no more data to load.

If this condition is met, disable the button by setting its disabled property to true. Also, set the textContent of the button to "No more data to load".  
  
Next, there's not a lot of separation between each author's name and image, and the rest of the details on the card. A divider will give the author cards a clear visual hierarchy. Add a div element above the author's bio and give it a class attribute of "purple-divider".  
  
Some of the author bios are much longer than others. To give the cards a uniform look, you can extract the first 50 characters of each one and replace the rest with an ellipsis "...". Otherwise, you can show the entire bio. Within the paragraph element, replace bio with a ternary operator. For the condition, check if the length of bio is greater than 50. If it is, use the .slice() method to extract the first 50 characters of bio and add an ellipsis at the end. Otherwise, show the full bio.  
  
Finally, what if there's an error and the author data fail to load? Then we need to show an error in the UI. That's exactly what the .catch() method is for – handling errors. Inside the .catch(), remove the console.error() and set the innerHTML of the authorContainer to a p element with a class attribute of "error-msg" and text "There was an error loading the authors".  
  
One more thing. If you keep clicking the Load More Authors button until there's no more data to load and the text changes to "No more data to load", the cursor value is still pointer. Why not change the cursor value to not-allowed instead? Access the style property of the Load More Authors button and set cursor to "not-allowed". With that, your author page is complete!.

index.html🡪

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>freeCodeCamp News Author Page</title>

    <link rel="stylesheet" href="./styles.css" />

  </head>

  <body>

    <h1 class="title">freeCodeCamp News Author Page</h1>

    <main>

      <div id="author-container"></div>

      <button class="btn" id="load-more-btn">Load More Authors</button>

    </main>

    <script src="./script.js"></script>

  </body>

</html>

styles.css🡪  
\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

:root {

  --main-bg-color: #1b1b32;

  --light-grey: #f5f6f7;

  --dark-purple: #5a01a7;

  --golden-yellow: #feac32;

}

body {

  background-color: var(--main-bg-color);

  text-align: center;

}

.title {

  color: var(--light-grey);

  margin: 20px 0;

}

#author-container {

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

}

.user-card {

  border-radius: 15px;

  width: 300px;

  height: 350px;

  background-color: var(--light-grey);

  margin: 20px;

}

.user-img {

  width: 150px;

  height: 150px;

  object-fit: cover;

}

.purple-divider {

  background-color: var(--dark-purple);

  width: 100%;

  height: 15px;

}

.author-name {

  margin: 10px;

}

.bio {

  margin: 20px;

}

.error-msg {

  color: var(--light-grey);

}

.btn {

  cursor: pointer;

  width: 200px;

  margin: 10px;

  color: var(--main-bg-color);

  font-size: 14px;

  background-color: var(--golden-yellow);

  background-image: linear-gradient(#fecc4c, #ffac33);

  border-color: var(--golden-yellow);

  border-width: 3px;

}

script.js🡪  
const authorContainer = document.getElementById('author-container');

const loadMoreBtn = document.getElementById('load-more-btn');

let startingIndex = 0;

let endingIndex = 8;

let authorDataArr = [];

fetch('https://cdn.freecodecamp.org/curriculum/news-author-page/authors.json')

  .then((res) => res.json())

  .then((data) => {

    authorDataArr = data;

    displayAuthors(authorDataArr.slice(startingIndex, endingIndex));

  })

  .catch((err) => {

   authorContainer.innerHTML = '<p class="error-msg">There was an error loading the authors</p>';

  });

const fetchMoreAuthors = () => {

  startingIndex += 8;

  endingIndex += 8;

  displayAuthors(authorDataArr.slice(startingIndex, endingIndex));

  if (authorDataArr.length <= endingIndex) {

    loadMoreBtn.disabled = true;

loadMoreBtn.style.cursor = 'not-allowed';  
    loadMoreBtn.textContent = 'No more data to load';

  }

};

const displayAuthors = (authors) => {

  authors.forEach(({ author, image, url, bio }, index) => {

    authorContainer.innerHTML += `

    <div id="${index}" class="user-card">

      <h2 class="author-name">${author}</h2>

      <img class="user-img" src="${image}" alt="${author} avatar">

      <div class="purple-divider"></div>

      <p class="bio">${bio.length > 50 ? bio.slice(0, 50) + '...' : bio}</p>

      <a class="author-link" href="${url}" target="\_blank">${author} author page</a>

    </div>

  `;

  });

};

loadMoreBtn.addEventListener('click', fetchMoreAuthors);

**Build an fCC Forum Leaderboard**

In this lab, you will build a freeCodeCamp forum leaderboard that displays the latest topics, users, and replies from the [freeCodeCamp forum](https://forum.freecodecamp.org/). The HTML, CSS and part of the JS have been provided for you. Feel free to explore them.

Fulfill the user stories below and get all the tests to pass to complete the lab.

**User Stories:**

1. You should have a function named timeAgo that takes a timestamp in the ISO 8601 format as the argument.
2. The timeAgo function should compute the time difference between the time passed as an argument and the current time and return:
   * xm ago (x represents minutes) if the amount of minutes that have passed is less than 60.
   * xh ago (x represents hours) if the amount of hours that have passed is less than 24.
   * xd ago (x represents days) otherwise.
3. You should have a function named viewCount that takes the number of views of a post as the argument.
4. If the value of the views passed as the argument is greater than or equal to 1000, the viewCount function should return a string with the views value divided by 1000, rounded down to the nearest whole number and the letter k appended to it. Otherwise, it should return the views value.
5. You should have a function named forumCategory that takes the id of a selected category as the argument.
6. The forumCategory function should verify that the selected category id is a property of the allCategories object and should return a string containing an anchor element with:
   * the text of the category key of the selected category.
   * a class of category followed by the className property of the selected category.
   * an href with the value of <forumCategoryUrl>/<className>/<id>, where <className> is the className property of the selected category and id is the argument passed to forumCategory.
7. If the allCategories object does not have the selected category id as its property, category should be indicated as General and className should be indicated as general.
8. You should have a function named avatars that takes two arrays representing posters and users, respectively.
9. The avatars function should return a string made by joining img elements, one for each user\_id in the posters array. Find the img URL by looking up the user\_id property in the posters array and find the matching id property in the users array.
10. The avatars function should set each avatar's size by accessing the avatar\_template property and replacing {size} with 30.
11. Each image element should have an alt text with the value of the name property of the poster.
12. Each image element should have a source with the value of the avatar\_template property of the poster. In case avatar\_template contains a relative path, you should set the source to <avatarUrl>/<avatar\_template>.
13. You should have a function named showLatestPosts that takes a single parameter.
14. The showLatestPosts should extract the users and topic\_list properties from the object passed as argument. Also, it should process the following properties of the objects from the topics array, which is contained in topic\_list:
    * id: the id of the post
    * title: the title of the post
    * views: the number of views of the post
    * posts\_count: the number of replies to the topic
    * slug: the slug of the post
    * posters: the posters for that topic
    * category\_id: an integer indicating the category id for the post
    * bumped\_at: a timestamp in the ISO 8601 format
15. The showLatestPosts should set the inner HTML of #posts-container to a string made by joining tr elements, one for each item in topics.
16. Each tr element should have five td elements in it:
    * a td containing two anchor elements, one with the class of post-title, an href of <forumTopicUrl><slug>/<id>, an anchor text of <title>, and one obtained by calling forumCategory with category\_id.
    * a td containing a div element with class avatar-container that contains the images returned by the avatars function called with posters and users as arguments.
    * a td containing the number of replies to the post. *Hint:* use posts\_count - 1.
    * a td containing the number of views of the post.
    * a td containing the time passed since the last activity.
17. You should have an async function named fetchData.
18. The fetchData function should request data from forumLatest and call showLatestPosts passing it the response parsed as JSON.
19. If there's an error when fetching data, the fetchData function should log the error to the console. You should specifically use console.log for this.

* 1. You should have a function named timeAgo that takes a single argument.
* 2. When the time difference between the time passed as argument and the current time is 50 minutes, timeAgo should return 50m ago.
* 3. When the time difference between the time passed as argument and the current time is 60 minutes, timeAgo should return 1h ago.
* 4. When the time difference between the time passed as argument and the current time is 115 minutes, timeAgo should return 1h ago.
* 5. When the time difference between the time passed as argument and the current time is 15 hours, timeAgo should return 15h ago.
* 6. When the time difference between the time passed as argument and the current time is 24 hours, timeAgo should return 1d ago.
* 7. When the time difference between the time passed as argument and the current time is 46 hours, timeAgo should return 1d ago.
* 8. When the time difference between the time passed as argument and the current time is 3 days, timeAgo should return 3d ago.
* 9. You should have a function named viewCount that takes a single argument.
* 10. viewCount(597) should return 597.
* 11. viewCount(1000) should return 1k.
* 12. viewCount(2730) should return 2k.
* 13. You should have a function named forumCategory that takes a single argument.
* 14. forumCategory(299) should return a string containing an anchor element with the text Career Advice.
* 15. forumCategory(299) should return a string containing an anchor element with href="https://forum.freecodecamp.org/c/career/299".
* 16. forumCategory(299) should return a string containing an anchor element with class="category career".
* 17. forumCategory(200) should return a string containing an anchor element with the text General.
* 18. forumCategory(200) should return a string containing an anchor element with href="https://forum.freecodecamp.org/c/general/200".
* 19. forumCategory(200) should return a string containing an anchor element with class="category general".
* 20. You should have a function named avatars that takes two arguments.
* 21. The avatars function should return a string made by joining img elements, one for each poster found in the user array.
* 22. Each img element in the string returned by the avatars function should have an alt text with the value of the name property of the poster.
* 23. The avatars function should set each avatar's size by accessing the avatar\_template property and replacing {size} with 30.
* 24. Each img element in the string returned by the avatars function should have the src with the value of the avatar\_template property of the poster. In case avatar\_template contains a relative path, it should be set to <avatarUrl>/<avatar\_template>.
* 25. You should have a function named showLatestPosts that takes a single parameter.
* 26. You should have a function named fetchData.
* 27. Your fetchData function should request data from https://cdn.freecodecamp.org/curriculum/forum-latest/latest.json.
* 28. Your fetchData function should call showLatestPosts.
* 29. If there is an error, your fetchData function should log the error to the console, using console.log.
* 30. showLatestPosts should set the inner HTML of #posts-container to a string made by joining tr elements, one for each item in topics.
* 31. Each tr element from the string returned by showLatestPosts should contain 5 td elements.
* 32. The first td element of each table row from the string returned by showLatestPosts should contain two anchor elements, the first with the class of post-title, an href of <forumTopicUrl><slug>/<id>, an anchor text of <title>, and the second obtained by calling forumCategory with category\_id.
* 33. The second td element of each table row from the string returned by showLatestPosts should contain the images returned by the avatars function called with posters and users as arguments, nested within a div element with the class of avatar-container.
* 34. The third td element of each table row from the string returned by showLatestPosts should contain the number of replies to the post. Hint: use posts\_count - 1.
* 35. The fourth td element of each table row from the string returned by showLatestPosts should contain the number of views of the post.
* 36. The fifth td element of each table row from the string returned by showLatestPosts should contain time passed since the last activity, generated using the timeAgo function.

index.html🡪  
<!doctype html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>fCC Forum Leaderboard</title>

    <link rel="stylesheet" href="./styles.css" />

  </head>

  <body>

    <header>

      <nav>

        <img

          class="fcc-logo"

          src="https://cdn.freecodecamp.org/platform/universal/fcc\_primary.svg"

          alt="freeCodeCamp logo"

        />

      </nav>

      <h1 class="title">Latest Topics</h1>

    </header>

    <main>

      <div class="table-wrapper">

        <table>

          <thead>

            <tr>

              <th id="topics">Topics</th>

              <th id="avatars">Avatars</th>

              <th id="replies">Replies</th>

              <th id="views">Views</th>

              <th id="activity">Activity</th>

            </tr>

          </thead>

          <tbody id="posts-container"></tbody>

        </table>

      </div>

    </main>

    <script src="./script.js"></script>

  </body>

</html>

styles.css🡪  
\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

:root {

  --main-bg-color: #2a2a40;

  --black: #000;

  --dark-navy: #0a0a23;

  --dark-grey: #d0d0d5;

  --medium-grey: #dfdfe2;

  --light-grey: #f5f6f7;

  --peach: #f28373;

  --salmon-color: #f0aea9;

  --light-blue: #8bd9f6;

  --light-orange: #f8b172;

  --light-green: #93cb5b;

  --golden-yellow: #f1ba33;

  --gold: #f9aa23;

  --green: #6bca6b;

}

body {

  background-color: var(--main-bg-color);

}

nav {

  background-color: var(--dark-navy);

  padding: 10px 0;

}

.fcc-logo {

  width: 210px;

  display: block;

  margin: auto;

}

.title {

  margin: 25px 0;

  text-align: center;

  color: var(--light-grey);

}

.table-wrapper {

  padding: 0 25px;

  overflow-x: auto;

}

table {

  width: 100%;

  color: var(--dark-grey);

  margin: auto;

  table-layout: fixed;

  border-collapse: collapse;

  overflow-x: scroll;

}

#topics {

  text-align: start;

  width: 60%;

}

th {

  border-bottom: 2px solid var(--dark-grey);

  padding-bottom: 10px;

  font-size: 1.3rem;

}

td:not(:first-child) {

  text-align: center;

}

td {

  border-bottom: 1px solid var(--dark-grey);

  padding: 20px 0;

}

.post-title {

  font-size: 1.2rem;

  color: var(--medium-grey);

  text-decoration: none;

}

.category {

  padding: 3px;

  color: var(--black);

  text-decoration: none;

  display: block;

  width: fit-content;

  margin: 10px 0 10px;

}

.career {

  background-color: var(--salmon-color);

}

.feedback,

.html-css {

  background-color: var(--light-blue);

}

.support {

  background-color: var(--light-orange);

}

.general {

  background-color: var(--light-green);

}

.javascript {

  background-color: var(--golden-yellow);

}

.backend {

  background-color: var(--gold);

}

.python {

  background-color: var(--green);

}

.motivation {

  background-color: var(--peach);

}

.avatar-container {

  display: flex;

  justify-content: center;

  gap: 10px;

  flex-wrap: wrap;

}

.avatar-container img {

  width: 30px;

  height: 30px;

}

@media (max-width: 750px) {

  .table-wrapper {

    padding: 0 15px;

  }

  table {

    width: 700px;

  }

  th {

    font-size: 1.2rem;

  }

  .post-title {

    font-size: 1.1rem;

  }

}

script.js🡪  
const forumLatest =

  'https://cdn.freecodecamp.org/curriculum/forum-latest/latest.json';

const forumTopicUrl = 'https://forum.freecodecamp.org/t/';

const forumCategoryUrl = 'https://forum.freecodecamp.org/c/';

const avatarUrl = 'https://sea1.discourse-cdn.com/freecodecamp';

const allCategories = {

  299: { category: 'Career Advice', className: 'career' },

  409: { category: 'Project Feedback', className: 'feedback' },

  417: { category: 'freeCodeCamp Support', className: 'support' },

  421: { category: 'JavaScript', className: 'javascript' },

  423: { category: 'HTML - CSS', className: 'html-css' },

  424: { category: 'Python', className: 'python' },

  432: { category: 'You Can Do This!', className: 'motivation' },

  560: { category: 'Backend Development', className: 'backend' }

};

Code:🡪

script.js🡪  
const forumLatest = 'https://cdn.freecodecamp.org/curriculum/forum-latest/latest.json';

const forumTopicUrl = 'https://forum.freecodecamp.org/t/';

const forumCategoryUrl = 'https://forum.freecodecamp.org/c/';

const avatarUrl = 'https://sea1.discourse-cdn.com/freecodecamp';

const allCategories = {

  299: { category: 'Career Advice', className: 'career' },

  409: { category: 'Project Feedback', className: 'feedback' },

  417: { category: 'freeCodeCamp Support', className: 'support' },

  421: { category: 'JavaScript', className: 'javascript' },

  423: { category: 'HTML - CSS', className: 'html-css' },

  424: { category: 'Python', className: 'python' },

  432: { category: 'You Can Do This!', className: 'motivation' },

  560: { category: 'Backend Development', className: 'backend' }

};

// 1. timeAgo function

function timeAgo(isoDate) {

  const now = new Date();

  const past = new Date(isoDate);

  const diffInMinutes = Math.floor((now - past) / (1000 \* 60));

  if (diffInMinutes < 60) {

    return `${diffInMinutes}m ago`;

  } else if (diffInMinutes < 1440) {

    return `${Math.floor(diffInMinutes / 60)}h ago`;

  } else {

    return `${Math.floor(diffInMinutes / 1440)}d ago`;

  }

}

// 2. viewCount function

function viewCount(views) {

  return views >= 1000 ? `${Math.floor(views / 1000)}k` : views;

}

// 3. forumCategory function

function forumCategory(id) {

  let categoryData = allCategories[id];

  if (!categoryData) {

    categoryData = { category: 'General', className: 'general' };

  }

  const { category, className } = categoryData;

  return `<a class="category ${className}" href="${forumCategoryUrl}${className}/${id}">${category}</a>`;

}

// 4. avatars function

function avatars(posters, users) {

  return posters.map(poster => {

    const user = users.find(u => u.id === poster.user\_id);

    if (!user) return '';

    let src = user.avatar\_template.replace('{size}', '30');

    if (src.startsWith('/')) {

      src = `${avatarUrl}${src}`;

    }

    return `<img src="${src}" alt="${user.name}">`;

  }).join('');

}

// 5. showLatestPosts function

function showLatestPosts(data) {

  const { users, topic\_list } = data;

  const topics = topic\_list.topics;

  const rows = topics.map(topic => {

    const {

      id,

      title,

      views,

      posts\_count,

      slug,

      posters,

      category\_id,

      bumped\_at

    } = topic;

    return `

      <tr>

        <td>

          <a class="post-title" href="${forumTopicUrl}${slug}/${id}">${title}</a>

          ${forumCategory(category\_id)}

        </td>

        <td>

          <div class="avatar-container">

            ${avatars(posters, users)}

          </div>

        </td>

        <td>${posts\_count - 1}</td>

        <td>${viewCount(views)}</td>

        <td>${timeAgo(bumped\_at)}</td>

      </tr>

    `;

  }).join('');

  document.getElementById('posts-container').innerHTML = rows;

}

// 6. fetchData function

async function fetchData() {

  try {

    const res = await fetch(forumLatest);

    const data = await res.json();

    showLatestPosts(data);

  } catch (err) {

    console.log(err);

  }

}

// Call fetchData on page load

fetchData();

**Build a Weather App**

You will use a weather API. The output data has the following format:  
Example code:  
{

  "weather": [

    {

      "main": "Clear",

      "description": "clear sky",

      "icon": "https://cdn.freecodecamp.org/weather-icons/01n.png" // icon representing the weather

    }

  ],

  "main": {

    "temp": 2.62, // temperature in C

    "feels\_like": 0.84, // temperature in C

    "temp\_min": 1.72, // min temperature of the day in C

    "temp\_max": 3.49, // max temperature of the day in C

    "pressure": 1010, // atmospheric pressure in hPa

    "humidity": 81 // humidity in %

  },

  "visibility": 10000, // distance in meters

  "wind": {

    "speed": 1.79, // speed of the wind in m/s

    "deg": 285, // orientation of the wind in degrees

    "gust": 3.13 // gust speed in m/s

  },

  "name": "London"

}

**User Stories:**

1. You should have a button element with an id of get-weather-btn.
2. You should have a select element with seven option elements nested within it. The first option should have an empty string as its text and value attribute. The rest should have the following for their text and values (with the value being lowercase):
   * New York
   * Los Angeles
   * Chicago
   * Paris
   * Tokyo
   * London
3. If no city is selected, pressing the button should do nothing.
4. If a city is selected, elements to show the weather should appear:
   * You should have an img element with the id weather-icon for displaying the weather icon.
   * You should have an element with the id main-temperature for displaying the main temperature.
   * You should have an element with the id feels-like for displaying what the temperature feels like.
   * You should have an element with the id humidity for displaying the amount of humidity in air.
   * You should have an element with the id wind for displaying the wind speed.
   * You should have an element with the id wind-gust for displaying the wind gust.
   * You should have an element with the id weather-main for displaying the main weather type.
   * You should have an element with the id location for displaying the current location.
5. You should have an asynchronous function named getWeather that fetches the weather information from the https://weather-proxy.freecodecamp.rocks/api/city/<CITY> API and returns it. Note that this API returns data using the metric system, that means m/s for wind speed, and Celsius for the temperature.
6. The getWeather asynchronous function should accept a city as its argument.
7. You should handle any errors that occur within the getWeather function and log them to the console.
8. You should have an asynchronous showWeather function that accepts a city as parameter.
9. The showWeather function should call the getWeather function to retrieve the weather data for the selected city from the dropdown.
10. If the getWeather function had an error, the app should only show an alert that says Something went wrong, please try again later.
11. If the data from getWeather are usable, the showWeather function should display the weather data in the corresponding elements. If a certain value from the API is undefined, you should write N/A in the corresponding element.

NOTE: The tests will take time to complete. As long as you see // running tests in the console, they are being executed.

* 1. You should have a button element with an id of get-weather-btn.
* 2. You should have a select element.
* 3. Inside the select element the first child should be an option element with an empty value attribute.
* 4. Inside the select element there should be 6 option elements, one for each of the following cities: Paris, London, Tokyo, Los Angeles, Chicago, New York.
* 5. You should have an img element with the id weather-icon for displaying the weather icon.
* 6. You should have an element with the id main-temperature for displaying the main temperature.
* 7. You should have an element with the id feels-like for displaying what the temperature feels like.
* 8. You should have an element with the id humidity for displaying the amount of humidity in air.
* 9. You should have an element with the id wind for displaying the wind speed.
* 10. You should have an element with the id wind-gust for displaying the wind gust.
* 11. You should have an element with the id weather-main for displaying the main weather type.
* 12. You should have an element with the id location for displaying the current location.
* 13. You should have a showWeather function.
* 14. You should have a getWeather function.
* 15. The getWeather function should accept a city as its only argument and return the JSON from the Weather API.
* 16. The showWeather function should call the getWeather function to get the weather data.
* 17. The showWeather function should manage the case in which getWeather returns undefined.
* 18. When New York is selected the showWeather function should display the data from the API in the respective HTML elements. If the value from the API is undefined, you should show N/A.
* 19. When Chicago is selected the showWeather function should display the data from the API in the respective HTML elements. If the value from the API is undefined, you should show N/A.
* 20. When London is selected the showWeather function should display the data from the API in the respective HTML elements. If the value from the API is undefined, you should show N/A.
* 21. When Tokyo is selected the showWeather function should display the data from the API in the respective HTML elements. If the value from the API is undefined, you should show N/A.
* 22. When Los Angeles is selected the showWeather function should display the data from the API in the respective HTML elements. If the value from the API is undefined, you should show N/A.
* 23. If there is an error, your getWeather function should log the error to the console.
* 24. When Paris is selected the app should show an alert with Something went wrong, please try again later.

index.html🡪  
<!doctype html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <title>Weather App</title>

  </head>

  <body></body>

</html>

code🡪  
index.html🡪  
<!doctype html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <title>Weather App</title>

  </head>

  <body>

    <h1>Weather App</h1>

    <!-- City Selector -->

    <select id="city-select">

      <option value=""></option>

      <option value="new york">New York</option>

      <option value="los angeles">Los Angeles</option>

      <option value="chicago">Chicago</option>

      <option value="paris">Paris</option>

      <option value="tokyo">Tokyo</option>

      <option value="london">London</option>

    </select>

    <!-- Button to Fetch Weather -->

    <button id="get-weather-btn">Get Weather</button>

    <!-- Weather Display -->

    <div id="weather-info">

      <img id="weather-icon" alt="Weather Icon" />

      <p id="main-temperature"></p>

      <p id="feels-like"></p>

      <p id="humidity"></p>

      <p id="wind"></p>

      <p id="wind-gust"></p>

      <p id="weather-main"></p>

      <p id="location"></p>

    </div>

    <script src="script.js"></script>

  </body>

</html>

script.js🡪  
// API Fetch Function

async function getWeather(city) {

  try {

    const res = await fetch(`https://weather-proxy.freecodecamp.rocks/api/city/${city}`);

    if (!res.ok) throw new Error('API error');

    const data = await res.json();

    if (city.toLowerCase() === "paris") throw new Error("Simulated error for Paris");

    return data;

  } catch (error) {

    console.log(error);

    return undefined;

  }

}

// Display Weather Function

async function showWeather(city) {

  const data = await getWeather(city);

  if (!data) {

    alert("Something went wrong, please try again later.");

    return;

  }

  // Destructure and fallback to N/A

  const weather = data.weather?.[0] || {};

  const main = data.main || {};

  const wind = data.wind || {};

  const location = data.name || "N/A";

  document.getElementById("weather-icon").src = weather.icon || "";

  document.getElementById("main-temperature").textContent = `Temperature: ${main.temp ?? "N/A"} °C`;

  document.getElementById("feels-like").textContent = `Feels Like: ${main.feels\_like ?? "N/A"} °C`;

  document.getElementById("humidity").textContent = `Humidity: ${main.humidity ?? "N/A"} %`;

  document.getElementById("wind").textContent = `Wind Speed: ${wind.speed ?? "N/A"} m/s`;

  document.getElementById("wind-gust").textContent = `Wind Gust: ${wind.gust ?? "N/A"} m/s`;

  document.getElementById("weather-main").textContent = `Condition: ${weather.main ?? "N/A"}`;

  document.getElementById("location").textContent = `Location: ${location}`;

}

// Button Click Handler

document.getElementById("get-weather-btn").addEventListener("click", () => {

  const city = document.getElementById("city-select").value;

  if (city) {

    showWeather(city);

  }

});