Decimal to Binary Converter

Your Convert button should be working now. But it could get tiring for users to enter in a number, then click that button each time they want to convert from decimal to binary. It would be much more convenient to perform the conversion when the Enter or Return key is pressed.

The keydown event fires every time a user presses a key on their keyboard, and is a good way to add more interactivity to input elements.

Chain .addEventListener() to numberInput. The event listener should listen for keydown events and take an empty arrow function as a callback.

Step 10

In an earlier project you learned about truthy and falsy values, which are values that evaluate to true or false. In JavaScript, some common falsy values you'll see are null, undefined, the number 0, and empty strings.

Rather than check if a value is equal to a falsy value, you can use the *logical NOT* operator (!) to check if the value itself is falsy. For example:

Example Code

```
const num = 0;


console.log(num === 0); // true
console.log(!num); // true
```

Update the condition in your if statement to use the logical NOT operator to check if numberInput.value is falsy.

Step 11

Because the input type="number" element allows special characters like ., +, and e, users can input floats like 2.2, equations like 2e+3, or even just e, which you don't want to allow.

A good way to check and normalize numbers in JavaScript is to use the built-in parseInt() function, which converts a string into an integer or whole number. parseInt() takes at least one argument, a string to be converted into an integer, and returns either an integer or NaN which stands for Not a Number. For example:

Example Code

```
parseInt(2.2); // 2
parseInt("2e+3"); // 2
parseInt("e") // NaN
```

Add a logical OR operator (||) after the first condition in your if statement. Then, pass the value of numberInput into the parseInt() function as the second condition of your if statement.

Step 12

Next, you need to check if the value returned by the parseInt() function is a number or not.

To do that, you can use the isNaN() function. This function takes in a string or number as an argument, and returns true if it evaluates to NaN. For example:

Example Code

```
isNaN("test"); // true
isNaN(2); // false
isNaN("3.5"); // false
```

Update the second condition in your if statement to use the isNaN() function to check if the value returned by parseInt() is NaN.

Also,as we mentioned in step 1 that we are considering only positive numbers, we should add a third condition in if statement to check whether the number is less than 0 (i.e negative numbers)

Step 19

In the base-2 number system, the rightmost digit represents the ones place, the next digit to the left represents the twos place, then the fours place, then the eights place, and so on. In this system, each digit's place value is two times greater than the digit to its right.

Here are numbers zero to nine in the base-10 and base-2 number systems:

Example Code

Base-10	Base-2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Notice that binary numbers are formed from left to right, from the digit with the greatest place value on the left, to the least significant on the right. For example, the number 3 in binary is 11, or 1 in the twos place and 1 in the ones place. Then for the number 4, a digit to represent the fours place is included on the left and set to 1, the twos place is 0, and the ones place is 0.

In your decimalToBinary function, convert the number 10 into binary and return it as a string.

Step 20

Bits are often grouped into an octet, which is an 8-bit set known as a *byte*. A byte can represent any number between 0 and 255. Here are the placement values for each bit in a byte:

Example Code

```
128 | 64 | 32 | 16 | 8 | 4 | 2 | 1
```

Because bits are often grouped into bytes, it's common to see binary numbers represented in groups of eight, sometimes with leading zeros. For example, the number 52 can be represented as 110100, or 00110100 with leading zeros. Here's how that breaks down with the placement values:

Example Code

In your decimalToBinary function, convert the number 118 into binary with leading zeros and return it as a string.

For the decimal to binary conversion, you need to divide `input` by `2` until the *quotient*, or the result of dividing two numbers, is `0`. But since you don't know how many times you need to divide `input` by `2`, you can use a `while` loop to run a block of code as long as `input` is greater than `0` and can be divided.

As a reminder, a `while` loop is used to run a block of code as long as the condition evaluates to `true`, and the condition is checked before the code block is executed. For example:

Example Code

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Create a `while` loop that runs as long as `input` is greater than `0`. Leave the body of the loop empty for now.

To divide numbers in JavaScript, use the division operator (`/`). For example:

Example Code

```
const quotient = 5 / 2; // 2.5
```

In the example above, `5` is the *dividend*, or the number to be divided, and `2` is the *divisor*, or the number to divide by. The result, `2.5`, is called the *quotient*.

Inside your `while` loop, create a variable named `quotient` and assign it the value of `input` divided by `2`.

Like you saw in the last step, division can lead to a floating point number, or a number with a decimal point. The best way to handle this is to round down to the nearest whole number.

Use the `Math.floor()` function to round down the quotient of `input` divided by `2` before it's assigned to `quotient`.

Step 28

Next, you need to calculate the remainder of input divided by 2. You can do this by using the *remainder operator* (%), which returns the remainder of the division of two numbers. For example:

Example Code

```
const remainder = 5 % 2; // 1
```

In other words, the dividend, 5, can be divided by the divisor, 2, multiple times. Then you're left with a remainder of 1.

Inside your while loop, create a variable named remainder and use the remainder operator to assign it the remainder of input divided by 2.

Now if you enter in the number 6 and click the Convert button, you'll see the following output:

Example Code

```
Inputs:  [ 6, 3, 1 ]
Quotients:  [ 3, 1, 0 ]
Remainders:  [ 0, 1, 1 ]
```

Notice that the remainders array is the binary representation of the number 6, but in reverse order.

Use the .reverse() method to reverse the order of the remainders array, and .join() with an empty string as a separator to join the elements into a binary number string. Then, set result.innerText equal to the binary number string.

In computer science, a *stack* is a data structure where items are stored in a *LIFO* (last-in-first-out) manner. If you imagine a stack of books, the last book you add to the stack is the first book you can take off the stack. Or an array where you can only .push() and .pop() elements.

The *call stack* is a collection of function calls stored in a stack structure. When you call a function, it is added to the top of the stack, and when it returns, it is removed from the top / end of the stack.

You'll see this in action by creating mock call stack.

Initialize a variable named callStack and assign it an empty array.

A recursive function is a function that calls itself over and over. But you have to be careful because you can easily create an infinite loop. That's where the *base case* comes in. The base case is when the function stops calling itself, and it is a good idea to write it first.

Since your `countdown()` function will count down from a given number to zero, the base case is when the `number` parameter is equal to `0`. Then it should `return` to break out of its recursive loop.

Use an `if` statement to check if `number` is equal to `0`. If it is, use the `return` keyword to break out of the function.

Step 73

Now everything should work as expected. And since you know that `input` will either be the numbers `0` or `1` at this point, you can combine your two base cases and just return `input` as a string.

For a reliable way to convert a value into a string, even falsy values like `null` and `undefined`, you can use the `String()` function. For example:

Example Code

```
const num = 5;


console.log(String(num)); // "5"
console.log(String(null)); // "null"
```

Combine your `if` and `else if` statements into a single `if` statement checking if `input` is equal to `0` or `1`. If it is, use the `String()` function to convert `input` into a string and return it.

Step 75

You'll show the animation when users try to convert the decimal number `5` to binary, so you'll need to add a check for that within your `checkUserInput()` function.

Use an `if` statement to check if the `value` attribute of `numberInput` is equal to the number `5`. Remember to use the `parseInt()` function to convert the string into a number before comparing it to `5`. Leave the `if` statement empty for now.

The setTimeout function takes two arguments: a callback function and a number representing the time in milliseconds to wait before executing the callback function.

For example, if you wanted to log "Hello, world!" to the console after 3 seconds, you would write:

Example Code

```
setTimeout(() => {
  console.log("Hello, world!");
}, 3000);
```

Use the setTimeout function to add a one second delay before the text "Code" is logged to the console. Then see what happens after you enter 5 into the number input and click the Convert button.

Step 82

While asynchronous, or async, code can be difficult to understand at first, it has many advantages. One of the most important is that it allows you to write non-blocking code.

For example, imagine you're baking a cake, and you put the cake in the oven and set a timer. You don't have to sit in front of the oven waiting the entire time – you can wash dishes, read a book, or do anything else while you wait for the timer to go off.

Async code works in a similar way. You can start an async operation and other parts of your code will still work while that operation is running.

You'll learn more about async code in future projects, but the setTimeout() function is a good introduction.

Add a 1500 millisecond delay before the text "Camp" is logged to the console.

tep 84

Next, you'll create an object to represent the first frame of your animation. Your object should have three properties or keys: inputVal, marginTop, and addElDelay.

inputVal will represent the value of the input each time your recursive function runs. marginTop will be the top margin for DOM elements you'll add to the page. And addElDelay will be the delay between adding DOM elements to the page.

Add an object to `animationData` with an `inputVal` property set to `5`, a `marginTop` property set to `300`, and an `addElDelay` property set to `1000`.

Recall that the call stack is a LIFO (last in, first out) data structure. This means that, as functions are called, they are added to the top or end of the stack, and as functions return, they are removed from the top of the stack.

Treat your `animationData` array as a stack and add a new object to it. Your new object should have the properties `inputVal`, `marginTop`, and `addElDelay` set to `2`, `-200`, and `1500`, respectively. Remember to add this object to the top of the stack, or in other words, to the end of the `animationData` array