LEARN THE DATE OBJECT BY BUILDING A DATE FORMATTER

Introduction:

Working with dates in JavaScript can be challenging. You have to navigate various methods, formats, and time zones. In this project, you'll learn how to work with the JavaScript Date object, including its methods and properties. You'll also learn how to correctly format dates.

This project will cover concepts such as the `getDate()`, `getMonth()`, and `getFullYear()` methods.

Step 1:

In this project, you will learn about the JavaScript `Date` object by building a date formatter.

All of the HTML and CSS for this project have been provided for you.

Start by getting the `#current-date` element using the `.getElementById()` method and assign it to a `const` variable called `currentDateParagraph`.

Step 2:

Use the `.getElementById()` method to get the `#date-options` element and use `const` to assign it to a variable named `dateOptionsSelectElement`.

Step 3:

In JavaScript, there are many built-in constructors that create objects. A constructor is like a regular function, but starts with a capital letter, and is initialized with the `new` operator.

For example, you can use the `Date()` constructor with the `new` operator to create a new `Date` object that returns a string with the current date and time:

Example Code:

```
const currentDate = new Date();

console.log(currentDate);



// Output:

// Mon Aug 23 2021 15:31:00 GMT-0400 (Eastern Daylight Time)
```

Create a new `const` variable called `date` and assign it a `Date` object with `new Date()`.

Step 4:

The `Date` object has a number of methods that allow you to get the date and time in different formats.

One of those is the `.getDate()` method, which returns a number between `1` and `31` that represents the day of the month for that date. For example:

Example Code:

```
const date = new Date();

const dayOfTheMonth = date.getDate();

console.log(dayOfTheMonth); // 20
```

Using `const`, create a variable named `day` and assign it the day of the month from `date` with the `.getDate()` method.

Step 5:

The `.getMonth()` method returns a number between `0` and `11`. This represents the month for the date provided, where `0` is January and `11` is December. Because the number this method returns is zero-based, you need to add `1` to it to get the expected month number.

Using `const`, create a variable named `month` and assign it the month from `date` with the `.getMonth()` method.

Remember to add `1` to the number returned by `.getMonth()`.

Step 6:

The `.getFullYear()` method returns a number which represents the year for the provided date.

Using `const`, create a variable named `year` and assign it the year from `date` with the `.getFullYear()` method.

Step 7:

The `.getHours()` method returns a number between `0` and `23`. This represents the hour for the provided date, where `0` is midnight and `23` is 11 p.m.

Create a `const` variable named `hours` and assign it the hour from `date` with the `.getHours()` method.

Step 8:

The `.getMinutes()` method returns a number between `0` and `59` which represents the minutes for the provided date.

Create a `const` variable named `minutes` and assign it the minutes from `date` with the `.getMinutes()` method.

Step 9:

Next, create a `const` variable named `formattedDate` and assign it empty template literals.

Step 10:

Inside the template literal, add an embedded expression that contains the day variable.

Step 11:

After the `day` variable, add a dash (`-`) followed by another embedded expression that contains the `month` variable.

Step 12:

After the `month` variable, add a dash followed by another embedded expression that contains the `year` variable.

Step 13:

To see the results of the formattedDate variable, add a console.log() statement and pass in the formattedDate variable as an argument.

Open up the console and you should see the date printed out.

Step 14:

Use the textContent property on currentDateParagraph to set its text content to the value of the formattedDate variable.

Also, make sure to remove your console.log(formattedDate); line.

Step 15:

In JavaScript, the change event is used to detect when the value of an HTML element has changed:

Example Code:

```
element.addEventListener("change", () => {

});
```

Attach the addEventListener method to the dateOptionsSelectElement. The first argument of the event listener should be the string "change" and the second argument should be an empty arrow function.

Step 16:

When a user makes a selection from the dropdown menu, the function should get the user's value and display the date in

their chosen date format. To do this, you can use the `switch` statement.

A `switch` statement is used to compare an expression against multiple possible values and execute different code blocks based on the match. It's commonly used for branching logic.

For example, here's how to compare the expression `dayOfWeek` against possible values:

Example Code:

```
switch (dayOfWeek) {
  case 1:
    console.log("It's Monday!");
    break;
  case 2:
    console.log("It's Tuesday!");
    break;
  // ...cases for other workdays
  default:
    console.log("It's the weekend!");
}
```

Create a `switch` statement and use `dateOptionsSelectElement.value` as the expression.

Step 17:

When the user chooses the `Year, Month, Day` option from the dropdown, the date format should reflect this choice.

To do this, you can add a `case` clause in the `switch` statement that checks for a match against the expression `expr`, followed by code to run if there's a match. Here's an example where the `case` clause checks that `expr` is equal to the string `"case123"`:

Example Code:

```
switch (expr) {

  case 'case123':

    // Write your logic here

}
```

Add a `case` where the value is `"yyyy-mm-dd"`. Inside the `case`, set the text content of `currentDateParagraph` to the value of `formattedDate`.

Step 18:

To format the date into `yyyy-mm-dd`, you will need to use the split, reverse, and join methods. But first, you will need to go through a few practice examples so you can better understand how to use them in the context of this project.

The `split()` method is used to divide a string into substrings based on a specified separator. It then returns these substrings as elements of an array.

Here is an example of taking the words `"Hello World"` and returning an array of one element:

Example Code:

```
const greeting = "Hello World";
```

```
greeting.split(); // ["Hello World"]
```

Create a new `const` variable called `exampleSentence` and assign it the result of `"selur pmaCedoCeerf".split()`.

Then add a console statement to log the value of `exampleSentence`. Open up the console to see the result.

Step 19:

The `split` method takes in a parameter known as a separator. The separator is used to tell the computer where each split should occur.

Here is an example of using an empty string as a separator:

```
Example Code:

// returns ["h", "e", "l", "l", "o"]

"hello".split("");
```

Other examples of separators can include a space `" "`, or a hyphen `"-"`. If you don't provide a separator, the method will return an array with the original string as the only element.

Update your `split` method, to use an empty string as a separator. Open up the console again to see the result.

Step 20:

To reverse an array of elements, you can use the `reverse` method. This method reverses the order of the elements in the array in place. The first element becomes the last, and the last element becomes the first.

Here is an example of using the reverse method:

Example Code:

```
// returns [5, 4, 3, 2, 1]

[1, 2, 3, 4, 5].reverse();
```

Chain the reverse method to your split method. Open up the console again to see the result.

Remember that you learned how to chain methods in the previous project like this:

Example Code:

```
method1().method2().method3();
```

Step 21:

In the previous project, you learned how to work with the join method. This method takes an array of elements and joins them into a string. Similar to the split method, the join method also takes an optional separator. If you don't provide a separator, the default separator is a comma.

Here is an example of using the join method:

Example Code:

```
// returns "1-2-3-4-5"

[1, 2, 3, 4, 5].join("-");
```

Chain the join method to your reverse method. Pass in an empty string as the separator.

Open up the console and see the output

Step 22:

Now that you have a better understanding on how to work with the `split`, `reverse`, and `join` methods, you can delete your `exampleSentence` variable and console statement.

Step 23:

Like in the previous step, use method chaining to `split`, `reverse`, and `join` the `formattedDate` variable. Use `'-'` in the split and join methods.

Test out your changes by selecting the `Year, Month, Day` option from the dropdown menu. The date should now be displayed in the format `yyyy-mm-dd`.

Step 24:

If your `switch` statement is going to have multiple cases, it is best practice to include a `break` statement.

The `break` statement will tell the JavaScript interpreter to stop executing statements. Without adding a `break` statement at the end of each `case` block, the program will execute the code for all matching `cases`:

Example Code:

```
switch (someVariable) {
  case 'case123':
    // Write your logic here
```

```
        break; // Terminates the switch statement
    }
```

Add a `break` statement to the end of your `case` block.

Step 25:

Add another `case` with the value `"mm-dd-yyyy-h-mm"`. Inside that `case`, set the text content of `currentDateParagraph` to empty template literals.

Also, make sure to include a `break` statement to terminate the `switch` statement.

Step 26:

When the user selects the `Month, Day, Year, Hours, Minutes` option from the dropdown, you need to display the date in the format `mm-dd-yyyy h Hours m Minutes`.

Inside the `case` for `mm-dd-yyyy-h-mm`, use string interpolation to assign the formatted date from above to the `textContent` property of `currentDateParagraph`. Make sure to use the `month`, `day`, `year`, `hours`, and `minutes` variables in your answer.

Step 27:

In a `switch` statement, the `default` case is executed when none of the previous case conditions match the value being evaluated. It serves as a catch-all for any other possible cases. For example:

Example Code:

```
const dayOfWeek = 7;


switch (dayOfWeek) {
  case 1:
    console.log("It's Monday!");
    break;
  case 2:
    console.log("It's Tuesday!");
    break;
  // ...cases for other workdays
  default:
    console.log("It's the weekend!");
}
```

For the `default` case, set the text content of `currentDateParagraph` to the value of `formattedDate`.

And with that, your date formatter is complete! You can now format the current date three different ways.