Lesson | Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Arrays and Looping Objectives

Text

In this section, we'll learn about two important programming concepts that will allow us to better work with data:

- **Arrays:** Lists of data or information
- **Looping:** A way of repeating code until a condition is met

We'll learn many different kinds of looping techniques in this section. Then, at the end of the section, we'll review these techniques and learn about when to use each kind of loop.

While we learn how to use arrays and looping in our applications, we'll also learn about Test-Driven Development (TDD), a development technique that focuses on breaking down large problems into a series of smaller steps. With TDD, we'll start writing tests for our business logic, and manually testing the code we write. Specifically, we'll learn the TDD workflow by building out an application called Text Analyzer. In the process, we'll continue to hone our organization skills for the JavaScript we write, applying:

- Separation of business and user interface logic, which we were introduced to in the last section. In this section, we'll continue to practice separation of logic.

- Separation of concerns, which is a key programming concept that dictates that each function in an application should only be responsible for doing one thing. In this context, a 'concern' is a responsibility. For instance, one function might be 'concerned' about one thing (adding two numbers together) while another function might be 'concerned' with returning those numbers to the user.
- **D**on't **R**epeat **Y**ourself, or DRY, another programming concept that dictates that we should avoid repeating code where possible. As we'll learn, DRY code sometimes comes at the expense of code that's separated by concern.

We'll also build on the last section by learning about:

- `Document` methods that return collections of HTML elements, and how to access just one element in the collection.
- Using a code linter as another debugging tool.
- How to use checkboxes in our forms.
- New methods to create and add new HTML elements to our webpage, as well as to remove elements from our webpage.
- Git tools to rewrite our commits.

We'll dedicate the rest of our time to coding and putting into practice everything we've learned so far.

This section also shifts where our attention will be focused. So far, we have been creating websites with HTML, CSS, and (sometimes) Bootstrap. We've learned how to access and manipulate the DOM using Web APIs, and gotten started on the basics of JavaScript. When a project was completed, we ran the code in the browser to display it. In this section, we will focus more on building out JavaScript business logic functions that we can call on in our user interface logic.

At the end of this section, you will be able to:

- Break problems down into simple achievable steps using TDD
- Build JavaScript functions that use arrays and looping

- Create applications with well-separated logic, including many business logic functions that process data.

Remember that speed and quantity are not measurements of success at the end of a day...*understanding* is. While we expect you to review and share each other's work, there's a great saying that applies (with a slight Epicodus modification): *Don't compare your code to the code of others. Compare your code to the code you wrote yesterday*.

Independent Project Objectives

At the end of this section, you will complete an independent project. Your code will be reviewed for the following objectives:

- JavaScript business logic and user interface logic are separate.
- Tests are included for each business logic behavior and code is committed after each test passes.
- Application implements a loop and works as expected.
- The user can use the application repeatedly and see new results.
- Project is in a polished, portfolio-quality state.
- The prompt's required functionality and baseline project requirements are in place by the deadline.

What is a polished, portfolio-quality state?

When a project is both polished and in a portfolio-quality state, this means:

- You've reviewed your project and your README prior to submitting it to make sure there are no errors or missing information and you are consistent in your indentation, spacing, and code structure.
- You are following the best practices and coding conventions we teach. Make sure that your:

- Code is clean, well-refactored, and easy-to-read. This includes correct indentation, spacing, and including only necessary comments and debugging tools.
- Variable names are descriptive and use lower camel case (e.g. `myVariableExample`).
- Commits are made regularly with clear messages that finish the phrase "It will...".

What are the baseline project requirements?

All independent coding projects at Epicodus have these baseline requirements:

- A complete and informative README
  - It is *not* required to include a link to your site hosted on gh-pages, but you are welcome and encouraged to do so!
- The project's commit history demonstrates that the project's required work schedule and hours have been met:
  - 8 hours completed on Friday is required for full-time students
  - 4 hours completed over the weekend is required for part-time students
- Completion of the project based on the prompt *and* objectives. The prompt contains details on the project's theme and features that are not always detailed in the objective. Carefully read through the prompt towards the end of your work session to make sure that you are not missing anything.

Next (/introduction-to-programming/arrays-and-looping/additional-pair-programming-tips)

(http://www.epicodus.com)