Spell It!

In this lesson, we will build an application that challenges children or students of English as a second language to spell words of various things on the web page.

1. Open **06.04-ABCs-Spell-It.html** and preview it in the browser. The interface should load with a set of lettered buttons above a big empty area.

2. Click the **A** button. It should generate a grid of "A-words".

Below each image is a text *input* box. The *placeholder* text says **inglés 英文**, which means "English" in Spanish and Chinese. This is the user's cue to spell the word in English.

3. The first word is *abacus*. Type *abacus* in the box below the picture, and hit Enter. The box will turn green.

4. Click in any other text box, or hit Tab to advance to the next box.

5. Misspell a word, and hit Enter or Tab. The box will turn red.

6. Fix the spelling, and hit Enter (or Tab). The box will turn green.

7. Click the **B** button to clear away the A-words and load the B-words.

previewing the html file in the code editor

8. Have a look at the HTML code. As we can see, none of the ABC's content appears. That's because, as with the Chinese Zodiac Animals application, the interface is generated dynamically, with JS:

```
<div class="container">
    <header>
        <h1>
            Spell words in English:<br>
            Escribe palabras en inglés:<br>
            拼写以字母开头的单词 :
        </h1>
        <div class="buttons-box"></div>
    </header>
    <main></main>
</div>
```

The **main** element is empty on page load. That is because its content is generated by a function which is called when the user clicks a letter button. The function contains a *for loop* in which the picture boxes for the chosen letter are produced.

the data file

9. Open **abc-data.js**, the dataset for the application. It contains a single variable, **abcs**, which is an array of objects. Each object but the first represents a word:

```
    let abcs = [

['a','b','c','d','e','f','g','h','k','l','m','n','o','p','r','s','t','v'],
        { word: "abacus", level: 3, keywords: ["Chinese", "calculator"],
alive: false },
        -- ETC.  --
        { word: "vinegar", level: 2, keywords: ["food", "liquid",
"cupboard", "meal"], alive: false }
    ];
```

The first item in **abcs** is likewise an array--of letters. If the letter is in the array, there's a button for it.

10. Remove the **'a'** from the array.

11. Save the file and reload the browser. The A-button is gone.

12. Restore the **'a'** and reload. The "A-button" is back.

13. Add **"z"** to the array of letters.

14.Reload the browser. There should be a "Z-button" now. Clicking it doesn't yield any "Z-words", though, since there are none in the dataset.

15. Remove the **"z"** from the array and reload the browser; the "Z-button" is gone.

array objects

All other items in the array are **objects**, arranged alphabetically by their **word** property, the value of which is a string ("alligator", "anvil", etc). These values are used to:

- concatenate the file name of the image to be displayed.
- assign an id to each text input box. If the text entered by the user matches the id of that text box, the spelling is correct.

Each object has three other properties: **level**, **keywords** and **alive**, but these are not used in the current exercise.

16. Switch to using the **START.js** file and reload the browser. The app no longer works, of course. All we see is the static html and css.

17. Get the elements that hold the interface:

- **buttons-box** div holds the lettered buttons

- **main** holds the grid of wrod-picture boxes Neither of these divs has an id, so use **querySelector**:

```
const buttonsBox = document.querySelector('.buttons-box');
const main = document.querySelector('main');
```

Next we'll make the lettered buttons, which exist as an array that is itself the first item in the **abcs** array, and so is accessed by the syntax **abcs[0]**.

18. Save the first item in the **abcs** array as its own array variable:

```
const letters = abcs[0];
```

19. Loop through the array of **letters**:

```
for(let i = 0; i < letters.length; i++) {
}
```

20. Make a button object each time through the loop:

```
for(let i = 0; i < letters.length; i++) {
    let btn = document.createElement('button');
}
```

21. Assign the **letter-btn** class to the button:

```
    btn.className = 'letter-btn';
```

22. Assign an id to the button. This is the letter of the button, so the "A button" is has **id='a'**, and so on. We get the current letter by using **i** to look up that item in the letters array by index.

```
    btn.id = letters[i];
```

23. Label the button by setting its **textContent** property to the current letter, to uppercase:

```
    btn.textContent = letters[i].toUpperCase();
```

24. Have the button call the function **makeWordBoxes**:

```
        btn.addEventListener('click', makeWordBoxes);
```

25. Output the button to the parent element div, **buttonsBox**:

```
        buttonsBox.appendChild(btn);
```

26. Declare the **makeWordBoxes** function. For openers, just have it log the id of the button, as **this.id**. Remember: inside a function **this** is the thing that called the function, in this case the button:

```
    function makeWordBoxes() {
        console.log(this.id);
    }
```

27. Reload the browser, click a letter button, and check the console. The button's id should appear.

28. Now for the actual code. Start by clearing **main** in case any lettered buttons are already there.

```
    function makeWordBoxes() {
        console.log(this.id);
        main.innerHTML = "";
    }
```

29. Loop the **abcs** array. The counter **i** start at 1, since we want to skip the array of letters at index 0. We want the loop to only iterate over the word objects:

```
    function makeWordBoxes() {
        console.log(this.id);
        main.innerHTML = "";
        for(let i = 1; i < abcs.length; i++) {
        }
    }
```

30. In the loop, simplify the code by saving the current word object as **word**.

```
        for(let i = 1; i < abcs.length; i++) {
            let word = abcs[i].word;
        }
```

If the "A-button" was clicked, we want A-words *only* ("abacus", "actor", etc.). Such filtering requires if-logic:

31. Check if the first letter of the word (**word[0]**) matches the id of the clicked button (**this.id**):

```
    for(let i = 1; i < abcs.length; i++) {
        let word = abcs[i].word;
        if(word[0] == this.id) {
        }
    }
```

32. If the condition returns true, make a "word box", and assign it the divvy class:

```
        if(word[0] == this.id) {
            let wordBox = document.createElement('div');
            wordBox.className = "divvy";
        }
```

33. Output the **wordBox** div to **main**:

```
    if(word[0] == this.id) {
        let wordBox = document.createElement('div');
        wordBox.className = "divvy";
        main.appendChild(wordBox);
    }
```

making images to go inside the word box divs

34. Still inside the if statement, make a new image object:

```
        let pic = new Image();
```

35. Concatenate the image's **src** property. The folder names are the letter to uppercase, followed by a lowercase "s": letters **As**, etc.:

```
        pic.src = 'img/ABCs/' + word[0].toUpperCase() + 's/' + word +
 '.png';
```

36. Ouput the image to the word box div:

```
        wordBox.appendChild(pic);
```

37. Still in the if statement, make an **input** object of type **search**:

```
    let inputBox = document.createElement('input');
    inputBox.type = "search";
```

38. Assign the input box a **placeholder** that says *English* in Spanish and Chinese. This prompts the user to spell the word:

```
    inputBox.placeholder = "ingl\u00E9s 英文";
```

The input box call the function **checkSpelling** when the user hits Enter or Tab, which fire the **search** and **blur** events, respectively:

39. Add the two listeners to the input box.

```
        inputBox.addEventListener('search', checkSpelling);
        inputBox.addEventListener('blur', checkSpelling);
```

40. Assign **word** as the id to the input box. With this, we will be able to check spelling by comparing the id of the box to what the user typed into the box:

```
        inputBox.id = word;
```

41. Output the input box to the div:

```
        wordBox.appendChild(inputBox);
```

42. Define the **checkSpelling** function:

```
    function checkSpelling() {
    }
```

43. Compare **this.value** and **this.id**. If they are equal, the spelling is correct, so set the input box background to green and its text to white:

```
    function checkSpelling() {
        if(this.value == this.id) {
            this.style.backgroundColor = '#181';
            this.style.color = '#fff';
        }
    }
```

44. Else, the spelling is wrong, so set the input background to red:

```
        } else {
            this.style.backgroundColor = '#811';
            this.style.color = '#fff';
        }
```

45. Save all and take it for a spin. Buttons should load the image packs for their respective letters. Text boxes should turn green or red when you enter text and hit Enter or Tab.

- **END: Lesson 06.04**
- **NEXT: Lesson 06.05**