Chinese Zodiac Animals, Part 1

making DOM elements dynamically

createElement(), appendChild()

new Image()

Chinese Zodiac Animals

In this first of two lessons, we will begin to make an educational interface called **Chinese Zodiac Animals**. It is for learning the Chinese names of the 12 animals of the Chinese zodiac. The interface is equally suitable for students of English as a foreign language to learn the English names of these dozen animals.

From a coding standpoint, we will learn how to create DOM elements dynamically, as the interface is *not* hard-coded in the HTML. It is all made dynamically with JS.

1. Open **06.01-Creating-Elements.html**, and have a look at the code. There is a **container** which is practically empty: just an h1 and an empty **section**:

```
<div class="container">
    <h1>Lesson 06.01: Creating Elements – Chinese Zodiac Animals</h1>
    <section></section>
</div>
```

2. Preview the page in the browser. The HTML file is using the next lesson's final JS, **06.02-Chinese-Zodiac-Animals-FINAL.js**. This gives us a sneak preview of the completed interface, which will require this and the next lesson to code from scratch.

The completed interface consists of 12 divs, one for each animal of the Chinese zodiac. Each div contains a myriad of animal info, but in this lesson all we will be making are:

- the animal images (chicken, cow, dog, dragon, etc.)
- the input boxes for spelling the animal in English or piynin

Adding the Chinese characters and years, implementing the audio and displaying animal names will be left to **Lesson 06.02**.

3. Click the sound icon to play the **mp3** for that animal. You should hear the animal name spoken in English and Chinese. We will leave until the next lesson the implementation of the **Audio** object.

4. Try out the input box:

- type **chicken** in the chicken's text box and hit Enter. The box turns green and the English and pinyin appear.
- mess up the spelling and hit Enter again; the box turns red. - type **ji** and hit Enter. The box turns green, because *ji* is the pinyin for *chicken*.

5. Open the **images/animals** folder. The file naming format is **"english.jpg"**, so: chicken.jpg, cow.jpg, dog.jpg, etc.

6. The HTML file that **js/zodiac-animals-data.js** is already being imported. Open the file. It contains an **animals** array of 12 objects:

```
    let animals = [
        {eng: "chicken", also: 'rooster', chi: "ji", pin: "ji&#772;",
year: 2029, tone: 1},
        -- ETC --
        {eng: "tiger", chi: "hu", also: 'laohu', pin: "la&#780;ohu&#780;",
year: 2022, tone: 3}
    ];
```

animals object properties

The 12 objects are ordered alphabetically by their **eng** (English) property, from "chicken" to "tiger". Each object is an animal with six properties. We will be using all of these properties to build our interface, although in this lesson we will only be using the **eng** property.

What's pinyin?

The Chinese language does not use an alphabet. Instead, each word-syllable exists as a *character*. Even so, Chinese pronunciation can be spelled out in a Romanization system called **pinyin**. In our application, the text input can be English or pinyin. So, the English *chicken* and the pinyin *ji* are both acceptable spellings.

7. Open **06.01-Creating-Elements.css** and have a look. There are classes that we will apply to the dynamically generated DOM elements.

8. Switch the html page to using **START .js** and reload the page. Now the **section** element is just a big, empty box.

how to make HTML elements dynamically with JS

There are two ways to make an element:

- **createElement('div')** method creates an object, in this case a div
- **new Image()** creates an object, in this case an image
- the object needs to be saved to a variable, preferably a **const** to prevent it from having its data type mutated:
    - **const divvy = createElement('div')** makes a div object named **divvy**
    - **const pic = new Image()** makes an image object named **pic**
- besides Image, Audio, Video and a few other elements, all elements are made with **createElement()**

- the
- once created, elements can have any attribute applied as a property:
    - **divvy.id = 'div1'** assigns divvy an id of div1.
    - **divvy.className = 'my-div'** assigns divvy a class of my-div.
    - **image.src = 'images/dog.jpg'** sets the image source to dog.jpg.
- **appendClild()** is used to output the dynamically generated object to the DOM. The method is called on an existing DOM element and takes the newly created element as its argument:
    - **section.appendChild(divvy)** outputs **divvy** as a child element of **section**

Now let's start to actually make the Chinese Zodiac Animals interface.

9. Open **06.01-Creating-Elements-START.js** and get the **section** that serves as the parent wrapper of the interface:

```
const section = document.querySelector('section');
```

10. Make a div with **createElement()**:

```
const divvy = document.createElement('div');
```

11. Apply the existing **.divvy** class to the div and Give the div some text content, as a test:

```
divvy.className = 'divvy';
divvy.textContent = 'dynamic div';
```

12. Use **appendChild()** to output the div inside the section:

```
section.appendChild(divvy);
```

13. Reload the page. In the upper left corner of the section there should be one smallish box that says *dynamic div*.

making the set of 12 Chinese Zodiac Animals

Now that we have that working, let's use a loop to make all 12 divs.

14. Wrap the code that made the div in a for loop that runs 12 times. For text output, number the divs with the counter, **i**:

```
for(let i = 0; i < 12; i++) {
    const divvy = document.createElement('div');
    divvy.className = 'divvy';
    divvy.textContent = 'div ' + i;
```

```
        section.appendChild(divvy);
    } // end loop
```

15. Reload the page. All 12 divs should appear, in two rows of six. The divvy class has *display: inline-block* to arrange the divs side-by-side.

16. Still in the loop, make an image for each div, using **new Image()**:

```
        let pic = new Image();
    } // end loop
```

17. Set the source of the image to **cow.jpg**. The result will be 12 cows:

```
        let pic = new Image();
        pic.src = 'images/animals/cow.jpg';
```

18. Apply the **.animal-pic** class to the image:

```
        let pic = new Image();
        pic.src = 'images/animals/cow.jpg';
        pic.className = 'animal-pic';
```

19. Delete the text and output the image inside the div. In terms of nesting, **divvy** goes inside **section** and **pic** goes inside **divvy**:

```
        pic.className = 'animal-pic';
        divvy.textContent = '';
        divvy.appendChild(pic);
    } // end loop
```

20. Reload the page. We should have 12 divs, each with the cow.

21. Change the source to be dynamic, so that we get all 12 animals--not just the cow. Concatenate the **eng** property into the file path. The current animal is available as **animals[i]** and its English name is **animals[i].eng**:

```
        let pic = new Image();
        pic.src = `images/animals/${animals[i].eng}.jpg`;
```

22. Reload the page. We should have 12 divs, each with a different animal.

23. We will be using the **animals** array a lot, so to simplify property access, save the current animal **animals[i]** to a variable, **animal**. Update the concatenation to use the more streamlined **animal.eng**:

```
        let animal = animals[i];
        let pic = new Image();
        pic.src = `images/animals/${animal.eng}.jpg`;
```

24. Still in the loop, make the **input** box:

```
        const inputBox = document.createElement('input');
```

25. Assign the input box a type of **search**. This will give the box a little X to clear it, and it also provides a **search** event, which fires when the user hits Enter:

```
        inputBox.type = 'search';
```

26. Assign the box a **placeholder** to prompt the user to enter the animal name:

```
        inputBox.placeholder = 'name';
```

27. Assign **eng**, **chi** and **also** properties to the input box. These will be used by the **checkSpelling** function to see if the user input matches any of the three accepted spellings:

```
        inputBox.eng = animal.eng;
        inputBox.chi = animal.chi;
        inputBox.also = animal.also;
```

28. Also save the current index as a property of the input box. This will come in handy down the road (next lesson):

```
        inputBox.also = animal.also;
        inputBox.i = i;
```

29. Have the input box call the function when its search event is fired:

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

30. Also the input box call the function on **blur**, and event which fires when the user hits **Tab** to leave an input box. Tab moves the cursor to the next input box, so it's a handy way to navigate from one animal div to the next:

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

31. Output the input box to the div. It will appear under the animal image:

```
        divvy.appendChild(inputBox);
    } // end loop
```

We need to make the **checkSpelling** function before reloading the page in the browser. If the function referred to in the listener does not exist at that point, we'll get an error.

this keyword

Inside a function, the **this** keyword refers to the object that called the function. In the **checkSpelling** function, **this** is the input box:

- **this.eng** is the English name
- **this.chi** is the pinyin spelling (without tone markings)
- **this.also** is the alternate spelling

With these properties, we compare the user input to the correct spelling:

32. Define the **checkSpelling** function and start by getting the **value** of the input element--whatever the user typed into the box:

```
    function checkSpelling() {
        let input = this.value;
    }
```

33. Compare the user input to the English, pinyin and alternate (also) spelling. The user input needs to match *one* of the three correct spellings:

```
    function checkSpelling() {
        let input = this.value;
        if(input == this.eng || input == this.chi || input == this.also) {
        }
    }
```

34. If the user input is correct, turn the input box green; else turn the box red:

```
    function checkSpelling() {
        let input = this.value;
        if(input == this.eng || input == this.chi || input == this.also) {
            this.style.backgroundColor = '#0B0';
            this.style.color = '#fff';
        } else {
            this.style.backgroundColor = '#921';
            this.style.color = '#fff';
        }
    }
```

35. Reload the page. Each box should have an animal pic and an input box.

36. Enter a spelling in a box and hit Enter.

- if the spelling is correct, the input box turns green.
- else, the spelling is incorret, so the box turns red.
- **END: Lesson 06.01**
- **NEXT: Lesson 06.02**