setTimeout(function()):

setInterval(function()):

Calling a Function on a Delay with a Timer

setTimeout(function()): calling a function once on a timer

The **setTimeout()** method executes a callback function on a time delay.

- the **setTimeout()** method takes two arguments:
  - an inline, anonymous callback function
  - a delay in milliseconds before the function runs
- as a global **window** method, it can optionally be invoked as **window.setTimeout()**
- the **setTimeout()** runs only once

1. Call the **setTimeout()** method:

```
setTimeout();
```

2. Pass in the first argument: the anonymous, inline callback function:

```
setTimeout(function() {
});
```

3. Write some code for the function. We'll just console.log a greeting:

```
setTimeout(function() {
  console.log("Hello World!");
});
```

4. Pass in the second argument: a five-second delay (5000 ms) before the function runs:

```
setTimeout(function() {
    console.log("Hello World!");
}, 5000);
```

5. Refresh the browser. The greeting should appear after five seconds.
6. Run another setTimeout, but with the more concise arrow function syntax. Increase the delay, and change the message:

```
setTimeout(() => console.log("Hello World, again!"), 7500);
```

7. The callback can also call a function. Write a function, and then call it from inside the setTimeout callback. Again, increase the delay and change the message:

```
function greetWorld() {
    console.log("Hello World one more time!");
}

setTimeout(() => greetWorld(), 10000);
```

8. Write another function to call from the setTimeout callback, only this time give the function a parameter:

```
function greetUser(user) {
    console.log("Hello " + user + "!");
}

setTimeout(() => greetUser("Brian"), 12500);
```

9. Try a set timeout with output to the web page. Get the h2 tag where the output is to appear. The **querySelector('h2')** method gets the first h2 element, which is what we want:

```
const h2 = document.querySelector('h2');
```

10. On a 7-second delay, output a random integer from 1-100:

```
setTimeout(() => {
    h2.textContent = Math.ceil(Math.random() * 100);
}, 7000);
```

setInterval(function()): calling a function repeatedly on a timer

The **setInterval()** method executes a callback function repeatedly on a time delay.

- the **setInterval()** method takes two arguments:

    - an inline, anonymous callback function
    - a delay in milliseconds between function calls
- as a global **window** method, it can optionally be invoked as **window.setInterval()**
- the **setInterval()** runs repeatedly
- to stop the **setInterval()** use **clearInterval()**

11. Call the **setInterval()** method, and pass it its callback function:

```
setInterval(function() {
});
```

12. Have the callback output a random integer from 1-100 every 3 seconds:

```
setInterval(function() {
  h2.textContent = Math.ceil(Math.random() * 100);
}, 3000);
```

stopping setInterval() with clearInterval()
To stop the setInterval callback functino from running, use the **clearInterval()** method:

- set the **setInverval()** equal to a variable
- pass the variable to **clearInterval()**
- clearInterval() can be invoked on a counter or by a DOM event, such as a STOP button click

13. Above the **setInterval()**, declare a counter variable, **i**, and inside the callback, increment the counter with **i++**:

```
let i = 0;

setInterval(function() {
  i++;
  h2.textContent = Math.ceil(Math.random() * 100);
}, 3000);
```

14. Convert the callback to an arrow function. We will try to use arrow syntax for all inline anonymous functions, from now on:

```
let i = 0;

setInterval(() => {
  i++;
  h2.textContent = Math.ceil(Math.random() * 100);
}, 3000);
```

15. Wrap the output line in an if-statement that only runs if **i** is less than or equal to 10:

```
setInterval(() => {
  i++;
  if(i <= 10) {
    h2.textContent = Math.ceil(Math.random() * 100);
  }
}, 3000);
```

The if-statement will stop the output, but not the callback, which will keep running and incrementing the counter. To stop the callback function, we need to call the **clearInterval()** method.

16. Set the **setInterval()** method equal to a variable:

```
let intrvl = setInterval(() => {
  i++;
  if(i <= 10) {
    h2.textContent = Math.ceil(Math.random() * 100);
  }
}, 3000);
```

17. Add an **else** part that calls **myInterval()**, passing it the **intrvl** object:

```
let intrvl = setInterval(() => {
  i++;
  if(i <= 10) {
    h2.textContent = Math.ceil(Math.random() * 100);
  } else {
    clearInterval(intrvl);
  }
}, 3000);
```

setInverval() for choosing items from an array at random on a timer

In this next example, we start with an array of 13 animals. The **setInterval()** callback function will run 13 times, choosing an animal at random from the array each time

```
  const animals = ['aardvark', 'bunny', 'cheetah', 'deer', 'elephant',
'frog', 'giraffe', 'hippo', 'iguana', 'jaguar', 'kangaroo', 'lion',
'moose'];
```

18. Set up the **setInterval()** method with a callback that runs every 2 seconds:

```
setInterval(() => {
}, 2000);
```

19. Generate a random integer from 0-12, which is in the range of the 13-item **animals** array:

```
setInterval(() => {
  let r = Math.floor(Math.random() * animals.length);
});
```

20. Get the other h2, the one with id of **animals**:

```
let animalsH2 = document.getElementById('animals');
```

21. Output the animal whose index is the random number:

```
setInterval(() => {
  let r = Math.floor(Math.random() * animals.length);
  animalsH2.textContent = animals[r];
});
```

22. Specify a timer delay of 2 seconds, and check the browser:

```
setInterval(() => {
  let r = Math.floor(Math.random() * animals.length);
  animalsH2.textContent = animals[r];
}, 2000);
```

23. It should work, but the interval has no way of stopping yet. Set the method equal to a variable:

```
let myInterval = setInterval(() => {
  let r = Math.floor(Math.random() * animals.length);
  animalsH2.textContent = animals[r];
}, 2000);
```

24. Add a counter and increment it inside the callback function:

```
let counter = 0;
let myInterval = setInterval(() => {
  counter++;
  let r = Math.floor(Math.random() * animals.length);
```

```
  animalsH2.textContent = animals[r];
}, 2000);
```

25. Add an if-else-statement, so that the interval is cleared after it has run 13 times, wthe length of the **animals** array:

```
let counter = 0;
let myInterval = setInterval(() => {
  counter++;
  if(counter < animals.length) {
    let r = Math.floor(Math.random() * animals.length);
    animalsH2.textContent = animals[r];
  } else {
    clearInterval(myInterval);
  }
}, 2000);
```

avoiding repeats when choosing array items at random

That should work, but better might be to not repeat animals. The most efficent way to avoid repeats is to remove the item from the array after it has been chosen. This is done with the **splice()** method, which takes two arguments: a start index and the number or items to remove:

26. Use the **splice()** method to remove the chosen item:

```
let counter = 0;
let myInterval = setInterval(() => {
  counter++;
  if(counter < animals.length) {
    let r = Math.floor(Math.random() * animals.length);
    animalsH2.textContent = animals[r];
    animals.splice(r, 1);
  } else {
    clearInterval(myInterval);
  }
}, 2000);
```

The animals are being removed, but the process ends prematurely, once the incrmenting counter variable equals the length of the dwindling array. The answer is to pass **animals.length** to the if-statement. Zero is a **falsey** value, so once **animals.length** reaches 0, the **else** part will run, which clears the interval.

26. Remove the counter everywhere, with **animals.length** becoming the new condition to evaluate. In the ouput, switch to **+=**, so that we see all the animals appear. In this way we will know if our algorithm and code are working:

```
let myInterval = setInterval(() => {
  if(animals.length) { // 0 is falsey
```

```
      // if(animals.length > 0) { // also works
      let r = Math.floor(Math.random() * animals.length);
      animalsH2.textContent += animals[r] + " ";
      animals.splice(r, 1);
    } else {
      clearInterval(myInterval);
    }
  }, 2000);
```

27. Refresh the browser. All 13 animals should output, one at a time, in random order.

28. To see how the random number is always within the range of the dwindling array length, include the array length and the random number in the output. Add a
    tag to separate each animal on its own line. The inclusion of html in the string requires us to switch from **textContent** to **innerHTML**:

29. To visualize how the random number is always within the range of the dwindling array length, output the array length as well as the random number in the output for each animal.
    Add a
    tag to separate each animal on its own line.
    The inclusion of html in the string requires switching from **textContent** to **innerHTML**.
    The output is too much for concatenation, so use backticks:

```
  let myInterval = setInterval(() => {
      if(animals.length > 0) {
          let r = Math.floor(Math.random() * animals.length);
          animalsH2.innerHTML += animals[r] + "  - rand num: " + r + ";
array len: " + animals.length;
          animals.splice(r, 1);
      } else {
          animalsH2.innerHTML += " done!";
          clearInterval(myInterval);
      }
  }, 2000);
```

END Lesson 08.01
NEXT: Lesson 08.02