1 `for` Loop vs `forEach()`

1.1 `for` Loop

In programming, it is common to use “i” as an iterator. With nested loops, it is common to use “i”, “j”, “k”, “l”, “m”, etc...

If you’re coming from a Python background, you will know that Python has syntactical sugar to make loops more symantic (eg `for fruit in fruits:`). JavaScript doesn’t.

Setting up a for loop in JavaScript:

```
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

1.2 `forEach()`

Arrays are often named using plural nouns, like "fruits" because they are a collection of those things. This keeps our code more *symantic* and easier to understand. It makes sense then, when iterating through an array, to use arguments that represent a singular thing in a given array. Thus, adding symantic meaning to our `forEach()` loop.

```
fruits.forEach(fruit => {
  console.log(fruit)
});
```

JavaScript allows us to write single-line functions all on one line and ommit the curlies. Allowing us to write the above more concisely:

```
fruits.forEach(fruit => console.log(fruit));
```

Question:

What is the "=>" ?

```
() => {};

\\ vs

function() {};
```

Is there a difference?

1.2.1 The `forEach()` Method

Pros:

- Easier to read and understand
- Bugs easier to avoid
  - Infinite loops impossible
  - Avoids incrementing mistakes

    - Wrong condition

Cons:

- Can’t break out early
    - Rare that you’d need to, but use for or while if you do