Adding Items (Arrays)

Arrays

As you probably know by now, an array is a data type used to store collections of data in indexes. You can imagine it as a bookshelf where every shelf is an index, but one key difference is that indexes in arrays start from `0` not `1`.

Creating an array is quite simple, we just need to use the brackets (`[]`) symbol and wrap values in it like so:

```
const fruits = ["Apple", "Orange", "Banana"]
```

Notice that each value is separated by a comma. Values in an array can be of different data types, however it is very recommended to use only one data type for one array since it will be much easier to predict the values' type, therefore making it much easier to manage.

Adding Items

There are actually many ways to add items to an array, but we'll only talk about the 2 main ways of doing it here.

Push

Arrays have a method called `.push` to "push" or insert items to the end of an array. When a value is pushed into an array, it will automatically mutate (update) the array which means we don't need to re-assign the array into a variable.

The following code snippet is a wrong example of using `.push`. We don't need to re-assign `fruits` with `fruits.push`

```
// Wrong
let fruits = ["Apple", "Orange", "Banana"]
fruits = fruits.push("Pineapple")
```

This is the correct way of using `.push`.

```
// Correct
let fruits = ["Apple", "Orange", "Banana"]
fruits.push("Pineapple")
```

Assigning Indexes

To add items, we can also directly assign a value to an index. Let's say we have an array with 3 items and we want to add a new item without using the `.push` method. What we can do is just directly assign index 3 (since indexes 0, 1, and 2 are occupied) a value.

```
let fruits = ["Apple", "Orange", "Banana"]
fruits[3] = "Pineapple"
```

However sometimes we don't actually know which index we should be using, and it will also be fatal if we assign a value to an occupied index, overwriting the previous value. What we can do here is use the array's length to get the nearest non-occupied index.

```
let fruits = ["Apple", "Orange", "Banana"]
// fruits.length = 3
fruits[fruits.length] = "Pineapple"
```

Here's a bit of an explanation if you are still confused. `fruits` is an array containing 3 items, which means its array length is **3** and the currently occupied indexes are 0, 1, and 2. Now when we want to add an item to the array, we need to use index number 3, and we can get the number 3 from the array's length. After adding the item, the array's length turns into 4, and the occupied indexes are now 0, 1, 2, and 3. Now we can repeat the process of taking the array's length (4), and assigning a value according to the array's length.

Reserving Indexes

There is actually an interesting interaction when we create an empty array, and assign let's say index number 5 a value while ignoring the other previous indexes. Here's the code snippet if the explanation is unclear

```
let myArr = []
myArr[5] = "Hello"
```

We now know that index 5 has a value of `"Hello"`, the question is, since the other indexes have not been assigned values, what is the length of the array, and what are the values of the previous indexes?

The **length of the array is now actually 6**, and the **value of indexes 0 through 4** is actually `undefined`. So technically, the indexes do have, in fact, their own values. When we assign an index a value while skipping over the previous indexes, Javascript actually reserves these indexes with `undefined` values and then we can use the array normally.