the shortcut

Fullstack Web Development Tutorial Lesson 8

Today’s lesson will cover

- Array methods

JavaScript fundamentals

Array methods

- To recap, methods that add and remove items from the beginning or the end:
  - `arr.push(...items)` – adds items to the end,
  - `arr.pop()` – extracts an item from the end,
  - `arr.shift()` – extracts an item from the beginning,
  - `arr.unshift(...items)` – adds items to the beginning.
- Deleting an element possible with object `delete` operator but the removed element index position is still occupied which can be checked with `arr.length`
  - Hence, special methods should be used

Array methods: splice

- The `arr.splice(start)` method is a swiss army knife for arrays. It can do everything: insert, remove and replace elements
- Syntax:
  - `arr.splice(index[, deleteCount, elem1, ..., elemN])`
  - It starts from the position `index`: removes `deleteCount` elements and then inserts `elem1, ..., elemN` at their place. Returns the array of removed elements.
- Negative indexes allowed for array methods

Array methods: slice

- For copying, it's similar to a string method `str.slice`, but instead of substrings it makes subarray
- Syntax:
  - `arr.slice([start], [end])`
  - It returns a new array copying to it all items from index `start` to `end` (not including `end`). Both `start` and `end` can be negative, in that case position from array end is assumed.

Array methods: concat

- The method arr.concat creates a new array that includes values from other arrays and additional items.
- Syntax:
  - `arr.concat(arg1, arg2...)`
- It accepts any number of arguments – either arrays or values.
- The result is a new array containing items from `arr`, then `arg1`, `arg2` etc.
- If an argument `argN` is an array, then all its elements are copied. Otherwise, the argument itself is copied.
- Normally, it only copies elements from arrays. Other objects, even if they look like arrays, are added as a whole.

Array iterate: forEach

- The arr.forEach method allows to run a function for every element of the array
- Syntax:
  - ```
    arr.forEach(function(item, index, array) {
      // ... do something with item
    });
    ```
- The result of the function (if it returns any) is thrown away and ignored.
    ```

Searching in array: indexOf/lastIndexOf and includes

- The methods arr.indexOf, arr.lastIndexOf and arr.includes have the same syntax and do essentially the same as their string counterparts, but operate on items instead of characters:
  - `arr.indexOf(item, from)` – looks for `item` starting from index `from`, and returns the index where it was found, otherwise `-1`.
  - `arr.lastIndexOf(item, from)` – same, but looks for from right to left.
  - `arr.includes(item, from)` – looks for `item` starting from index `from`, returns `true` if found.
- If we want to check for inclusion, and don't want to know the exact index, then `arr.includes` is preferred.

Searching in array: find and findIndex

- Used to find object with specific condition
- Syntax:

```
let result = arr.find(function(item, index, array) {

  // if true is returned, item is returned and iteration is stopped

  // for falsy scenario returns undefined

});
```

- The function is called for elements of the array, one after another:
  a. `item` is the element.
  b. `index` is its index.
  c. `array` is the array itself.

Searching in array: filter

- The syntax is similar to `find`, but `filter` returns an array of all matching elements
- Syntax:

```
let results = arr.filter(function(item, index, array) {

  // if true item is pushed to results and the iteration continues

  // returns empty array if nothing found

});
```

Exercise

- Write functions `rangeFruits(fruits, a, b)` and `rangeNumbers(numbers, a, b)` that gets an array, looks for elements between `a` and `b` in it and returns an array of them.
- The function should not modify the array. It should return the new array on the console.
- Use filter method to return new array, which filters:
    - Array `numbers = [3, 2, 4, 5, 8, 9, 1]` and returns array `smallNumbers` with value ranging between 1 to 5
    - Array `fruits = ["apple", "banana", "cantaloupe", "durian", "jackfruit"]` and returns array `bigFruits` filtering string starting with “c” to “z”
- Return the `smallNumbers` and `bigFruits` arrays on the console

Transform array: map

- calls the function for each element of the array and returns the array of results.
- Syntax:

```
let result = arr.map(function(item, index, array) {

  // returns the new value instead of item

});
```

Transform array: sort

- The call to arr.sort() sorts the array *in place*, changing its element order.
- It also returns the sorted array, but the returned value is usually ignored, as `arr` itself is modified.
- **The items are sorted as strings by default.**

Exercise

- We have an array of strings `webTech`. We'd like to have a sorted copy of it, but keep `webTech` unmodified.
- Create a function `copySorted(webTech)` that returns such a copy in `orderedWebTech` array.
- ```
  let webTech = ["HTML", "JavaScript", "CSS"];

  // Your copySorted(webTech) function

  let orderedWebTech= copySorted(webTech);

  console.log( orderedWebTech); // CSS, HTML, JavaScript

  console.log( webTech ); // HTML, JavaScript, CSS (no changes)
  ```
  ```

Exercise

- Write the function `sortByAge(users)` that gets an array of objects with the `age` property and sorts them by `age`.
- ```
  let john = { name: "John", age: 25 };
  let pete = { name: "Pete", age: 30 };
  let mary = { name: "Mary", age: 28 };
  let users = [ pete, john, mary ];
  sortByAge(arr);
  ```
- Now create an array `names` which contains the name properties of each objects in newly sorted `users` array and show the `names` array on the console
  ```

Transform array: split and join

- `split` Splits the string into an array by the given delimiter `delim`.
- Syntax:

```
str.split([separator[, limit]])
```

- `join` method creates and returns a new string by concatenating all of the elements in an array
- Syntax:

```
arr.join([separator])
```

Transform array: reduce/reduceRight

- Used to calculate a single value based on the array.
- Syntax:

```
let value = arr.reduce(function(accumulator, item, index, array) {
  // …
}, [initial]);
```

- The function is applied to all array elements one after another and “carries on” its result to the next call.
- Arguments:
  - `accumulator` – is the result of the previous function call, equals `initial` the first time (if `initial` is provided).
  - `item` – is the current array item.
  - `index` – is its position.
  - `array` – is the array.

Summary: Cheat sheet of array of methods you will mostly use

- To add/remove elements:
  - `push(...items)` – adds items to the end,
  - `pop()` – extracts an item from the end,
  - `shift()` – extracts an item from the beginning,
  - `unshift(...items)` – adds items to the beginning.
  - `splice(pos, deleteCount, ...items)` – at index `pos` delete `deleteCount` elements and insert `items`.
  - `slice(start, end)` – creates a new array, copies elements from position `start` till `end` (not inclusive) into it.
  - `concat(...items)` – returns a new array: copies all members of the current one and adds `items` to it. If any of `items` is an array, then its elements are taken.
- To search among elements:
  - `indexOf/lastIndexOf(item, pos)` – look for `item` starting from position `pos`, return the index or `-1` if not found.
  - `includes(value)` – returns `true` if the array has `value`, otherwise `false`.
  - `find/filter(func)` – filter elements through the function, return first/all values that make it return `true`.
  - `findIndex` is like `find`, but returns the index instead of a value.

Summary: Cheat sheet of array of methods you will mostly use

- To iterate over elements:
  - `forEach(func)` – calls `func` for every element, does not return anything.
- To transform the array:
  - `map(func)` – creates a new array from results of calling `func` for every element.
  - `sort(func)` – sorts the array in-place, then returns it.
  - `reverse()` – reverses the array in-place, then returns it.
  - `split/join` – convert a string to array and back.
  - `reduce(func, initial)` – calculate a single value over the array by calling `func` for each element and passing an intermediate result between the calls.

the shortcut

Self Study Assignments

To Dos

- Don’t get stuck in tutorial purgatory as that’ll prolong the journey for no reason. Dive right into your own project. Reach out to me if you need help coming up with project ideas and to any of the mentors when executing them.
- Complete the feedback form before Friday June 12, 2020 one to one session
- Continue freecodecamp Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- If you need help pushing your HTML CSS project on GIthub and using Github pages let me know right away.