JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA


HOMEWORK

Functional programming with arrays

Declarative vs Imperative

- Imperative - how we want to do something
- Declarative - what we want to do
- Read more: https://ui.dev/imperative-vs-declarative-programming

Pure functions

- For same input (argument) functions will return same output
- No side effects

pure-functions.js

```
// Pure functions

function addNumbers(x, y) {
    return x + y;
}

console.log(addNumbers(3, 5)); // 8

```

Array forEach

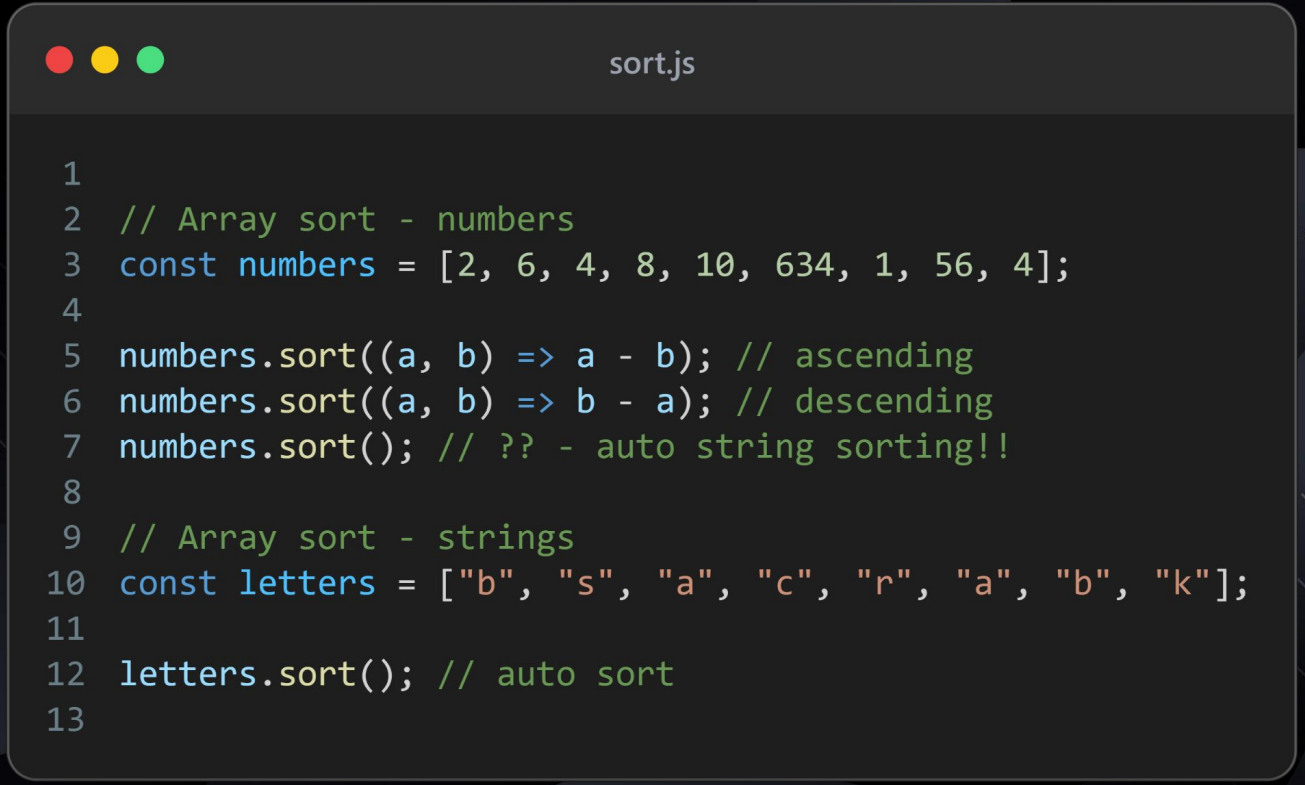
- It’s a built-in function for looping over an array
- Accepts a callback (function) as an argument
- Runs the functions for each element
- Return nothing (undefined)

forEach.js

```
1
2  // Array forEach
3  const userNames = ["Bob", "Frank", "Jimmy"];
4
5  //Imperative
6  for (let i = 0; i < userNames.length; i++) {
7    console.log('Welcome ', userNames[i]);
8  }
9
10 //Declarative
11 userNames.forEach(user => console.log('Welcome ' + user));
12
13 // Welcome Bob
14 // Welcome Frank
15 // Welcome Jimmy
16
```

Array sort

- Accepts a comparing function as an argument
- This function will automatically be assigned 2 arguments - 2 elements of the array (a and b)
- Expects the function (callback) to return a number
- If returned number is > 0 - sort **a after b**
- If returned number is < 0 - sort **b after a**
- If returned number is 0 - **a and b are equal**
- **Changes original array!!**


sort.js
// Array sort - numbers
const numbers = [2, 6, 4, 8, 10, 634, 1, 56, 4];
numbers.sort((a, b) => a - b); // ascending
numbers.sort((a, b) => b - a); // descending
numbers.sort(); // ?? - auto string sorting!!
// Array sort - strings
const letters = ["b", "s", "a", "c", "r", "a", "b", "k"];
letters.sort(); // auto sort

sort.js

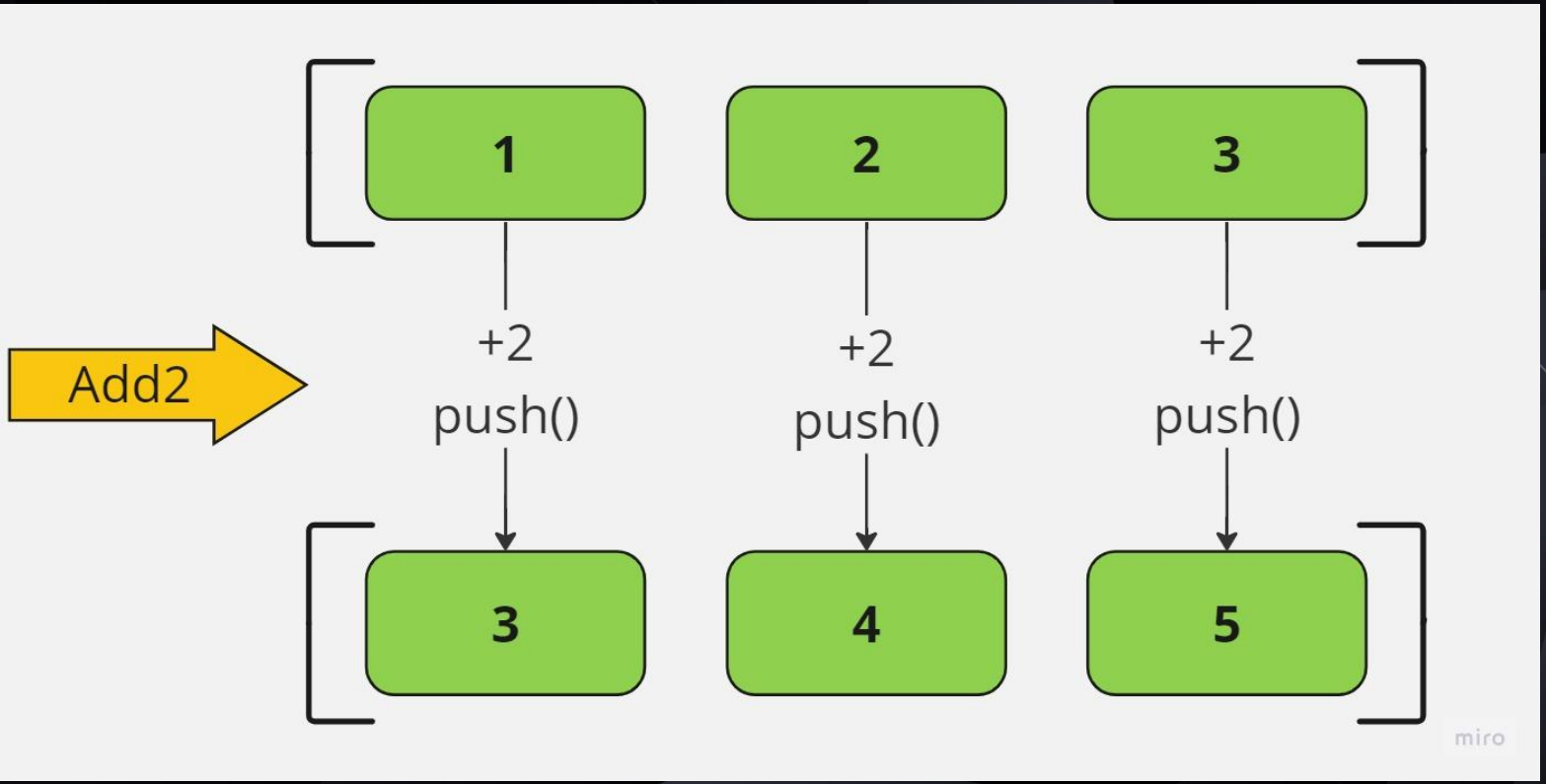
```
1
2  // Array sort - strings
3  const words = ['some', 'words', 'To', 'sort', 'to', 'See', 'some', 'Strange', 'result'];
4
5  words.sort();
6
7  words.sort((a, b) => a.localeCompare(b));
8
```

Array map

- Built-in ‘copy and manipulate’ function
- Returns a copy of the array without modifying the original one
- Takes in a function as an argument
- The function (callback) will be run for each element in our array
- The function will automatically be given 3 argument - callback (currentElement, index, array)

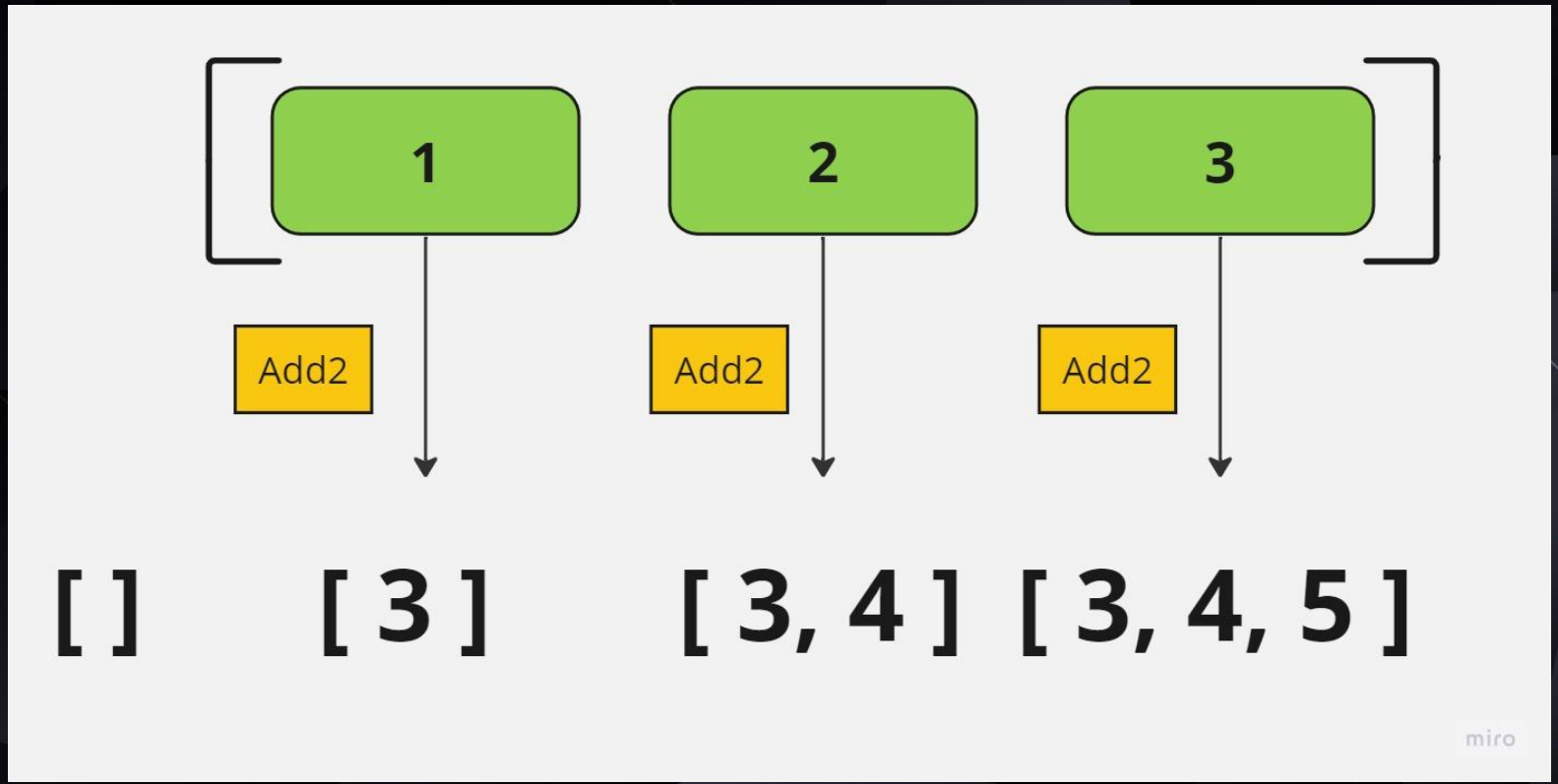
copyAndManipulate.js

```
1
2  function copyAndManipulate(array, instructions) {
3    const output = [];
4
5    for (let i = 0; i < array.length; i++) {
6      output.push(instructions(array[i]));
7    }
8
9    return output;
10 }
11
12 function add2(number) {
13   return number + 2;
14 }
15
16 const myArray = [1, 2, 3];
17 const result = copyAndManipulate(myArray, add2);
18
19 console.log(result); // [ 3, 4, 5 ]
20
```


1
2
3
Add2
+2
push()
+2
push()
+2
push()
3
4
5
miro

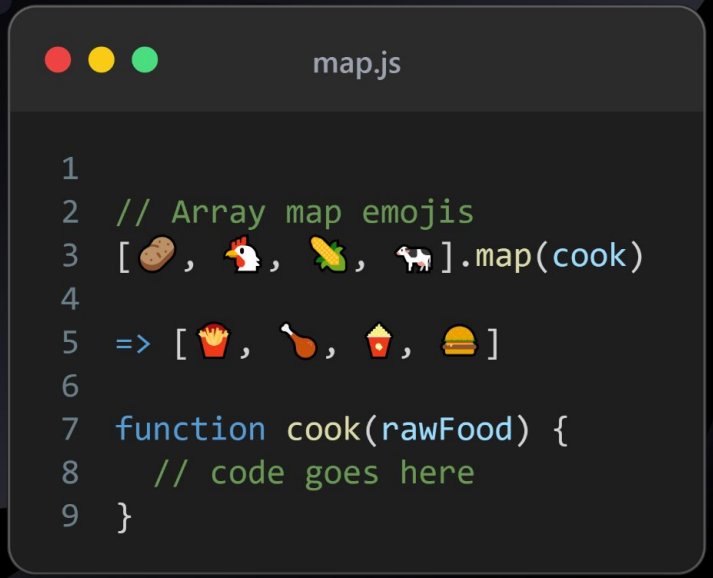
map.js

```
1
2  // Array map
3  function add2(number) { return number + 2 };
4  const myArray = [1, 2, 3];
5
6  const result = myArray.map(add2);
7
8  console.log(result); // [3,4,5]
```


1
2
3
Add2
Add2
Add2
[ ]
[3]
[3, 4]
[3, 4, 5]
miro

map.js

```
1
2  // Array map - arrow function
3  const myArray = [1, 2, 3];
4
5  const result = myArray.map((currentElement) => currentElement + 2);
6
7  console.log(result); // [3,4,5]
```


map.js
// Array map emojis
[🥔, 🐔, 🌽, 🐄].map(cook)
=> [🍟, 🍗, 🍿, 🍔]
function cook(rawFood) {
// code goes here
}

Array map exercise

- Given an array of numbers
- Multiply numbers at odd indexes by 2

map.js

```
1
2  // Array map - multiply odd numbers by 2
3  const myArray = [1, 6, 23, 8, 4, 98, 3, 7, 3, 98, 4, 98];
4
5  const result = myArray.map((currentElement, index) => {
6    if (index % 2 === 1) {
7      return currentElement * 2;
8    }
9
10   return currentElement;
11 });
12
13 console.log(result); // [1,  12, 23, 16, 4, 196, 3, 14,  3, 196, 4, 196]
```

Array reduce

- Returns one value - accumulator (it can still be an array or object!)
- Accepts 2 arguments - callback and initial accumulator value
- If no accumulator is passed - the first element becomes accumulator and is skipped in the loop
- callback (accumulator, currentElement, index, array)

reduce.js

```
1
2  // Array reduce
3  const numbers = [1, 2, 3];
4
5  const result = numbers.reduce((accumulator, currentNumber) => {
6    accumulator.push(currentNumber + 2);
7    return accumulator;
8  }, []);
9
10 console.logI(result); // [3, 4, 5]
```


reduce.js
// Array reduce emojis
[🍟, 🍗, 🍟, 🍔].reduce(eat)
=> 💩
function eat(arguments) {
// code goes here
}

Array reduce exercise

- Using reduce method sum all even numbers and subtract all odd ones

reduce.js

```
1
2  // Array reduce
3  const numbers = [12, 23, 54, 123, 22, 90, 53, 80];
4
5  const result = numbers.reduce((accumulator, currentValue) => {
6    if (currentValue % 2 === 0) {
7      return accumulator += currentValue;
8    } else {
9      return accumulator -= currentValue;
10   }
11 }, 0);
12
13 console.log(result); // 59
```

Array – more functional methods

- filter
- find
- ever
- some
- reduceRight

Array filter

- Returns a copy of the array without modifying the original one
- Takes in a function as an argument
- The function (callback) will be run for each element in our array
- The function will automatically be given 3 argument - callback (currentElement, index, array)
- The callback function should return truthy value to keep the element in resulting array or falsy otherwise

filter.js

```
1
2  // Array filter
3
4  const myArray = [null, 2, "text", false, 9, undefined, 0, [1, 2, 3], true]
5
6  const truthyValues = myArray.filter(value => {
7    return !!value // same as Boolean(value)
8  })
9
10 console.log(truthyValues); // [ 2, 'text', 9, [ 1, 2, 3 ], true ]
11
12 // Alternative
13 myArray.filter(Boolean); // [ 2, 'text', 9, [ 1, 2, 3 ], true ]
```


filter.js
// Array filter emojis
[🍟, 🍗, 🍿, 🍔].reduce(isVegetarian)
=> [🍟, 🍿]
function isVegetarian(food) {
// code goes here
}

Array filter exercise

- Using filter method filter only odd numbers

filter.js

```
const numbers = [12, 23, 54, 123, 22, 90, 53, 80];

const result = numbers.filter((number) => {
  if (number % 2 === 1) {
    return true;
  }

  return false;
});

console.log(result); // [ 23, 123, 53 ]
```

Array find

- Returns first sought element. If element can't be find return undefined
- Takes in a function as an argument
- The function (callback) will be run for each element in our array
- The function will automatically be given 3 argument - callback (currentElement, index, array)
- The callback function should return truthy value for sought element

find.js

```
1
2  // Array find
3
4  const haystack = ["Hay", "Hay", "Needle", "Hay", "Hay", "Hay", "Hay", "Hay"];
5
6  const result = haystack.find(element => element === "Needle");
7
8  console.log(result); // "Needle"
9
```

find.js

```
1
2  // Array find emojis
3
4  [🔴, 🔴, 🔴, 🔴, 🔴, 🔵, 🔴, 🔴, 🔴, 🔴].find(findBlueBall)
5
6  => 🔵
7
8  function findBlueBall(ball) {
9    // code goeas here
10 }
```

every_some.js

```
1
2  // Array every & some
3
4  [📖, 📖, 📖, 📖, 📖, 📘, 📖, 📖, 📖].every(isEveryBookOpen)
5
6  => false
7
8  function isEveryBookOpen(book) {
9    // code goes here
10 }
11
12 [📖, 📖, 📖, 📖, 📖, 📘, 📖, 📖, 📖].some(isSomeBookClosed)
13
14 => true
15
16 function isSomeBookClosed(book) {
17   // code goes here
18 }
```

Array - chaining

- The nature of functional methods allows for chaining multiple of theme in one go

Array chaining exercise

- For a given array
- Multiply each element by 2
- Show only result < 40
- sort them in descending order

chaining.js

```
1  // Chaining
2  const numbers = [4, 1, 3, 2, 12, 3, 29, 4, 6, 34, 16, 28];
3
4  const result = numbers
5    .map(number => number * 2)
6    .filter(number => number < 40)
7    .sort((a, b) => b - a)
8
9  console.log(result); // [32, 24, 12, 8, 8, 6, 6, 4, 2]
10
```

rickandmortyapi.com

Mapping exercises

- Calculate how many characters are dead

exercise.js

```
1  const characters = require("./characters.json");
2
3  function getCharactersCountWithStatus(status) {
4    const result = characters.reduce((count, character) => {
5      if (character.status === status) {
6        return count + 1;
7      }
8
9      return count;
10   }, 0);
11
12   return result;
13 }
14
15 console.log(getCharactersCountWithStatus("Dead"));
16
```

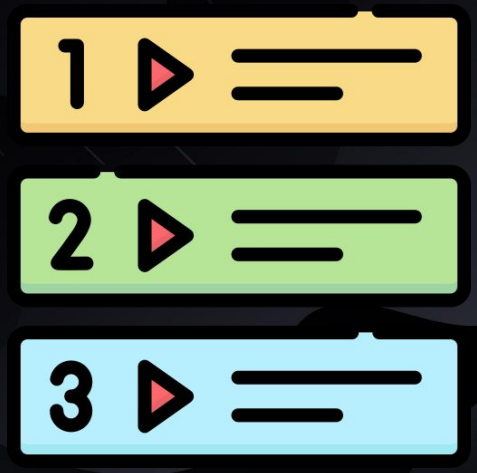
Mapping exercises

- Map all characters living on earth and sort them by name

exercise.js

```
const characters = require("./characters.json");

function getCharactersFromLocation(location) {
  const result = characters
    .filter((character) => {
      return character.location.name.includes(location);
    })
    .sort((charA, charB) => {
      if (charA.name > charB.name) {
        return 1;
      }
      if (charA.name < charB.name) {
        return -1;
      }
      return 0;
    });

  return result;
}

console.log(getCharactersFromLocation("Earth").slice(0, 10));
```

Mapping exercises



- Map and store information for each episode.
- Organize episodes by season, storing each season's episode as an array under a separate key in an object.
- Each episode object should contain only the episode name and number

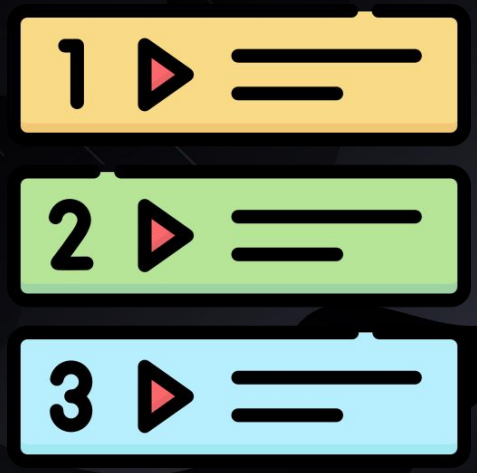
exercise.js

```
const episodes = require("./episodes.json");

function organizeEpisodesBySeason() {
  const result = episodes.reduce((acc, episodeData) => {
    const { name, episode } = episodeData;
    const [season, episodeNumber] = episode.split("E");

    if (!acc[season]) {
      acc[season] = [];
    }

    acc[season].push({ name, episodeNumber });

    return acc;
  }, {});

  return result;
}

console.log(organizeEpisodesBySeason());

```

exercise.js

```
const episodes = require("./episodes.json");

function organizeEpisodesBySeason() {
  const result = episodes.reduce((acc, episodeData) => {
    const { name, episode } = episodeData;
    const [season, episodeNumber] = episode.split("E");

    const dataToAdd = { name, episodeNumber };

    return {
      ...acc,
      [season]: acc[season] ? [...acc[season], dataToAdd] : [dataToAdd],
    };
  }, {});

  return result;
}

console.log(organizeEpisodesBySeason());

```

Mapping exercises



- Map characters and store information for each character
- Count how many episodes characters appeared in
- Store characters name and appearance count
- Sort character by appearance count

exercise.js

```
const characters = require("./characters.json");

function sortCharactersByAppearanceCount() {
  const result = characters
    .map((character) => {
      return {
        name: character.name,
        appearanceCount: character.episode.length,
      };
    })
    .sort((a, b) => b.appearanceCount - a.appearanceCount);

  return result;
}

console.log(sortCharactersByAppearanceCount().slice(0, 10));

```

Mapping exercises

- Map all characters created before April 15, 2018

exercise.js

```
const characters = require("./characters.json");

function getCharactersCreatedBefore(date) {
  const dateUnix = date.getTime();
  const result = characters.filter((character) => {
    const characterCreationUnix = new Date(character.created).getTime();
    return characterCreationUnix < dateUnix;
  });

  return result;
}

const targetDate = new Date(2018, 4, 15);

console.log(getCharactersCreatedBefore(targetDate).slice(0, 10));

```

Mapping exercises

- Map all characters from episodes created before April 30, 2020

exercise.js

```
const characters = require("./characters.json");
const episodes = require("./episodes.json");

function getCharactersFromEpisodesBefore(date) {
  const dateUnix = date.getTime();
  const result = episodes
    .filter((episode) => {
      const episodeCreationUnix = new Date(episode.created).getTime();
      return episodeCreationUnix < dateUnix;
    })
    .map((episode) => episode.characters)
    .flat()
    .map((characterUrl) =>
      characterUrl.replace("https://rickandmortyapi.com/api/character/", "")
    )
    .map((id) => characters[id - 1]);

  return Array.from(new Set(result));
}

const targetDate = new Date(2020, 4, 30);
console.log(getCharactersFromEpisodesBefore(targetDate).slice(0, 10));

```

Mapping exercises

- Map all locations of origin of characters that were in episodes 5 - 25

exercise.js

```
const characters = require("./characters.json");
const episodes = require("./episodes.json");
const locations = require("./locations.json");

function getCahractersLocationFromEpisodes(episodeFrom, episodeTo) {
  const result = episodes
    .filter((episode) => episode.id >= episodeFrom && episode.id <= episodeTo)
    .map((episode) => episode.characters)
    .flat()
    .map((characterUrl) => {
      const characterId = +characterUrl.replace(
        "https://rickandmortyapi.com/api/character/",
        ""
      );
      const characterOrigin = characters[characterId - 1].origin;
      const locationId = characterOrigin.url.replace(
        "https://rickandmortyapi.com/api/location/",
        ""
      );
      if (!locationId) {
        return null;
      }
      return locations[+locationId - 1];
    })
    .filter((location) => location != null);

  return Array.from(new Set(result));
}

console.log(getCahractersLocationFromEpisodes(5, 25).slice(0, 10));

```

HOMEWORK

- Financial data calculations and mapping
- Star Wars data mapping