Date. Regular Expression

DEADLINE: 20/12/2021

FOLDER STRUCTURE

FL18_HW7/*
- homework/*
  - index.html*
  - index.js*
  - .eslintrc.js

* - required

TASK

Write all of the tasks inside `index.js` file.

1. Write a function that accepts a person's date of birth as an input and returns the person's exact age today. For the sake of the task, let's assume that the input is always a valid date object.

```
const birthday22 = new Date(2000, 9, 22);
const birthday23 = new Date(2000, 9, 23);
getAge(birthday22); // 20 (assuming today is the 22nd October)
getAge(birthday23); // 19 (assuming today is the 22nd October)
```

2. Write a function that accepts a date/timestamp and returns a textual representation of the corresponding weekday (i.e. 'Monday', 'Tuesday', etc.). For the sake of the task, let's assume that the input is always a valid date object or a timestamp.

```
getWeekDay(Date.now()); // "Thursday" (if today is the 22nd October)
getWeekDay(new Date(2020, 9, 22)); // "Thursday"
```

3. Write a function that will return the number of days until the New Year.

```
getAmountDaysToNewYear(); // 124 (if today is the 30th August)
getAmountDaysToNewYear(); // 365 (if today is the 1st January)
```

4. Write a function that accepts a year as a number and returns the date of Programmer's Day in the year specified. The return value should be a string in the following format: 'DD Mon, YYYY (weekday)'. Reuse the function from the task 2. The Day of the Programmer is celebrated on the 256th day of each year (on September 13 during common years and on September 12 in leap years).

```
getProgrammersDay(2020); // "12 Sep, 2020 (Saturday)"
getProgrammersDay(2019); // "13 Sep, 2019 (Friday)"
```

5. Write a function that accepts a weekday as a string (e.g. 'Sunday') and returns the number of days to the next specified weekday. The input should be case-insensitive. If the specified weekday is today, return `` `Hey, today is ${ specifiedWeekday } =)` ``, otherwise return `` `It's ${ number } day(s) left till ${ specifiedWeekday }` ``.

Please note, although input is case-insensitive, weekday name in the output string should be always in proper case.

```
howFarIs('friday'); // "It's 1 day(s) left till Friday." (on October 22nd)
howFarIs('Thursday'); // "Hey, today is Thursday =)" (on October 22nd)
```

6. Write a function that accepts a string as an input and returns a boolean that defines if the input is a valid JavaScript variable. Use a regular expression to validate the input. Here is the syntax for valid identifiers:
- each identifier must have at least one character.
- valid identifier characters are the following: alpha, digit, underscore, or dollar sign.
- the first character cannot be a digit.

```
isValidIdentifier('myVar!'); // false
isValidIdentifier('myVar$'); // true
isValidIdentifier('myVar_1'); // true
isValidIdentifier('1_myVar'); // false
```

7. Write a function that accepts a string as an input, capitalizes the first letters of each word and returns the capitalized string. Use a regular expression to achieve the desired result.

```
const testStr = "My name is John Smith. I am 27.";
capitalize(testStr); // "My Name Is John Smith. I Am 27."
```

8. Write a function that accepts a string as an input and returns a boolean that defines if the input is a valid audio file. The file is valid if it satisfies the conditions:
- file name consists of 1+ uppercase and /or lowercase letter(s),
- valid extensions: .mp3, .flac, .alac, or .aac.
Use a regular expression to achieve the desired result.

```
isValidAudioFile('file.mp4'); // false
isValidAudioFile('my_file.mp3'); // false
isValidAudioFile('file.mp3'); // true
```

9. Write a function that accepts a string as an input and returns an array of all valid hexadecimal colors extracted from the string (or an empty array if it does not contain any). Use a regular expression that matches colors in either #abc or #abcdef format. Please, pay attention to the word boundaries, so that if the string contains some invalid hexadecimal colors like #eeee, thy will not be parsed partially (i.e. no #eee in the output array).

```
const testString = "color: #3f3; background-color: #AA00ef; and: #abcd";
getHexadecimalColors(testString); // ["#3f3", "#AA00ef"]
getHexadecimalColors('red and #0000'); // [];
```

10. Write a simple password validation function that accepts a string as an input and returns either true (valid) or false (invalid). The password is considered to be valid if it satisfies all of the following requirements:
- there is at least 1 uppercase letter.
- there is at least 1 lowercase letter.
- there is at least 1 number.
- needs to be at least 8 characters long.
It is invalid otherwise. Use a regular expression to validate the password.

```
isValidPassword('agent007'); // false (no uppercase letter)
```

```
isValidPassword('AGENT007'); // false (no lowercase letter)
isValidPassword('AgentOOO'); // false (no numbers)
isValidPassword('Age_007'); // false (too short)
isValidPassword('Agent007'); // true
```

11. Write a function that takes a number or a string that can be easily converted to a number, inserts commas between the numbers as thousands separators and returns the formatted string. Use a regular expression to achieve the desired result.

```
addThousandsSeparators("1234567890"); // "1,234,567,890"
addThousandsSeparators(1234567890); // "1,234,567,890"
```

12. Write a function that take a text as argument and return array of URLs which are in that text. Use a regular expression to achieve the desired result.

const text1 = "We use https://translate.google.com/ to translate some words and phrases from https://angular.io/ ";
const text2 = "JavaScript is the best language for beginners!"

```
getAllUrlsFromText(text1); // ["https://translate.google.com/", "https://angular.io/"]
getAllUrlsFromText(text2); // []
getAllUrlsFromText(); // (error)
```

RESTRICTIONS

- Using any external libraries is forbidden.

BEFORE SUBMIT

- Remove all unnecessary files that you might have included by mistake.
- Verify that the names of the folders and files meet the requirements.
- Verify that all functionality is implemented according to the requirements.
- Make sure you code is well-formatted and validated via validator (w3org Markup Validation Service).
- Add comments only if the code is difficult to understand.
- Make sure there are no errors/warnings in the browser console; fix them if any.
- Run the linter and fix all the warnings and errors.

HOW TO

Use linter:

- In order to use npm package manager you should install nodejs (https://nodejs.org/)
- Install eslint to check your code (npm install -g eslint)
  - open a terminal (or cmd)
  - run eslint (i.e. eslint ./js/task1.js)

  Code should be without ‘errors’

SUBMIT

- The folder should be uploaded to GitLab repository “**FL-18**” into **main** branch.

USEFUL LINKS

- https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Global_Objects/Date
- https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Global_Objects/RegExp
- https://javascript.info/date
- https://javascript.info/regular-expressions
- https://regex101.com/