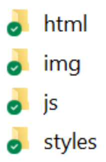
TP 03 : Web Programming
L2

Instruction :

- The purpose of these labs is to teach you to find information on your own. The teacher's role is to support you in this process and to guide you to the most relevant materials.
- You need to divide your tasks among the different members of your team. This will help you to make progress
- The lab "TP3" was designed on the basis of the Javascript-Part2 course
- The different parts of this lab are independent.
- Some parts of the lab are marked "**Bonus**", the others are mandatory!
- The final result is a compressed folder containing the code and having the following structure::



html
img
js
styles

This lab will be the one that will be graded

Learning Outcome:

- At the end of this lab, you should be able to:
  - Know how to define a function in 3 different ways
  - Use arrays and iteration protocols
  - Understand the different aspects related to the notion of objects in Javascript
  - Understand the JSON format and use the JSON object

Starting point :
The starting code for your TP is a git repository that you can found here here. It is the result of TP2.

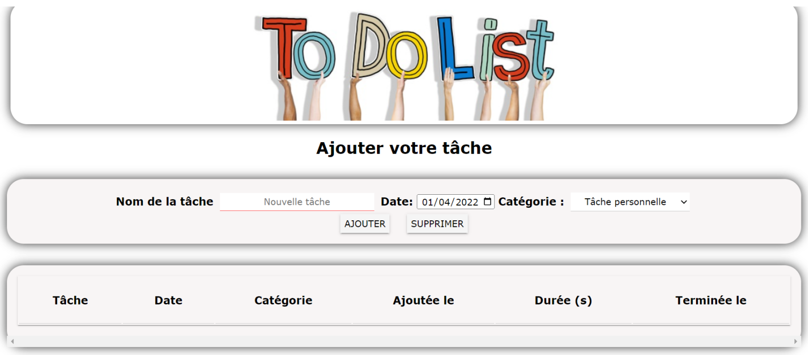
Part 1 : Handling the asynchronous

Recommended reading :

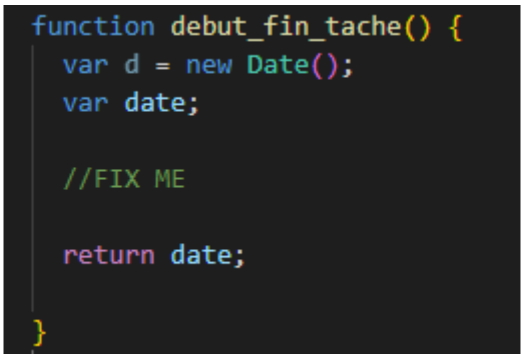
- setTimeout
- setInterval.
- L'objet Date()

Step 1 : Add the date and time of the beginning of a task

1. In the "AddTask.html" file, add 3 new columns to the task table: "Added on", "Duration (s)", "Completed on"



2. Write a function start_end_task() that retrieves the current date and time when a task was added.



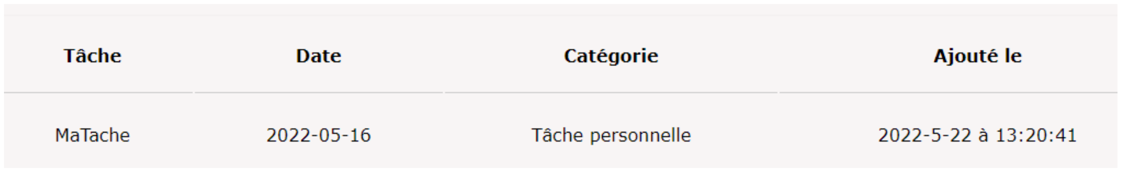
```
function debut_fin_tache() {
  var d = new Date();
  var date;

  //FIX ME

  return date;

}
```

N.B : Think of creating the new date using new Date(), you will then have access to the different methods: getHours(), getMinutes(), getSeconds(), getFullYear(), getMonth() and getDate(). The return of this function must be a string (see image below):



Tâche	Date	Catégorie	Ajouté le
MaTache	2022-05-16	Tâche personnelle	2022-5-22 à 13:20:41

3. When adding a new task (new row in the table), the function start_end_task() must be called and the result must be included in the column "Added on".

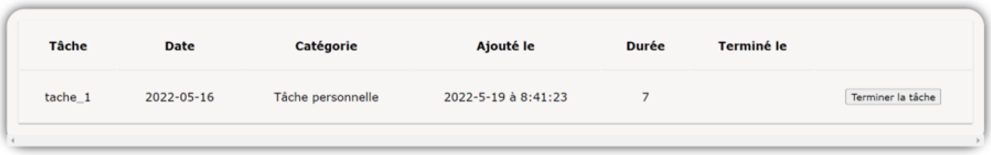
Step 2 : Starting a task

Our goal is to add a counter in the "Duration(s)" column. This is a value that will be incremented every second until the task is finished.

4. Using setTimout() and the incrementerDuree() function below, add a value in the column that automatically increments when a new task is created.

```
function incrementerDuree() {
    let durees = document.getElementsByClassName("duree")
    if (durees.length != 0) {
        Array.prototype.forEach.call(durees,
function(dureeElement) {
            let valeur = parseInt(dureeElement.textContent)
            dureeElement.textContent = valeur + 1
        });
    }
}
```

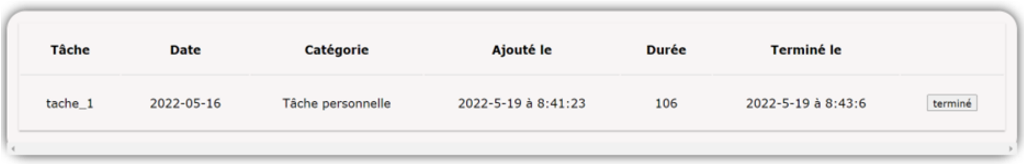
5. Add a "Finish task" button each time a task is added: :



Tâche	Date	Catégorie	Ajouté le	Durée	Terminé le	
tache_1	2022-05-16	Tâche personnelle	2022-5-19 à 8:41:23	7		Terminer la tâche

Hint: The idea here is to add the <button> element to the array (use of createElement). And to also add a text element that has an initial value of "Finish task", then when pressed (addEventListner), the text becomes "Finished!"

7- Display the end date when the user clicks on the "Finish task" button. You have to call the function start_end_task(). The counter should stop, the end date is displayed in the "Finished on" column and the button changes its value.



Tâche	Date	Catégorie	Ajouté le	Durée	Terminé le	
tache_1	2022-05-16	Tâche personnelle	2022-5-19 à 8:41:23	106	2022-5-19 à 8:43:6	terminé

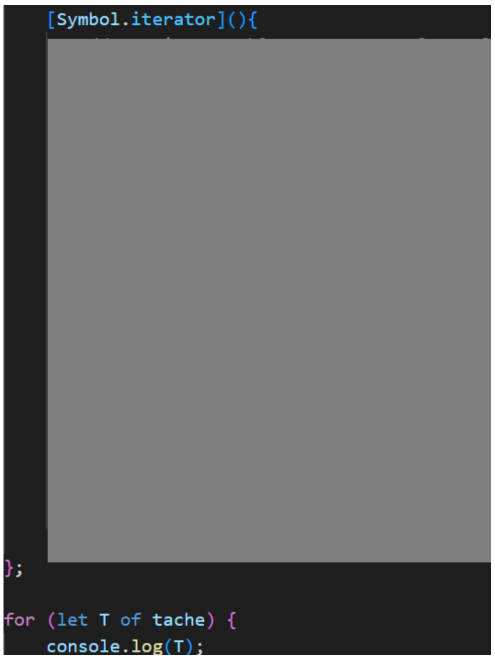
Bonus : Iteration protocols

The objective of this part is to manipulate the tasks using the iterator protocol.

1. In the script.js file, define a "task" iterator object. .

```
let tache = {
```
[redacted]

2. Display the elements of a task added in the console using the iterator protocol.



```
    [Symbol.iterator](){
```
[redacted]
```
};

for (let T of tache) {
    console.log(T);
```

Part 2 :Anonymous functions and arrow functionsBender

- The objective of this part is to add an "easter egg" in our list of tasks.
- By adding some tasks, we will be able to control a small hidden robot..

Step 1 : Create the robot

1. Create a new bender.js file and link it to our task list page.
2. In this file, add the create_robot() function which adds a robot image to our page. Our robot will be positioned at the top left of our page and has an id "bender".

```
function create_robot(){
    let bender = document.createElement('img');
    bender.src = "../img/Bender.png";
    bender.style.height = "100px";
    bender.style.position = "absolute";
    bender.style.top = "0px";
    bender.style.left = "0px";
    bender.style.transition = "all 2s";
    bender.id = "bender";
    document.body.append(bender);
}
```

You should have only modified your js file, not the HTML or CSS.

Step 2 : Controlling the robot

3. Within the bender.js file, create a move_bot_right() function that moves Bender to the right (by 100px). You will need to modify the "left" property. Be careful with your types.
4. Do the same to get the functions `move_bot_left()`, `move_bot_down()`, `move_bot_up()`

Step 3 : Easter Egg

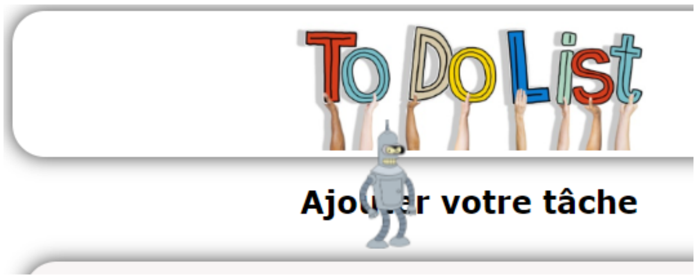
We will now use our task list as a script executor.
Each task that starts with BOT_ will be in fact a command to our robot.
The tasks we need to identify are the following: :

Nom de la tâche	**Effet**
BOT_CREATE	Fait apparaître le robot
BOT_RIGHT	Fait déplacer le robot vers la droite
BOT_LEFT	Fait déplacer le robot vers la gauche
BOT_UP	Fait déplacer le robot vers le haut
BOT_DOWN	Fait déplacer le robot vers le bas

BOT_RUN	Dès que cette tâche est ajoutée, elle déclenche l'exécution de toutes les tâches BOT de la liste de tâches

5. In the bender.js file, create a function `activate_bot()`. This action will be triggered when the BOT_RUN task is added.
6. To test, only put an alert saying "BOT GO!" for the moment in the `activate_bot`() function
7. Make sure that adding the BOT_RUN task triggers the call of the `activate_bot`() function. (you may have to modify another part of your code).
8. Edit the `activate_bot` function to console.log each task name in the task list. The idea is to go through the task list table and extract the HTML from the first box in each row.
9. .Now create a `get_bot_action(query)` function that takes a string as parameter. This function must parse the received query to determine which bot function to call. If for example your function receives the string "BOT_LEFT", you have to execute the function `move_bot_left()`.

The robot will appear in its final position, without you seeing it move, we will change that later.



Step 4 : Clean up

This step will introduce you to the anonymous functions.

The 5 bot action functions will only be useful in the `get_bot_action()` function. You can define them **IN** the get_bot_action function.
Also, you can return the functions as the return value of `get_bot_action`. So your `get_bot_action` function will return...a function.

You can then take the code from your previous functions and put it in the `get_bot_action function.`

So basically, depending on the query you receive, you will return one complete function or another.
You won't need to name the different functions anymore, and will have an architecture like this:

```
function get_bot_action(query){
    switch(query.substring(4)){
        case "CREATE":
            return function (){
                //Blah Blah
            };
        case "RIGHT":
            return function (){
                //Blah Blah
            };
        case "LEFT":
            return function (){
                //Blah Blah
            };
        case "DOWN":
            return function (){
                //Blah Blah
            };
        case "UP":
            return function (){
                //Blah Blah
            };
        default:
            return function(){};
    }
}
```

By using arrow functions, you can also remove the function keyword, replacing `function() {` with `() => {`.

But then the functions are returned, not executed yet. So you will have to execute the returned function in your table traversal loop of the `activate_bot()` function. Since your `get_bot_action` function **RETURNS** a function, you can add parentheses after your function call to execute it directly. So you will have a line close to:
`get_bot_action(params)`**();**

Step 5: additional features

Now you can add any feature by only modifying the get bot_action function. You only need to add a box, the recognized request, and the functionality performed by the anonymous function sent.

Add a feature of your choice to your easter egg.

Last Step (Bonus) : Animation

For the moment, your robot is not animated. It appears on its final position. This is because your browser does not have time to realize that the properties have changed.
Even if you add a "transition: all 2s" property to your robot, you won't see anything.

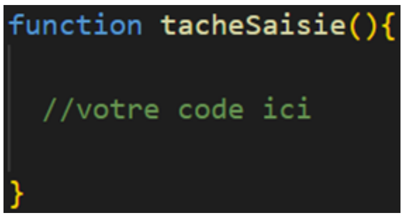
Using the setTimeout seen in part 1 of the tutorial, make sure to trigger each function returned by get_bot_action after a small delay.

Part 3 : The javascript objects

Recommended reading:

- Objet littéral.
- Classe et instance de classe
- Tableau d'objets

1. Create a `tacheSaisie()` function that retrieves the values entered by the user in the form



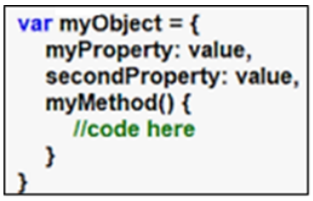
```
function tacheSaisie(){

  //votre code ici

}
```

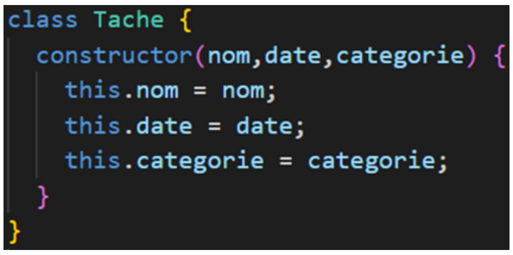
2. Create a literal object `myNewTask` (i.e. literally write the content of the object to create it, and return it). This object consists of three properties `name, date, category` whose values are those entered by the user.

The syntax of a literally created prototypical object is as follows:



```
var myObject = {
   myProperty: value,
   secondProperty: value,
   myMethod() {
      //code here
   }
}
```

3. Create a class **Tache** with three properties name, date and category. We create a new class with the class keyword. In our **class**, we define a **constructor()** method which will be used to initialize the properties of objects instantiated later from this class with the current values of the objects..



```
class Tache {
  constructor(nom,date,categorie) {
    this.nom = nom;
    this.date = date;
    this.categorie = categorie;
  }
}
```

4. Modify `your tacheSaisie()` function to create a `newTache` object instantiated from the `Tache` class (instead of the prototypical `myNewTache object`), and return it.

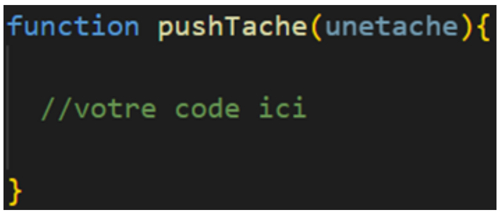
   To create an instance of the Task class, you use the following syntax, knowing that inputName, inputDate and inputCategory are the values retrieved from the form after being entered by the user::

```
const nouvelletache = new Tache(nomSaisi,dateSaisie,categorieSaisie);
```

   <u>Attention : Ne supprimez pas le code de la question 1, mais juste commenter le (encadrer le code par des /* */), afin qu'il soit bien noté.</u>

5. Declare a collection of tasks `myTasks` of type empty array.

6. Create a `pushTache()` function that adds a task passed as input to the `myTasks` collection. The size of the `myTasks` array is dynamically changed with each addition of an element..



```
function pushTache(unetache){

  //votre code ici

}
```

7. Modify the add() function you implemented in TP2 to:

   a. Retrieve the entered values, create the task object and add it to the myTasks array using the pushTache() and tacheSaisie() functions.

   b. Implement a forEach loop to add each task element of the myTasks table to the table in the main section of your html document. .

   c. Avoid duplicate instances in this table.

Part 4 : JSON file vs JSON object

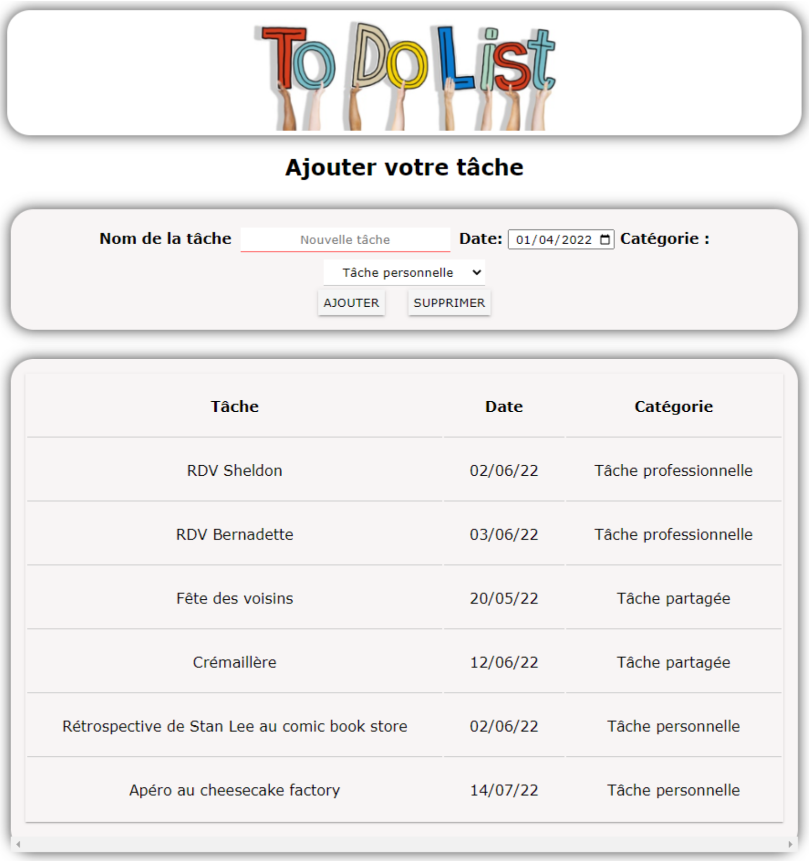
Recommended reading:

- Loop and iterator
- Fichier JSON externe
- Les Array
- Stringify

Step 1 : Read my local JSON file and get the data in JSON format

1. Create a tasks.json file that contains a list of tasks. N.B.: Each task should contain the key task, date and category

   Example :



To Do List

Ajouter votre tâche

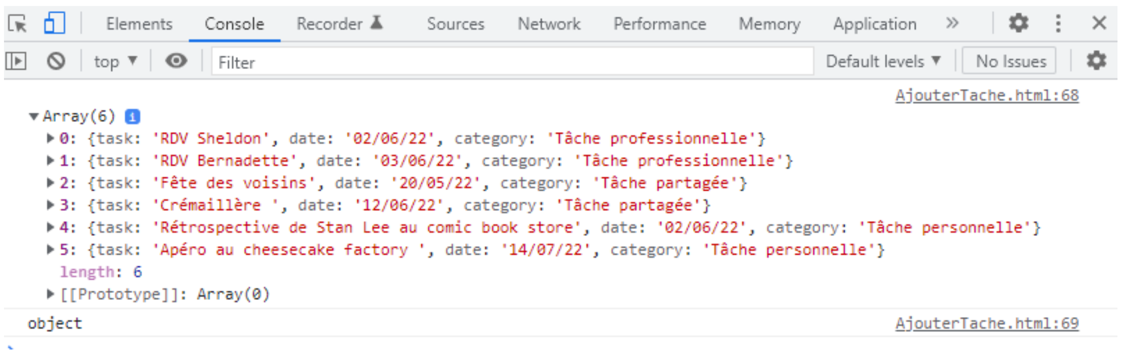
Nom de la tâche Nouvelle tâche Date: 01/04/2022 Catégorie :

Tâche personnelle

AJOUTER SUPPRIMER

Tâche	Date	Catégorie
RDV Sheldon	02/06/22	Tâche professionnelle
RDV Bernadette	03/06/22	Tâche professionnelle
Fête des voisins	20/05/22	Tâche partagée
Crémaillère	12/06/22	Tâche partagée
Rétrospective de Stan Lee au comic book store	02/06/22	Tâche personnelle
Apéro au cheesecake factory	14/07/22	Tâche personnelle

2. In your AddTask.html file, load the tasks.json file and display its type in the console using "typeof".



```
                                                                     AjouterTache.html:68
▼Array(6)
  ▶0: {task: 'RDV Sheldon', date: '02/06/22', category: 'Tâche professionnelle'}
  ▶1: {task: 'RDV Bernadette', date: '03/06/22', category: 'Tâche professionnelle'}
  ▶2: {task: 'Fête des voisins', date: '20/05/22', category: 'Tâche partagée'}
  ▶3: {task: 'Crémaillère ', date: '12/06/22', category: 'Tâche partagée'}
  ▶4: {task: 'Rétrospective de Stan Lee au comic book store', date: '02/06/22', category: 'Tâche personnelle'}
  ▶5: {task: 'Apéro au cheesecake factory ', date: '14/07/22', category: 'Tâche personnelle'}
   length: 6
  ▶[[Prototype]]: Array(0)
object                                                               AjouterTache.html:69
```

3. Make a loadTasks(tasks) function in your script.js file that takes the list of tasks from the tasks.json file and adds it to the table as soon as the page is loaded (see image above)
4. Make a function that retrieves the set of tasks from the html table and transforms it into a JS table. Convert this JS array to JSON format.
5. Display the value of this JSON object with console.log

Step 2 :Reading my external JSON file

Recommended reading:

- JSONPlaceholder

The objective of this exercise is to fill the Album table from a JSON file stored on an external server. To do this, we will use AJAX (Asynchronous JavaScript + XML). We will use the **fetch() API** of javascript to retrieve data from a given URL.

1. Let's go back to the main page of our example: Part_1.html. We will display a list of all the "Task Title" and "Status" information of the tasks stored in the JSON file found here.

2. In a <script> tag in your html, add the following code to load the data from the JSON file and the console.log.

```
// Indice de la page courante, on commence la pagination à l'indice 0
      let page = 0;
      const tasksCount = 200; // Nombre total de tâches
      const itemsPerPage = 10; // Nombre de lignes par page
// Fonction de récupération des tâches
function getTasks(page) {
  // On récupère les données depuis le serveur externe
  // Premier élément de la page courante
  const start = page * itemsPerPage;
  // Nombre d'éléments par page
  const limit = itemsPerPage;
  /*
    On se sert des paramètres (Query parameters) _start et _limit pour
obtenir un résultat paginé
    _start : définit l'indice du premier élément à récupérer dans le
tableau de résultat
    _limit : définit le nombre d'éléments à récupérer
  */
  fetch(
    "http://jsonplaceholder.typicode.com/todos?_start=" +
      start +
      "&_limit=" +
      limit
  )
```

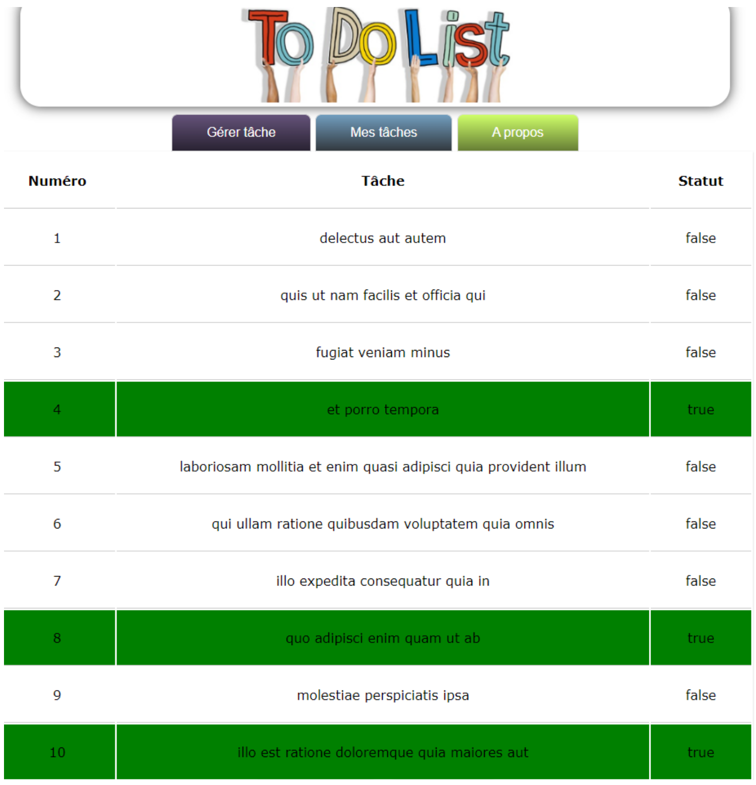
```
    .then((response) => response.json())
    .then(function (data) {
      // Un tableau (Array) de 200 objets javascript représentant des
tâches s'affiche dans la console du navigateur
      console.log(data);

      // On stocke le nombre total de tâches récupérées pour la
pagination
      //tasksCount = data.length;

      // On parcourt le tableau de tâches récupéré et on ajoute une
ligne au tableau de tâche pour chaque élément du tableau
      for (let i = 0; i < data.length; i++) {
        createTask(data[i]);
      }
    });
}
// Appel de getTasks au chargement de la page (On récupère la première
page de résultat (indice 0))
getTasks(0);
```

N.B : getTasks(0) allows to load the file when the page is loaded..

3. Create the createTask() function that adds a row to the task table. Hint: Take inspiration from TP2 😊
4. Using the data object received from the API, call createTask() as many times as there are elements in the Javascript array. .
5. Make the completed tasks green by using the "style.backgroundColor" property.



Numéro	Tâche	Statut
1	delectus aut autem	false
2	quis ut nam facilis et officia qui	false
3	fugiat veniam minus	false
4	et porro tempora	true
5	laboriosam mollitia et enim quasi adipisci quia provident illum	false
6	qui ullam ratione quibusdam voluptatem quia omnis	false
7	illo expedita consequatur quia in	false
8	quo adipisci enim quam ut ab	true
9	molestiae perspiciatis ipsa	false
10	illo est ratione doloremque quia maiores aut	true

BONUS : Limit the number of lines per page in the table

1. At the end of the task table, create a new <nav> navigation menu containing two buttons **Page Up** and **Page Down.**

2. Write a removeAll() function to empty the contents of an array using *tbody.removeChild()*.

3. Using a variable to store the current position, write the getPreviousPage() and getNextPage() functions that call the getAlbum() function and decrement or increment the current position respectively. .

4. Write the functions showPreviousPage() and showNextPage() which display the previous page or the next page respectively. These two functions call the removeAll function to empty the table before each new display.

N.B : Do not forget to deal with the case where you are on the first page (so no previous page) or on the last page (so no next page)..

A bonus : API et Ajax

It consists in handling a small API which returns the name, first name and email of a student with an identifier passed as an argument (GET). https://codepen.io/f_guillemard/pen/gOvLoPa