**Day1**

**File and FileReader**

 File objects inherits from blob and is extended with file system related capabilities and there are two ways to obtain it; through constructor and <input type = "file">, or drag and drop. File Reader is an object that allows web applications to asynchronously read the contents of files (or raw data buffers) stored on the user's computer.

**Here’s an example of reading a file**  
document.getElementById('file-input').addEventListener('change', function(event) {  
var file = event.target.files[0];  
var reader = new FileReader();  
reader.onload = function(event) {  
var contents = event.target.result; document.getElementById('output').textContent = contents;  
};reader.onerror = function(event) {  
console.error("File could not be read! Code " + event.target.error.code);  
};reader.readAsText(file);  
});

**Fetch**

We can use fetch() send network requests to the server and load new information whenever is needed. The syntax is let promise e= fetch (URL, [option]).

**Post Requests**

In order to make a POST request or a request with another method, we need to use the fetch method options;  
-method; HTTPS method like POST  
-body; string, fromData, blob/ bufferSource, etc.

**Activity 3**

Response provides multiple promise-based methods to access the body in various formats:  
response.json() – parse the response as JSON object,  
response.text() – return the response as text,

**Sending an Image**

Binary data can also be submitted directly using blob or bufferSource. The following are some of the response properties;  
-response.status -response.ok - response.headers  
- response.json() -response.text() etc.

**fetch users from Github**

Fetch API in JavaScript can be used to make asynchronous requests to the GitHub API, for example;  
async function getUsers(names) {  
const baseUrl = "https://api.github.com/users/";  
const userPromises = names.map(async name => {  
const response = await fetch(baseUrl + name);  
const userData = await response.json();  
return userData;  
});const users = await Promise.all(userPromises);  
return users;  
}

**Day 2**

**Sending a simple form**

We can send html forms with or without files, with additional files and so on. Its constructor is let formData = new formData ([form]). Network methods such as fetch can accept a formData object as body.

**FormData Methods**

 We modify the fields in formData using methods like; formData.append(name, value), formData.delete(name), etc. It is technically allowed to have many fields with the same name, multiple calls to append add more same-named fields.

**Sending a form with a file**

A form is always send as content; form or multipart, this encoding allows to send files. So <input type="file> fields are send as well, similar to an usual form submission.

**Fetch: Download progress**

The fetch method allows us to track download progress. Currently XMLHttpRequest is used to track upload progress. Response.body is used to track the download progress.

**Sending a form with Blob data**

FormData objects are used to capture HTML form and submit it using fetch or another network method. We can either create new FormData(form) from an HTML form, or create an empty object, and then append fields with methods:  
formData.append(name, value)  
formData.append(name, blob, fileName)  
formData.set(name, value)  
formData.set(name, blob, fileName)

**Fetch: Abort**

 Since JavaScript has no concept of 'aborting' a promise, we have a special built in objects for such purposes; AbortController().

**Day 3**

**Fetch: Cross-Origin Requests**

 Cross-Origin Requests (CORS) is a security feature implemented by web browsers to prevent web pages from making requests to a different domain (origin) than the one that served the web page.

**Using Forms**

One way to communicate with another server is to submit a <form>, it was submitted into <iframe> just to stay on the current page.

**Simple Requests**

Simple request is a request that satisfies two conditions; being simple method: GET, POST or HEAD and simple headers. A request with PUT method or with an API-key HTTP-header does not fit the limitations. The essential difference is that 'simple request' can be made with a <form> or a <script> without any special methods.

**CORS for Simple Requests**

 if the request is cross-origin, the browser always adds Origin header to it. Origin contains exactly the origin (domain/ protocol/ port), without a path. The server can inspect the origin and, if it agrees to accept such a request, it adds a special header Access-Control-Allow-Origin to the response.

**Response Headers**

Response headers are a part of the HTTP response sent by a server after receiving a request from a client (web browser). They provide additional information about the server's response, such as the content type, caching directives, content length, and more. These headers are included in the response message sent back to the client.

**Non-simple Requests**

 We can use any HTTP-method; not just GET/ POST, but also PATCH, DELETE and others. A preflight request uses method OPTIONS and has body.

**Credentials**

A cross-origin request by default does not bring any credentials (cookies or HTTP authentication) and that is common for http-requests. But cross-domain requests made by JavaScript methods are an exception because a request with credentials is much more powerful than an anonymous one.

**Fetch API**

 The Fetch API is a modern JavaScript interface for making HTTP requests. It provides a more powerful and flexible way to interact with resources on the web compared to older techniques like XMLHttpRequest.

**Reflections**

 Fetch-Cross Origin Requests, if we make a fetch from an arbitrary website, that will probably fail because of the origin-a domain; domain / port/ protocol triplet.

**Day 4**

**Patterns and Flags**

A regular expression, also regexp, or just reg consists of a pattern and optional flags, and there are two syntaxes to create a regular expression being long syntax and short syntax.

**Usage**

 To search inside the string we can use the method search. The short syntax for regexp does not support variable insertions ${...}, but the new RegExp allows to construct a pattern dynamically from a string, so its more flexible. Regular expressions may have flags that affect the search and there are six of them in JavaScript:  
- i - g -m -s - u -y

**Flags**

Flags are optional parameters that can be added to regular expressions and the g, i, m, and s flags. Flags modify the behaviour of these functions, allowing for more flexible and powerful operations.

**Daily Notes - Methods of RegExp and String**

 Regular expressions (RegExp) and strings have several methods that allow us to perform various operations. Some of the commonly used methods are regexp.exec, regexp.test etc. There are two methods to deal with regular expressions, they are objects of the built-in RegExp class.

**Character classes**

 Character classes, are a way to match a single character from a specified set of characters in a regular expression. We have a digit class, \d, word class,\w, and space class,\s. We can not use many character classes.

**Word boundary \b**

 A word boundary \b is a special character class that doesn't denote a character but rather a boundary between characters. It has zero match in a sense that usually a character class means a character in the result but not in this case. \bJava\b matches Java in the string Hello, Java!

Inverse classes in regular are patterns that match any character except those specified. For every character there exist an 'inverse class', denoted with the same letter, but uppercased. For example, \D for non-digits.

**Spaces are regular characters**

 Spaces are considered regular characters in regular expressions. In a regular expression pattern, a space is treated just like any other character, unless it's specified as part of a special character sequence like \s, which matches any whitespace character, including spaces, tabs, and newlines.

**A dot is any character**

 A dot is a special character that matches "any character" except for a new line. It means any character but not the absence of a character, there must be a character to match it.

**The dotall “s” flag**

 The dotall 'S' flag usually does not match a newline character, for instance, A.B matches A and then B with any character between A and B, except a newline.

**Find the Time**

let regexp = /\b\d{2}:\d{2}\b/g;  
let txt = "Breakfast at 09:00 in the room 123:456.";  
let matches = txt.match(pattern);  
console.log(matches);

**Escaping, Special character**

 Escaping in regular expressions involves treating a character with a special meaning as a literal character. In a regular expression, many characters have special meanings, such as : [ \ ^ $ . | ? \* + ( ). etc. To match these characters literally, we "escape" them by using a backslash \ before them.