Two examples of Asynchronous code

1. backend

Taken from the first Node/Express/Knex/MariaDB back-end demo code.

```
/** http://localhost:8787/api/people  (and with GET request) **/

router.get('/', function(req,res) {
    db.select().from('Person').then( (data) => {
        res.status(200);
        res.send(data);
    }
    );
}
);
```

1. The code with blue color is run when the >node index.js is run, i.e. in the server startup

The /index.js is the starting point of the node server. But before the /index.js will be completed, other modules are run too and they define common **settings** (app.use()), **routings** (app.use() and router.use()) and **services** (app.get(), app.post()). The /index.js will start the server (app.listen())

(Now several **minutes**, **hours** or even **days** might pass before the next parts of the code to be executed = Some Front-end user will ask this service to be used!)

2. The code with red color is run each time when an HTTP request comes to the correct URI with the correct (GET) method. That part will initiate database operation and **setup** the call-back function (with anonymous array function ( ) => { }) to be called when the database operation has completed. The red part has already finished before the database operation completes. So red part cannot do anything with the results! As it finishes before we get the results. It can only setup how the results WILL be handled. Setup is done using the .then( ) promise that has been defined in the Knex library.

(Now (hundreds of **nanoseconds**,) few **milliseconds** or even few **seconds** might pass before the next parts of the code can be executed = Database has to do its part first!

Delay length depends on many things. E.g. in-memory database that is kept in RAM in same machine and only a little data, could be nanoseconds. Busy or low power database, slow connections and large amount of data, could take few seconds = ~million times longer)

3. The code with golden color will be run when the database operation has completed. That third part will be already totally synchronous, no waits/promises/call-backs/event-handler setups anymore.

Short summary about : **Asynchronous operations**. We:

1. **Initiate** some function=action that needs something from another system (OS, file system, database, back-end server over internet) and we cannot continue directly in this thread, as we would not have the response yet.
2. Instead **we setup**/define **call-back** functions/event-handlers/promises that would be run when…
3. That **call-back will be called by the environment**/system after the operation somewhere else has finished, kind of returning to our code second time, with the needed results. => Write only in the callback how you continue with the ready results!

(Many different kinds of options/methods exist for setting up asynchronous code. Learn them when you come across them.)

2. frontend

Another code example, from Front-end, how the Axios AJAX library executes the request, and sets the success and error handlers. When the AJAX response arrives from backend one of them will be executed.

```
const ajaxRequest = {
      method: 'get',
      url: API_ROOT + '/category/all'
};

axios(ajaxRequest)
  .then((response) => {
      doSomethingWithSuccesfullyReceivedData(response.data);
  })
  .catch((error) => {
      console.error("Error: " + error);
      doSomethingElseWithTheErrorObject(error);
  });

// End
```