the shortcut

Fullstack Web Development Tutorial Lesson 17

Today's lesson will cover

- Promises chaining
- Promise static methods
- Async/await

JavaScript fundamentals

Promises chaining

- The idea is that the result is passed through the chain of `.then` handlers. The whole thing works, because a call to `promise.then` returns a promise, so that we can call the next `.then` on it.
- When a handler returns a value, it becomes the result of that promise, so the next `.then` is called with it.
- **A classic newbie error: technically we can also add many `.then` to a single promise. This is not chaining.**
- In practice we rarely need multiple handlers for one promise. Chaining is used much more often.

Exercise

- Fix the code to make it work providing proper output in both case argument `isGoingToResolve=true` or `isGoingToResolve=false`

```
function doSomething (isGoingToResolve) {
    return new Promise((resolve, reject) => {
    if () {
        resolve("something")
    } else {
        reject("something else")
    }
  }).then(response => {
    console.log("in my function")
  }).catch(error => {
    console.log("in my function",error)
  })
}

doSomething()
      .then(response => console.log("in my main call", response))
```

Promise static methods

- There are 5 static methods of `Promise` class:
  - `Promise.all(promises)` – waits for all promises to resolve and returns an array of their results. If any of the given promises rejects, it becomes the error of `Promise.all`, and all other results are ignored.
  - `Promise.allSettled(promises)` (recently added method) – waits for all promises to settle and returns their results as an array of objects with:
    - `status`: `"fulfilled"` or `"rejected"`
    - `value` (if fulfilled) or `reason` (if rejected).
  - `Promise.race(promises)` – waits for the first promise to settle, and its result/error becomes the outcome.
  - `Promise.resolve(value)` – makes a resolved promise with the given value.
  - `Promise.reject(error)` – makes a rejected promise with the given error.
- Of these five, `Promise.all` is probably the most common in practice.

Exercise

- Make this code provide the following expected output on console:

```
const a = () => new Promise(resolve => {
      setTimeout(() => resolve('result of a()'), 1000); // 1s delay
});

const b = () => new Promise(resolve => {
      setTimeout(() => resolve('result of b()'), 500); // 0.5s delay
});

const c = () => new Promise(resolve => {
      setTimeout(() => resolve('result of c()'), 1100); // 1.1s delay
});


// resolve once all a(), b(), c() resolves and gets a data object { key: 'I am plain data!' }

/* Expected output
success:  [
  'result of a()',
  'result of b()',
  'result of c()',
  { key: 'I am plain data!' }
] */
```

Async/Await

- Async functions always return a promise. If the return value of an async function is not explicitly a promise, it will be implicitly wrapped in a promise.
- Syntax
  - ```
    async function name([param[, param[, ...param]]]) {
       Statements // await mechanism may be used
    }
    ```
- Example
  - ```
    async function asyncFunction() {
          const result1 = await new Promise((resolve) => setTimeout(()resolve('1')))
          const result2 = await new Promise((resolve) => setTimeout(()resolve('2')))
    }
    asyncFunction()
    ```
- The keyword `await` makes JavaScript wait until that promise settles and returns its result. It's just a more elegant syntax of getting the promise result than `promise.then`, easier to read and write.
- When we use `async/await`, we rarely need `.then`, because `await` handles the waiting for us. And we can use a regular `try..catch` instead of `.catch`. That's usually (but not always) more convenient.

Exercise

- Complete this script to run the functions as per instructions in comment:

```
const a = () => new Promise( resolve => {
    setTimeout( () => resolve( 'result of a()' ), 1000 );
} );

const b = () => new Promise( resolve => {
    setTimeout( () => resolve( 'result of b()' ), 500 );
} );

const c = () => new Promise( resolve => {
    setTimeout( () => resolve( 'result of c()' ), 1100 );
} );

// Create an async function function doJobs which takes three variables
// resultA to run promise function a, resultB to run promise function B, and
// resultC to run promise function c sequentially awaiting one after another
// and returns an array with values of all three variables


// doJobs() returns promise result or errors in this code block


// normal flow part of script which outputs first
console.log( 'I am a sync operation!' );
```

the shortcut

Self Study Assignments

To Dos

- Create a game of Rock, Paper and Scissors using JS which works on console, or with interactive UI using HTML, CSS and JS however you prefer ***(If you are working on your own project where you are using JS already, feel free to ignore this task but please share the project update with Lena.)***
- Continue freecodecamp (FCC) Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- If you believe FCC exercises aren't the best for you if you are quite advanced already, please start working on your own project and reach out to mentors for help if needed.