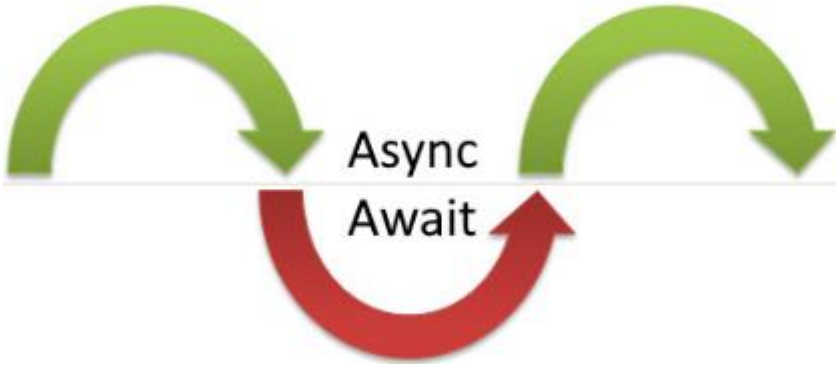
ASYNC & AWAIT

Promises

WHAT ARE THEY



A different syntax to work with **Promises.**

You can still use the Promise API

It's just a short syntax alternative.

OVERVIEW

- `async` is a keyword for the function declaration
- `await` is used during the promise handling
- `await` must be used within an `async` function, though Chrome now supports "top level" `await`
- `async` functions return a promise, regardless of what the `return` value is within the function
- `async` / `await` and promises are essentially the same under the hood

BENEFITS

- Your code is more simplistic, precise
- Debugging is easier thanks to less callbacks
- Conversion from promise `then` / `catch` code is easy
- Your code can be more "top down", less nesting

EXAMPLE

```
async function fetchContent() {
  // Instead of using fetch().then, use await
  let content = await fetch('/');
  let text = await content.text();

  // Inside the async function text is the request body
  console.log(text);

  // Resolve this async function with the text
  return text;
}

// Use the async function
var promise = fetchContent().then(...);
```

BEFORE

```
fetch('/users.json')
  .then(response => response.json())
  .then(json => {
    console.log(json);
  })
  .catch(e => { console.log('error!'); })
```

AFTER

```
async function getJson() {
  try {
    let response = [await] fetch('/users.json');
    let json = [await] response.json();
    console.log(json);
  }
  catch(e) {
    console.log('Error!', e);
  }
}
```

ASYNC

```
async function returnOne() { return 1; }
```

async ensures that the function returns a Promise.

ASYNC

```
async function returnOne() { return 1; }
```

async ensures that the function returns a Promise.

ASYNCHRONOUS
CODE EXECUTION

asynchronous functions will run, and the execution of other code halts (but does not block) until the async action finishes

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait till the promise resolves (*)

  alert(result); // "done!"
}

f();
```

A short syntax that returns a Promise result than using Promise.then()

AWAIT

await code waits until Promise settles and returns (resolve or reject).

```
const value = await myPromise;
```

Only works with async functions

AWAIT

await cannot be used in regular functions.

```
function f() {
  let promise = Promise.resolve(1);
  let result = await promise; // Syntax error
}
```

AWAIT

Replace .then() with await.

```
async function showAvatar() {

  // read our JSON
  let response = await fetch('/article/promise-chaining/user.json');
  let user = await response.json();

  // read github user
  let githubResponse = await fetch(`https://api.github.com/users/${user.name}`);
  let githubUser = await githubResponse.json();

  // show the avatar
  let img = document.createElement('img');
  img.src = githubUser.avatar_url;
  img.className = "promise-avatar-example";
  document.body.append(img);

  // wait 3 seconds
  await new Promise((resolve, reject) => setTimeout(resolve, 3000));

  img.remove();

  return githubUser;
}

showAvatar();
```

AWAIT

Await has to be inside $\boxed{\text{async function}}$.

Won't work

```
// syntax error in top-level code
let response = await fetch('/article/promise-chaining/user.json');
let user = await response.json();
```

Works

```
(async () => {
  let response = await fetch('/article/promise-chaining/user.json');
  let user = await response.json();
  ...
})();
```

DECLARING

Anonymous Async Function

```
let main = (async function() {
  let value = await fetch('/');
})();
```

Async Function Declaration

```
async function main() {
  let value = await fetch('/');
};
```

Async Function Assignment

```
let main = async function() {
  let value = await fetch('/');
};
```

PASSING
AS ARGUMENTS

Async Function as Argument

```
document.body.addEventListener('click', async function() {
  let value = await fetch('/');
});
```

OBJECTS
AND METHODS

```
// Object property
let obj = {
  async method() {
    let value = await fetch('/');
  }
};

// Class methods
class MyClass {
  async myMethod() {
    let value = await fetch('/');
  }
}
```

PARALLELISM

```
// Will take 1000ms total!
async function series() {
  await wait(500);
  await wait(500);
  return "done!";
}
```

```
// Would take only 500ms total!
async function parallel() {
  const wait1 = wait(500);
  const wait2 = wait(500);
  await wait1;
  await wait2;
  return "done!";
}
```

- Trigger both wait calls and then use await.
- Allows the async functions to happen concurrently

PROMISE.ALL()
EQUIVALENT

```
let [foo, bar] = await Promise.all([getFoo(), getBar()]);
```

ERROR HANDLING

This code:

```
1 async function f() {
2   await Promise.reject(new Error("Whoops!"));
3 }
```

...Is the same as this:

```
1 async function f() {
2   throw new Error("Whoops!");
3 }
```

We can catch that error using `try..catch`,

```
async function f() {

  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    alert(err); // TypeError: failed to fetch
  }
}

f();
```

ERROR HANDLING

We can catch that error using `try..catch`,

```
async function f() {
  let response = await fetch('http://no-such-url');
}

// f() becomes a rejected promise
f().catch(alert); // TypeError: failed to fetch // (*)
```

If there is an error, the exception is generated. (Same as if throw error was called)

SUMMARY

The $\boxed{\text{async}}$ keyword before a $\boxed{\text{function}}$ has two effects

Makes it always return a Promise

Allows to use $\boxed{\text{await}}$ in it