Next Gen Engineering
School of Tech - INDIA

Functional Programming (FP) 2

Async: Intro with Callbacks and Promises

Next Gen Engineering
School of Tech - INDIA

Overview

- Asynchronous code
- Callbacks
- Promises
- Creating a Promise
- Consuming a Promise

Next Gen Engineering
School of Tech - INDIA

Objectives

- Understand why we need callbacks
- Understand why we need promises
- Know how to create a promise
- Know how to consume a promise

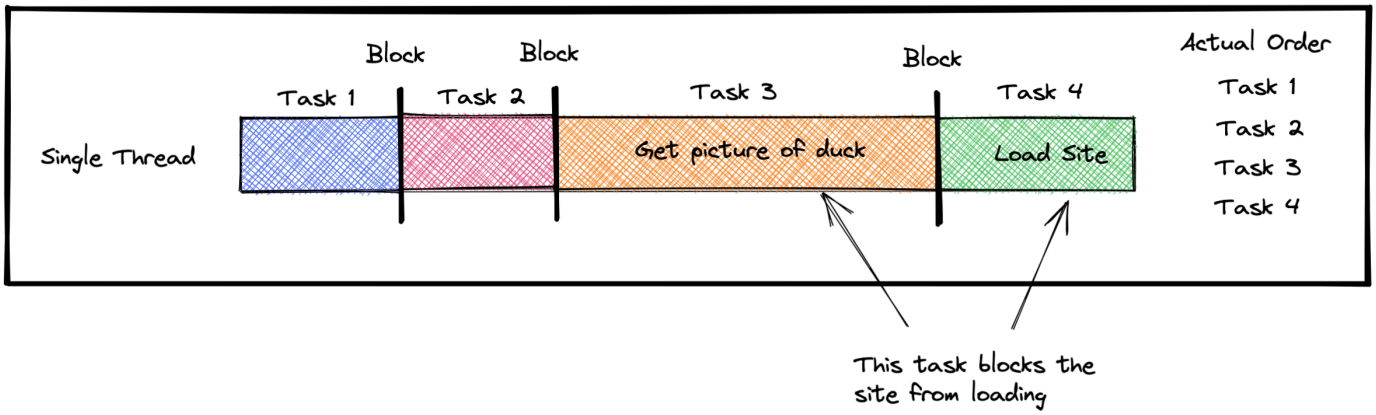
Next Gen Engineering
School of Tech - INDIA

The Problem

Node runs our code on a single-thread. This means a single stack of instructions are executed one after the other, sequentially or synchronously.

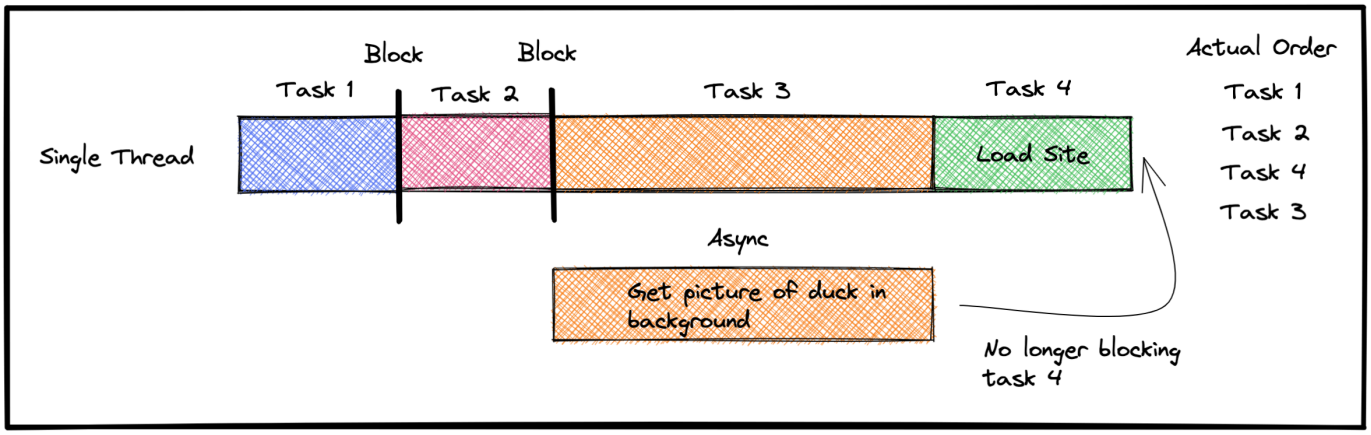
The Problem

This poses a problem. What if some of these instructions take some time to complete? Say, for example, fetching data from an API?
In a synchronous world all other instructions would be blocked while the fetch operation was running. Such as rendering, user-events etc...



The Problem

What if we could somehow mark an instruction as being asynchronous as to not block further instructions, and somehow interrupt or jump the queue once the async task was complete?



Callbacks

> ...Does anyone know what a callback is?

Callback concept

A callback function is one that

- we pass into another function for use later (i.e. literally `call me back later`)
- ...then after some delay (e.g. `1000ms`)
- is then "called back" by the receiver

An real-life example is turning on the kettle whilst you make toast; you can leave it going and come back to it once it's fully boiled.

Callback example

Some sample "callback" code could look like this:

```
const callback = () => console.log("I was called back")

setTimeout(callback, 1000)

console.log("Waiting...")
```

Callbacks

- This technique allows us to execute instructions asynchronously, and thus not block the next instruction
- The long-running code (`setTimeout`) does not stop the next line (`console.log`) from executing
- Once the async instruction has done its work, the callback will be executed

Task - Callback code

> Let's have a look at file `examples/example-callback.ts` together

- Open the file, run it, watch how the logs appear
- Note how the callback is a function

Hint:

- Run using `npx ts-node filename.ts` not the `Live Runner` plugin for all the examples in this session, as in these cases it gives a cleaner output

Emoji Check:

Do you understand what a callback is and the reasons we might use them?

1. 😢 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Next Gen Engineering
School of Tech - INDIA

Promises

Promises

Does anyone know what a promise is?
(In regular language, not necessarily programming terms)

Next Gen Engineering
School of Tech - INDIA

Promises

How do we know the async instruction is complete in order to execute the callback?
For that we need to use a promise.
Promises allow functions to return a result in the future once available.

Creating a Promise

A `Promise` constructor accepts a function with two arguments:

```
new Promise((resolve, reject) => {
  // Do some long task
})
```

- `resolve` lets us provide some data to resolve the promise with. This data is typically the result of the task.
- `reject` lets us provide a reason for the failure of the task, usually in the form of an error.

Creating a Promise

Both of these functions: `resolve` and `reject` are callback functions.

- In other words, we call back to `resolve` on success and `reject` on failure

Promises with resolve

```
const callback = () => console.log('Sending delivery!')

const asyncCookPizza = () =>
  new Promise((resolve) => {
    // Do something that takes ages, like cook a Pizza
    // Then
    resolve()
  })

asyncCookPizza().then(callback)

console.log("Waiting...")
```

Task - promise code

> Let's have a look at file `examples/example-promise.ts` together

- Open the file, run it, watch how the logs appear

Next Gen Engineering
School of Tech - INDIA

Emoji Check:

Do you understand what `Promises` are for and how to write them?

1. 😢 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Promise return values

Promises return the value of the `resolved` call into the next function in the chain.

```
promise1.then(nextFunction)
```

Which does something like this in the compiler:

```
promise1.then((result_of_promise1) => {
    nextFunction(result_of_promise1))
}
```

Consuming a Promise return value

We use `Promise.then` to pass the output value of a `resolved` promise to the next function of our choice:

```
const promiseMaker = (declaration) =>
  new Promise((resolve) => {
    resolve(`I promised to ${declaration} and I have`)
  })

const promise1 = promiseMaker("Order Pizza")

promise1.then(console.log)
// I promised to: Order Pizza and I have
```

Consuming a Promise return value

> We didn't have to pass anything to `console.log` manually, it used the fulfillment value of promise1 since we used `.then`

- The output of one promise is sent to the next function in the chain
- We usually use our own handler functions, rather than only `console.log`! (example in next task)

Task - promise return value

> Let's have a look at file `examples/consuming-promise-01.ts` together
> Then `examples/consuming-promise-02.ts`

- Open the file, run it, watch how the logs appear
- The second file is how we usually do things

Emoji Check:

How do you feel about Promise return values and how they are passed to the next function in the chain?

1. 😢 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Task - resolve and reject

> Let's have a look at file `examples/promise-resolve-reject.ts` together

- Open the file, run it, watch how the logs appear
- Switch the return value of `bakeAPizza` between `true` and `false`
- We start with the unhappy (false) case. What happens in the happy (true) case?

Emoji Check:

How do you feel about resolving promises?

1. 😥 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Next Gen Engineering
School of Tech - INDIA

Overview - recap

- Asynchronous code
- Callbacks
- Promises
- Creating a Promise
- Consuming a Promise

Next Gen Engineering
School of Tech - INDIA

Objectives - recap

- Understand why we need callbacks
- Understand why we need promises
- Know how to create a promise
- Know how to consume a promise

Next Gen Engineering School of Tech - INDIA

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Speaker notes