PROMISES

" If you promise something to someone, you either keep the promise or you break it "

In programming terms, the person who makes the promise can either use:

In programming terms, the person who makes the promise can either use:

resolve - they are going to keep the promise 🙂

In programming terms, the person who makes the promise can either use:

resolve - they are going to keep the promise 🙂

reject - they are going to break the promise 😑

If you were promised something, you can respond with:

If you were promised something, you can respond with:

then - the promise was kept 😀👍

If you were promised something, you can respond with:

then - the promise was kept 😀👍

catch - the promise was broken ☹️👎

Let's visualise this:

When things go well...

Person A 👩 is promised something by Person B 👨

When things go well...

Person A 👩 is promised something by Person B 👨

Person B 👨 fulfills the promise with `resolve()`

When things go well...

Person A 👩 is promised something by Person B 👨

Person B 👨 fulfills the promise with `resolve()`

Person A 👩 is happy, and uses `then()`

👩 💗 👨

When things go wrong...

Person A 👩 is promised something by Person B 👨

When things go wrong...

Person A 👩 is promised something by Person B 👨

Person B 👨 does not fulfill the promise and uses `reject()`

When things go wrong...

Person A 👩 is promised something by Person B 👨

Person B 👨 does not fulfill the promise and uses `reject()`

Person A 👩 is NOT happy, and uses `catch()`

👩 💔 👨

Why do we use promises, and where might we find them?

- When performing an asynchronous operation, for example when accessing resources from another server
- To handle errors in a better, more predictable way
- To help us write cleaner code

How do we create a promise in code?

How do we create a promise in code?

Introducing the `Promise` object

Just like `Date` or any ES6 class, we have to use `new` to instantiate the `Promise` object.

Just like `Date` or any ES6 class, we have to use `new` to instantiate the `Promise` object.

`Promise` is a constructor

A `Promise` needs a callback

A `Promise` needs a callback

The callback gives us 2 arguments, `resolve` and `reject`, which are both functions

A `Promise` needs a callback

The callback gives us 2 arguments, `resolve` and `reject`, which are both functions

```
const promise = new Promise((resolve, reject) => {});
```

Usually you will find the `Promise` object in the `return` statement of a function

Usually you will find the `Promise` object in the `return` statement of a function

```
function studyJavaScript() {
    const promise = new Promise((resolve, reject) => {});

    return promise;
}
```

Usually you will find the `Promise` object in the `return` statement of a function

```javascript
function studyJavaScript() {
    const promise = new Promise((resolve, reject) => {});

    return promise;
}
```

```javascript
function studyJavaScript() {
    return new Promise((resolve, reject) => {});
}
```

We use the `reject()` function to say "everything is NOT ok, we will NOT keep the `Promise`"

```
function studyJavaScript() {
    return new Promise((resolve, reject) => {
        reject();
    });
}
```

When we use `reject()` we should pass in an error

```javascript
function studyJavaScript() {
    return new Promise((resolve, reject) => {
        reject(new Error('No, I am too tired'));
    });
}
```

We use the `resolve()` function to say "everything is ok, we will keep the `Promise`"

```
function studyJavaScript() {
    return new Promise((resolve, reject) => {
        if(1 === 1) {
            resolve();
        } else {
            reject(new Error('No, I am too tired'));
        }
    });
}
```

Now we've seen how to create a `Promise`, how do we respond to one in code?

On the consumer side (the part of the code where we use the `Promise`) we can handle the `Promise` in one of two ways:

On the consumer side (the part of the code where we use the `Promise`) we can handle the `Promise` in one of two ways:

If the `Promise` was kept (fulfilled), we can run the method `then()` to mean "the promise was kept, let's do this"

On the consumer side (the part of the code where we use the `Promise`) we can handle the `Promise` in one of two ways:

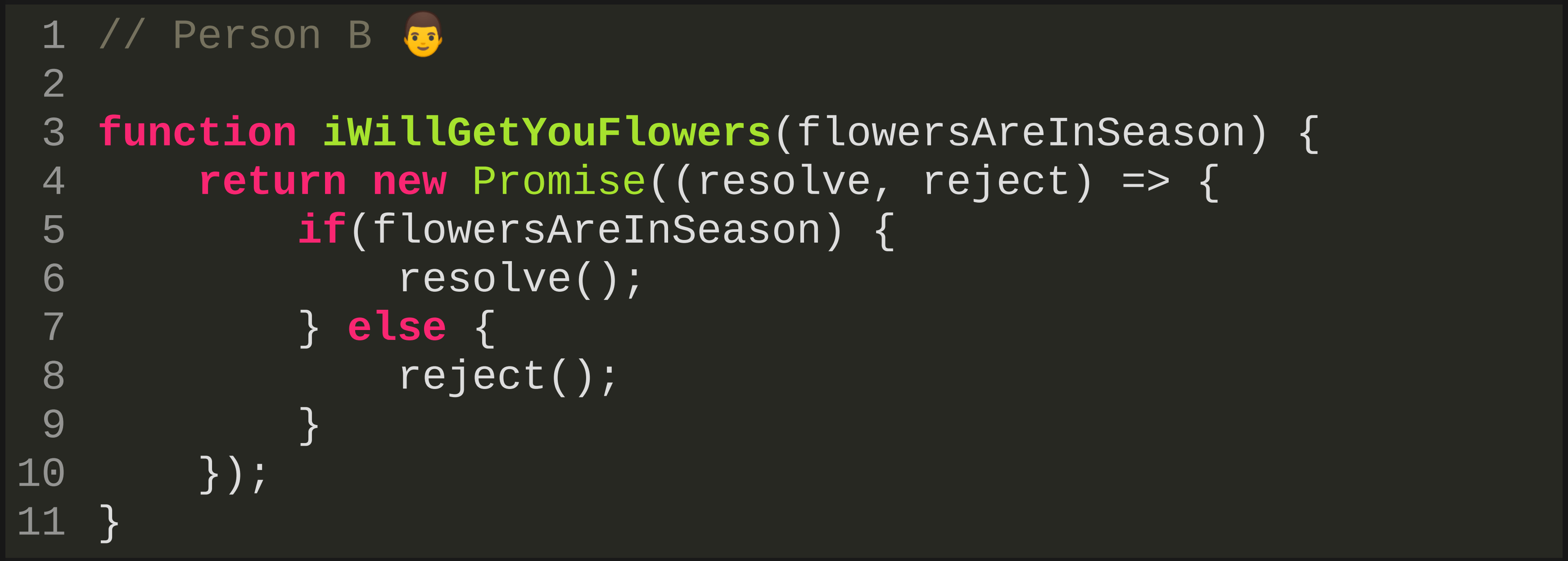
If the `Promise` was kept (fulfilled), we can run the method `then()` to mean "the promise was kept, let's do this"

If the `Promise` was broken (rejected), we can run some alternative code through the method `catch()` to say "the promise was NOT kept, let's do this instead"

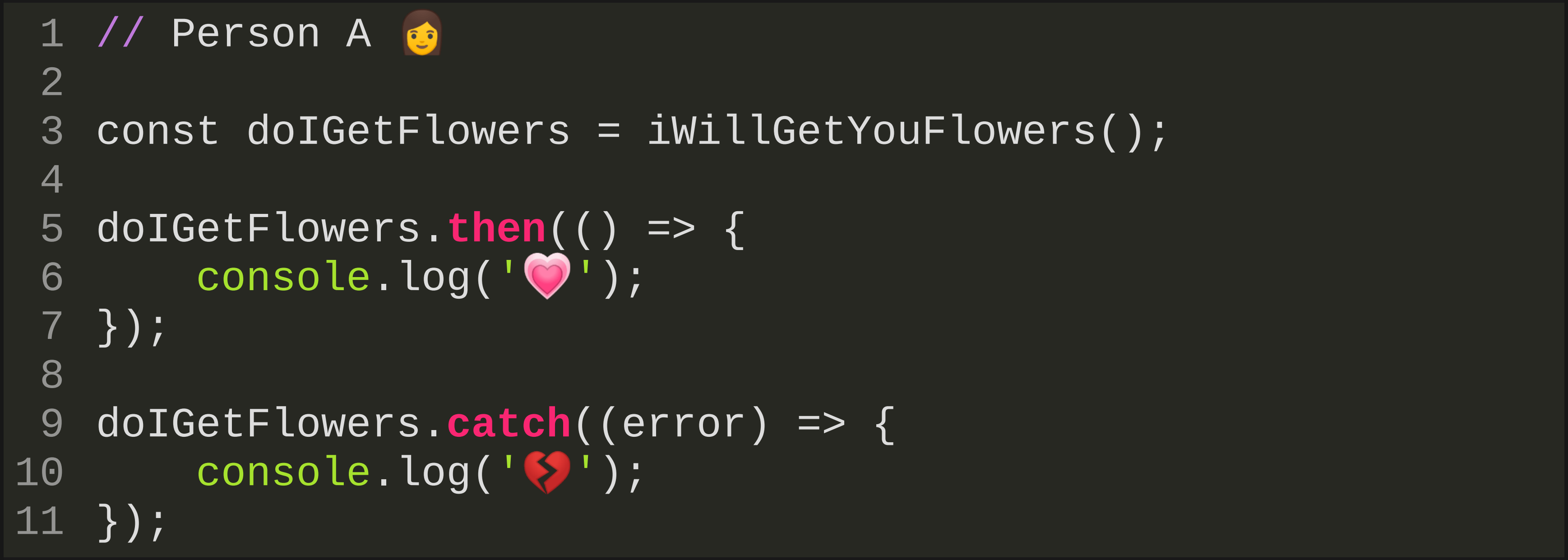
We use the `resolve()` function to say "everything is ok, we will keep the `Promise`"

```
const shouldIStudy = studyJavaScript();

shouldIStudy.then(() => {
    console.log("woohoo!")
});

shouldIStudy.catch(() => {
    console.log("NOOOOO!")
});
```

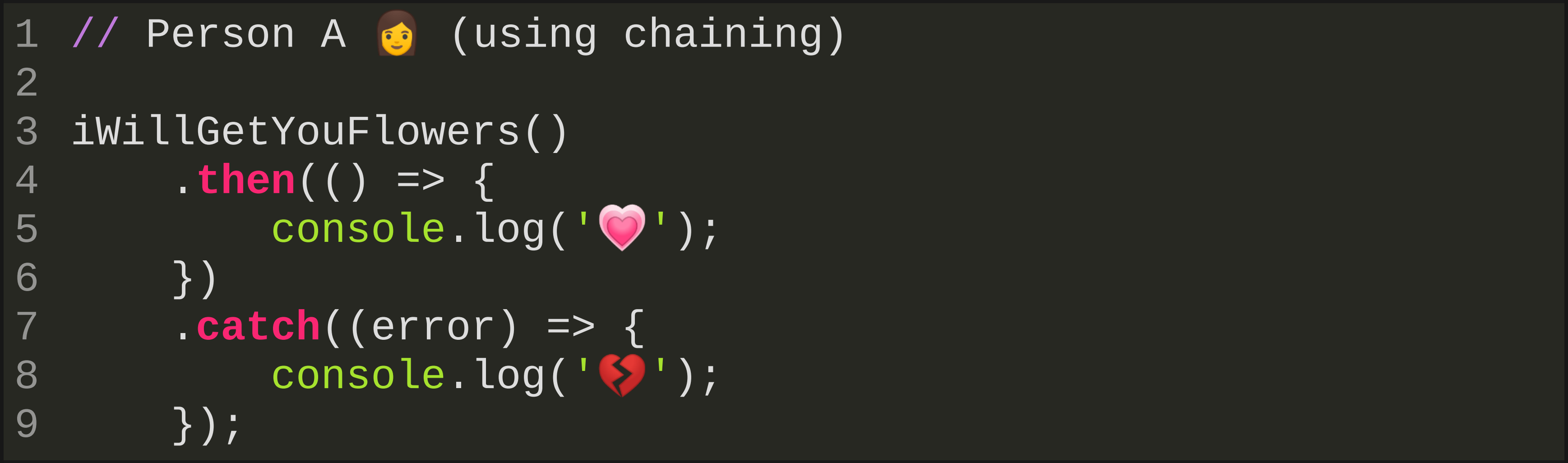
Let's look a full example in code:



```javascript
// Person B 👨

function iWillGetYouFlowers(flowersAreInSeason) {
    return new Promise((resolve, reject) => {
        if(flowersAreInSeason) {
            resolve();
        } else {
            reject();
        }
    });
}
```



```javascript
// Person A 👩

const doIGetFlowers = iWillGetYouFlowers();

doIGetFlowers.then(() => {
    console.log('💗');
});

doIGetFlowers.catch((error) => {
    console.log('💔');
});
```



```
// Person A 👩 (using chaining)

iWillGetYouFlowers()
    .then(() => {
        console.log('💗');
    })
    .catch((error) => {
        console.log('💔');
    });
```

A promise can be in one of 3 states:

pending - (waiting) the initial state. From here we can move to one of the other states

fulfilled - promise was kept

rejected - promise was NOT kept

Passing information with the `Promise`

👨 ➡️ 👩

Passing information with the `Promise`

👨 ➡️ 👩

When a `Promise` is either rejected or fulfilled, we can send some information back with it

For example, if we `reject()` a `Promise`, it might be useful to know why.

For example, if we `reject()` a `Promise`, it might be useful to know why.

```
function iWillGetYouFlowers(flowersAreInSeason) {
    return new Promise((resolve, reject) => {
        if(flowersAreInSeason) {
            resolve();
        } else {
            reject(new Error('There are no flowers'));
        }
    });
}
```

If we `resolve()` a `Promise`

If we `resolve()` a `Promise`

```
function iWillGetYouFlowers(flowersAreInSeason) {
    return new Promise((resolve, reject) => {
        if(flowersAreInSeason) {
            resolve('You get a tulip');
        } else {
            reject(new Error('There are no flowers'));
        }
    });
}
```

If we resolve( ) a Promise

```javascript
function iWillGetYouFlowers(flowersAreInSeason) {
    return new Promise((resolve, reject) => {
        if(flowersAreInSeason) {
            resolve('You get a tulip');
        } else {
            reject(new Error('There are no flowers'));
        }
    });
}
```

```javascript
const doIGetFlowers = iWillGetYouFlowers();

doIGetFlowers.then((message) => {
    console.log('💗', message); // 'You get a tulip'
});
```