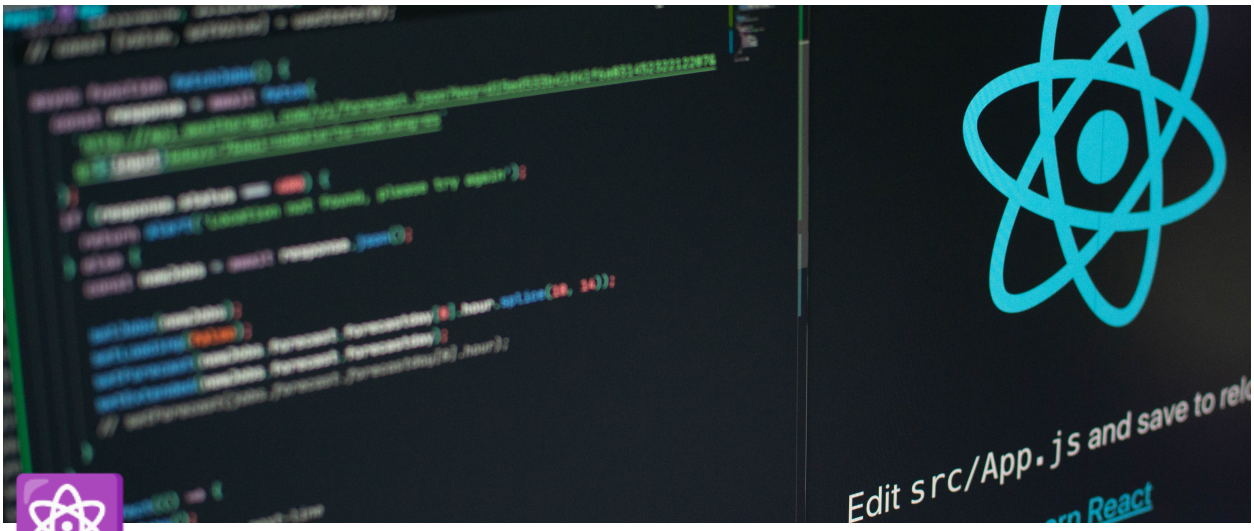


Javascript for React Q&A

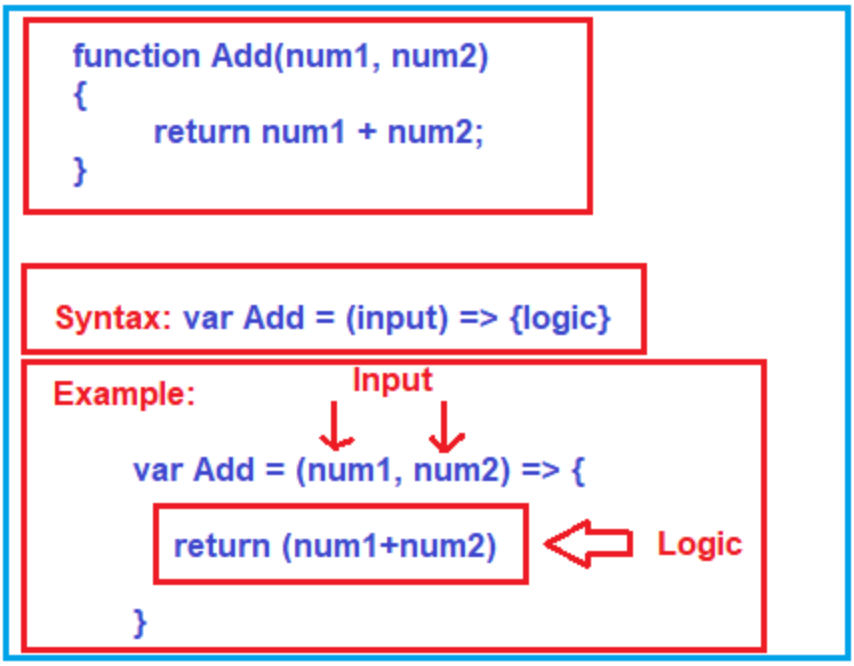
Tags

✅ **List of Common Javascript Syntax for React**

- Arrow functions
- Array destructuring / Object destructuring
- Spread operator / Rest operator
- Ternary conditions
- Template literals
- Closure & Callbacks
- Error handling: Try/Catch
- Promises
- Asynchronous functions: async/await
- Node.js basics

1. Arrow Functions

Transforming “regular functions” into one-line arrow function expressions:



2. Array / Object Destructuring

See Medium article.


```
let introduction = ["Hello", "I" , "am", "Sarah"];
let [greeting, pronoun] = introduction;

console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name, country, job} = person;

console.log(name);//"Sarah"
console.log(country);//"Nigeria"
console.log(job);//Developer"
```

How to use array and object destructuring in Javascript

by FreeCodeCamp

3. Spread & Rest Operators

- both operators use `(...)` syntax

Main difference between spread & rest operators

Rest Operator:

- Will return rest of some specific user-supplied values into a JavaScript array
- Collects multiple elements and condenses them into a single element
- Super useful when defining a function that can have an indefinite number of arguments

```
// Use rest to enclose the rest of specific user-supplied values into an array:
function myBio(firstName, lastName, ...otherInfo) {
  return otherInfo;
}

// Invoke myBio function while passing five arguments to its parameters:
myBio("Oluwatobi", "Sofela", "CodeSweetly", "Web Developer", "Male");

// The invocation above will return:
["CodeSweetly", "Web Developer", "Male"]
```

- `firstName` is assigned to `"Oluwatobi"`
- `lastName` is assigned to `"Sofela"`
- `otherInfo` is assigned to `"CodeSweetly", "Web Developer", "Male"`
- `return otherInfo` will return `["CodeSweetly", "Web Developer", "Male"]`

Spread Operator:

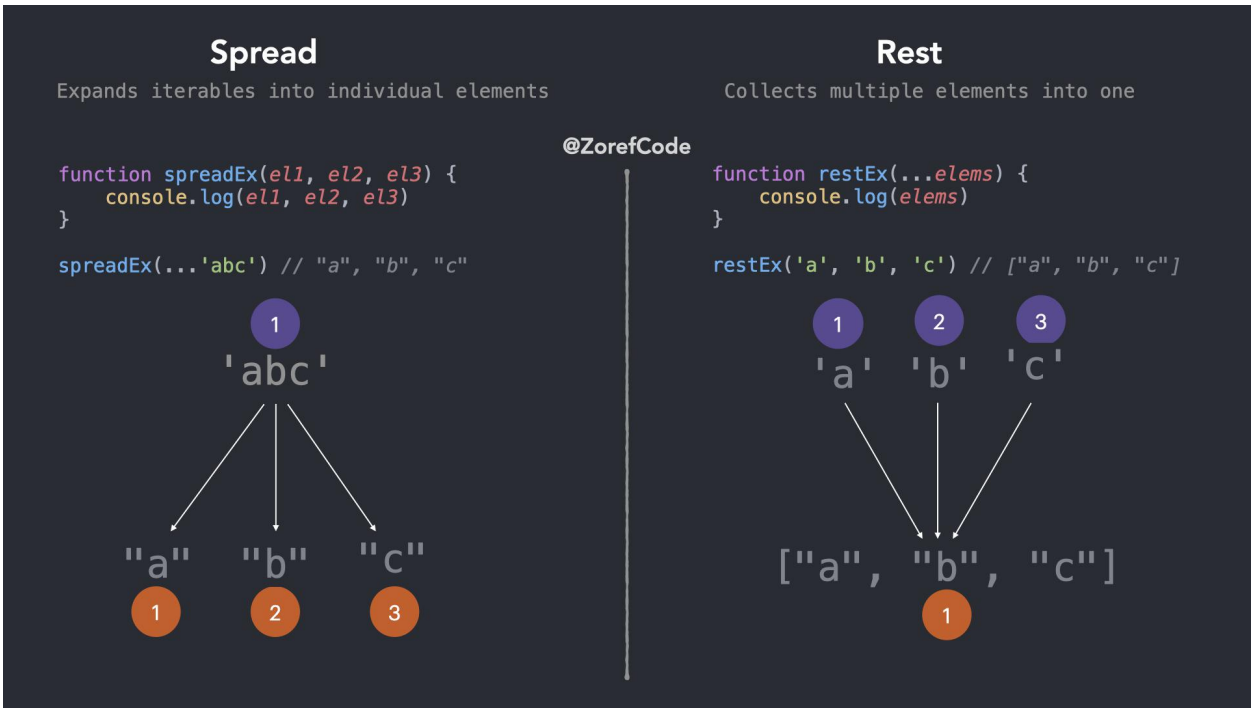
- Allows us to quickly copies all of part of an existing array or object into another array or object

- Allows an iterable (array, string, object) to be expanded where more arguments/elements are expected
- Adds array elements to another array
- Passes elements of an array as arguments to a function
- Copies an array

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];

console.log(numbersCombined)

1,2,3,4,5,6
```

- It is also used in combination with destructuring

```
const numbers = [1, 2, 3, 4, 5, 6];

const [one, two, ...rest] = numbers;

console.log(one)
console.log(two)
console.log(rest)

1
2
3,4,5,6
```



Video:

https://www.youtube.com/watch?v=oob7bxg-Ctc

TERNARY CONDITIONS

- a way to simplify expressions that will return one of two possible answers based on a conditional
- specifically for returning values that aren't booleans

```
if conditional "a" is met:
  return "answer1"
else:
  return "answer2"
```

examples in `javascript`:

```
function evenoff(num) {
  return num % 2 === 0 ? 2 : 1;
}
```

examples in `python`:

```
def evenodd(num):
  return 2 if num % 2 == 0 else 1
```

Template literals

We use template literals, because it is an easy way to interpolate variables and expressions into strings.

in·ter·po·late
verb
insert (something of a different nature) into something else.

Template literals are essentially the JavaScript version of formatted strings in Python. You must surround your template literal with back ticks (`) . You can then include variables by surrounding the variable with ${} to replace those variables with their values in the template literal. It can include multiple lines, as shown in the picture below where it kept the new line.

▼ Ex:

```
const a = 5;
const b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and
// not 20."
```

▼ Ex. from code

```
function createCard(name, description, pictureUrl, start, end, location) {
    return `
    <div class="col-sm-6 col-md-4 mb-1">
        <div class="shadow-lg p-3 mb-5 bg-body rounded">
            <img src="${pictureUrl}" class="card-img-top">
            <div class="card-body">
                <h5 class ="card-title">${name}</h5>
                <h6 class="card-subtitle mb-2 text-muted">${location}</h6>
                <p class="card-text">${description}</p>
            </div>
            <div class="card-footer">
                <small class="text-muted">${start}-${end}</small>
            </div>
        </div>
    </div>
    `;
}
```

Callbacks and Closures:

1. Callbacks are functions that are passed into another function as an argument. — John Au Yeung
2. Closures are functions that are nested in other functions, and it's often used to avoid scope clash with other parts of a JavaScript program. — John Au Yeung

3. Closure is a persistent scope which holds onto local variables even after the code execution has moved out of the block. Closures can be produced in JavaScript, Swift, and Ruby. They allow you to keep a reference even after the block in which those variables were declared and finished executing, provided you refer to the variable later on in the function

```
outer = function() {
  var a = 1;
  var inner = function() {
    console.log(a);
  }
  return inner; // this returns a function
}

var fnc = outer(); // execute outer to get inner
fnc();
```

https://stackoverflow.com/questions/36636/what-is-a-closure

```
function greeting(name) {
  alert(`Hello, ${name}`);
}

function processUserInput(callback) {
  const name = prompt("Please enter your name.");
  callback(name);
}

processUserInput(greeting)
```

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

https://www.w3schools.com/js/js_callback.asp

Error handling: Try/Catch

Error handling, "try...catch"

No matter how great we are at programming, sometimes our scripts have errors. They may occur because of our mistakes, an unexpected user input, an erroneous server response, and for a thousand other reasons.

Usually, a script “dies” (immediately stops) in case of an error, printing it to console.

But there’s a syntax construct `try...catch` that allows us to “catch” errors so the script can, instead of dying, do something more reasonable.

It works like this:

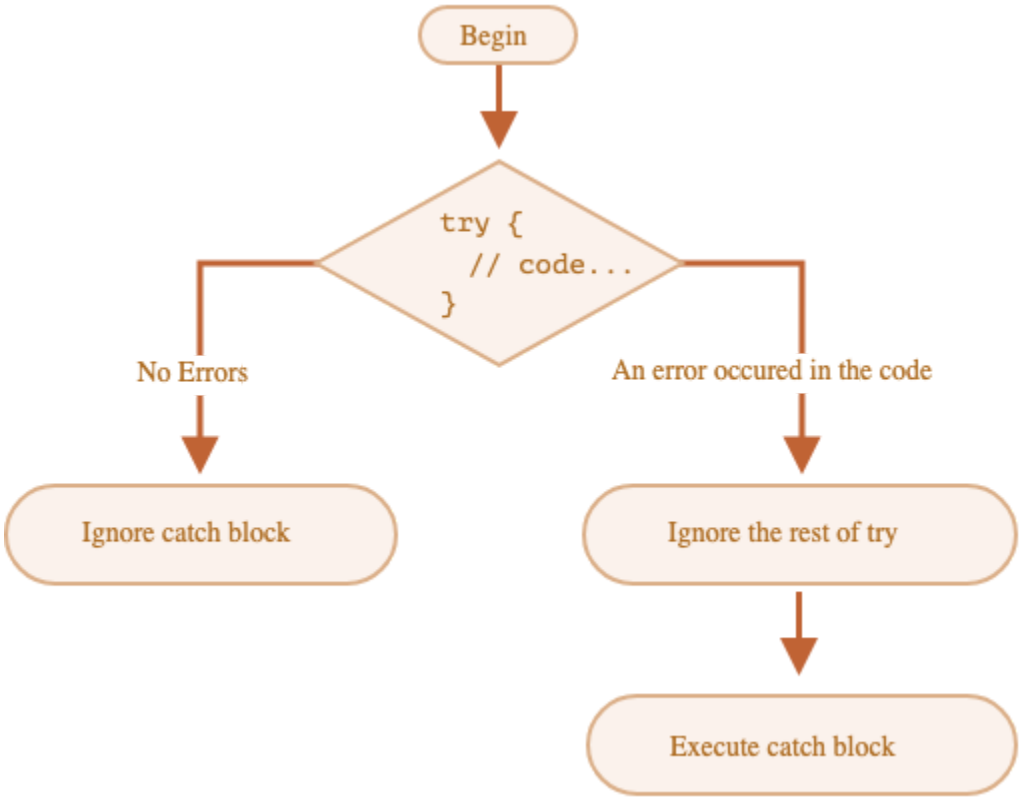
1. First, the code in `try {...}` is executed.
2. If there were no errors, then `catch (err)` is ignored: the execution reaches the end of `try` and goes on, skipping `catch`.
3. If an error occurs, then the `try` execution is stopped, and control flows to the beginning of `catch (err)`. The `err` variable (we can use any name for it) will contain an error object with details about what happened.

The `try` statement defines a code block to run (to try).

The `catch` statement defines a code block to handle any error.

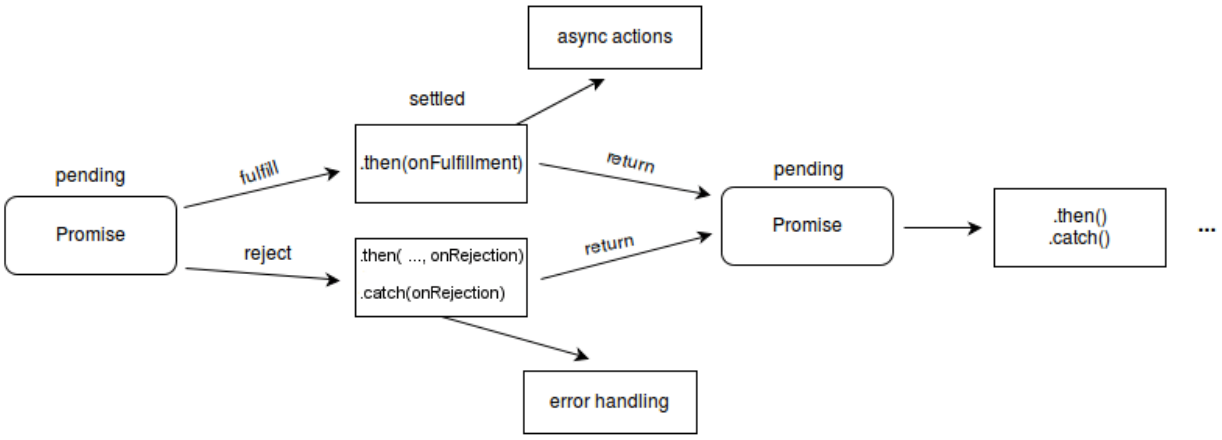
The `throw` statement throws a user-defined exception.

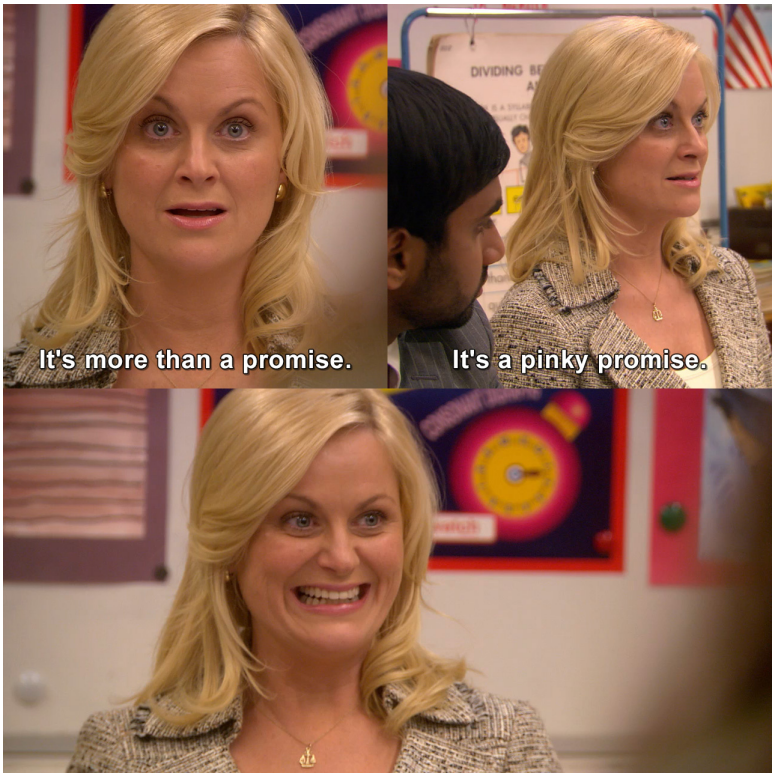
```
try {
    const response = await fetch(URL); //

    if (!response.ok) {
      throw "Invalid Response received"
    } else {
      const data = await response.json();
    }
  } catch (e) {
    console.log(`Error was received during conference fetching stage: ${e}`)
  }
```



8. Promises

- A promise is an object representing the eventual completion or failure of an asynchronous operation and its resulting value object
- A promise is a special object that links the producing code (the data from a function) to the consuming code together
- Promise objects can be fulfilled, pending, or rejected





9. Asynchronous functions: async/await

💡 asynchronous programming is a technique that allows the program to begin work that could potentially take a while to complete while still responding to other events without having to wait until that task is complete

Easier Promise Code

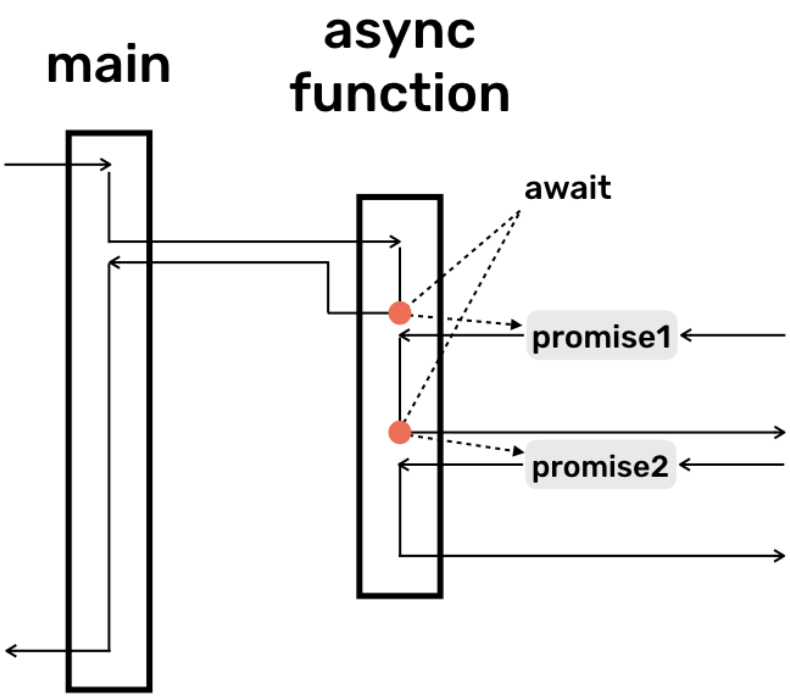
`await` keyword

- wait for a `Promise` object to either resolve with a value or reject with an error

`async` keyword

- whenever you have the `await` keyword inside a function, you have to mark that function as an `async` function

- **Inside an async function, you can use the await keyword before a call to a function that returns a promise**



1. **Node.js basics**

a. Node is an environment to execute code. It allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionalities.

b. The advantage of Node is that it allows developers to write JS for browser (server side and client side) without having to learn a completely different language

c. **example of Node in action****

d. what is used to install a package: “npm install” - “node package manage” (sort of like pip for python)

    i. runs everything in your package.JSON (sort of like requirements.txt)