# Callback, promises

Partha Das Chowdhury



#### **Basics**

- · JavaScript has synchronous execution
- They execute line by line
- In the real world we need to wait sometimes before we do something else.
- This should not stop the rest of our program from executing.
- There we need asynchronous execution too. So, we have <u>callbacks</u> & <u>promises</u>

## Callbacks

- Let's give the responsibility of calling a function to another function.
- This is useful when you want to call a function or do something post completion of another function/task.

#### Callbacks

```
setTimeout(function(){
    console.log("I will come after 5 seconds");
},5000)
function goFirst(callback){
    console.log("Hello World \n");
    callback();
function goSecond(){
    console.log("goFirst is calling me when it wants");
goFirst(goSecond);
```

## Callback Hell or Pyramid of doom

```
setTimeout(function(){
    console.log("I will come after 5 seconds");
       setTimeout(function(){
          console.log("I will also come after 5 seconds");
            setTimeout(function(){
              console.log("I will also come after 5 seconds");
            },5000)
       },5000)
},5000)
```

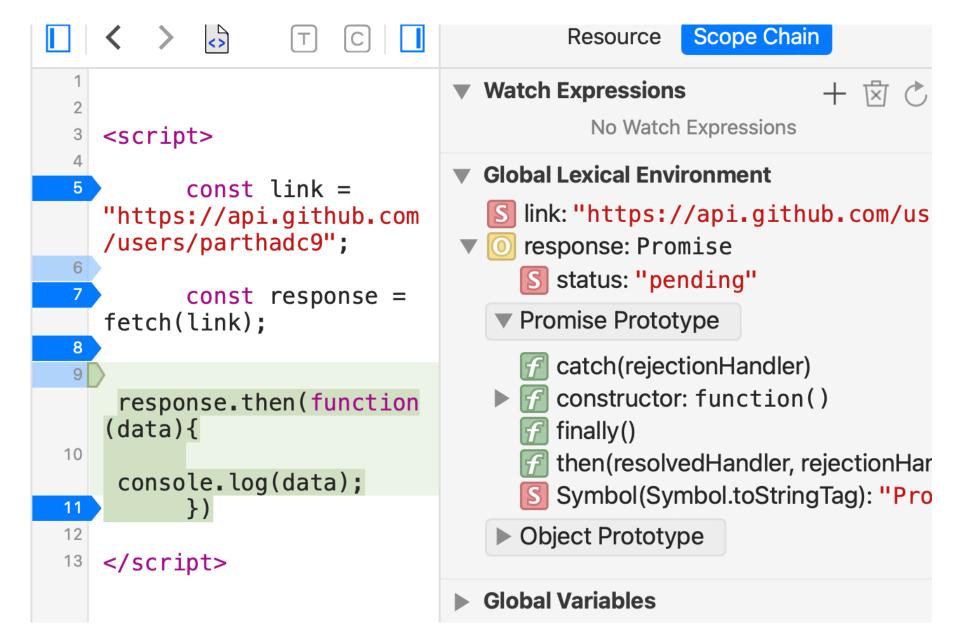
# Callback Hell or Pyramid of doom

- Say we have code from third parties
- We make multiple database requests
- · Callbacks can lead to losing control of our code.
- · Callbacks might not do what we want our code to do.
- · Losing trust of our code.

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. (MDN Documentation)

- Promises are used to handle asynchronous operation.
- Asynchronous means we are dependent on the user or some other task to finish
  - For example, if a user is browsing and choosing items to buy on your site.
- · Promises can be pending, fulfilled and rejected
- · Promise objects are immutable

```
const link = "https://api.github.com/users/parthadc9";
const response = fetch(link);
response.then(function(data){
  console.log(data);
```



```
const response = fetch();
response.then(function(){
   //remember to return
then (function(){ // OR then (returnedvalue => next function())
   //remember to return
```

#### For Each

```
const numbers = [1, 2, 3, 4, 5];
for (i = 0; i < numbers length; i++) {
  console.log(numbers[i]);
numbers.forEach(function(number) {
    console.log(number);
});
numbers.forEach(number => console.log(number));
```

#### For Each

```
const numbers = [1, 2, 3, 4, 5];
      numbers.forEach((number, index, array) => {
 3
          console.log('Index: ' + index + ' Value: ' + number);
 4
      });
PROBLEMS OUTPUT
                    DEBUG CONSOLE
                                    TERMINAL
```

ja21121@C02DWCVPML7H Teaching 2023 % node foreach.js Debugger attached. Index: 0 Value: 1

Index: 1 Value: 2 Index: 2 Value: 3

Index: 3 Value: 4

Index: 4 Value: 5

