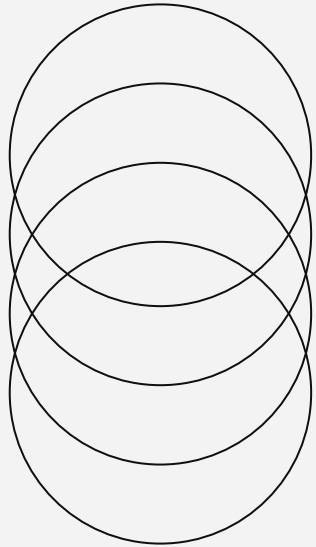
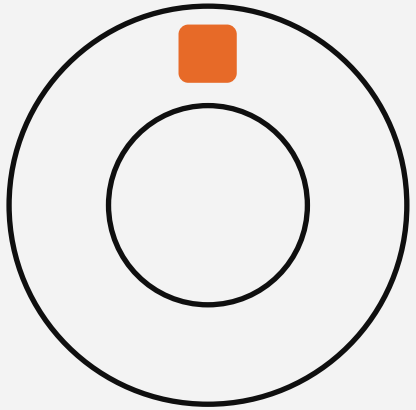
Асинхронно програмиране с TypeScript (JavaScript)

Изготвена от:
Мартин Менчев, Кирил Димов, Денислав Манахов



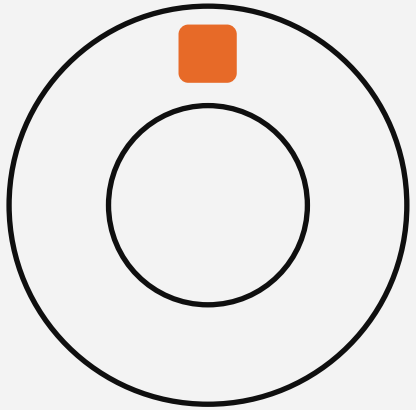
СЪДЪРЖАНИЕ

01

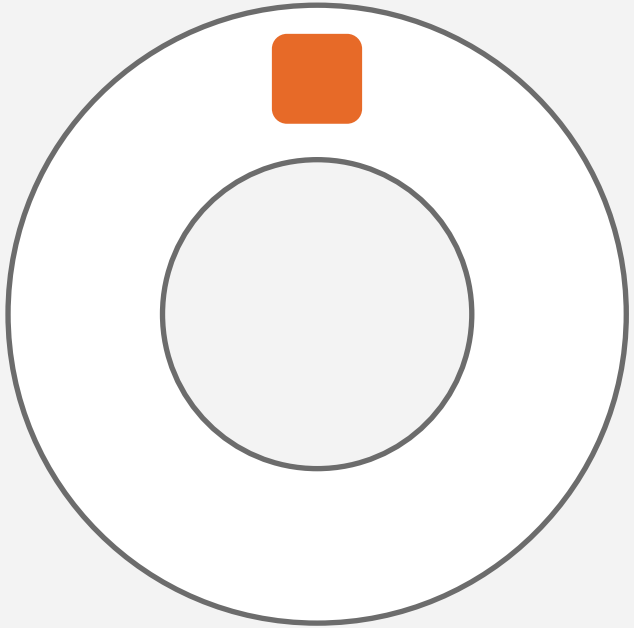


Какво е Event Loop?

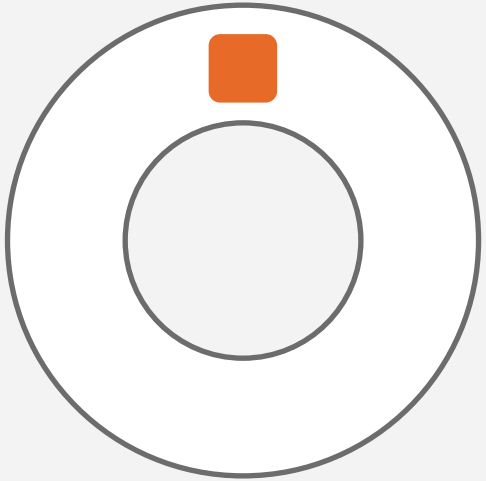
01

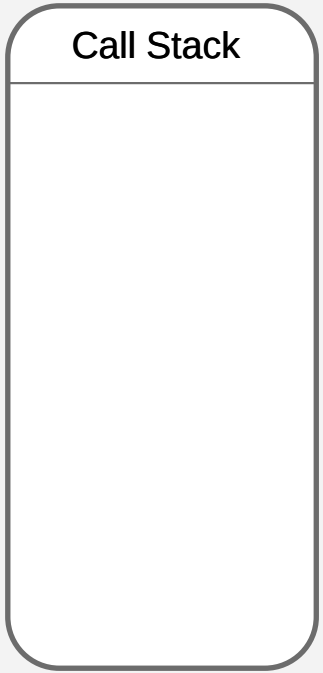


Какво предствлява цикъла на събития?

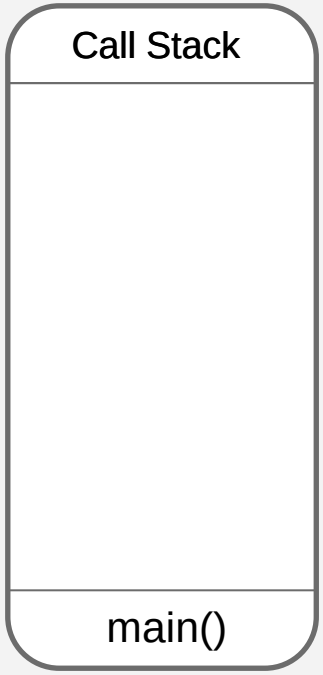


Call Stack

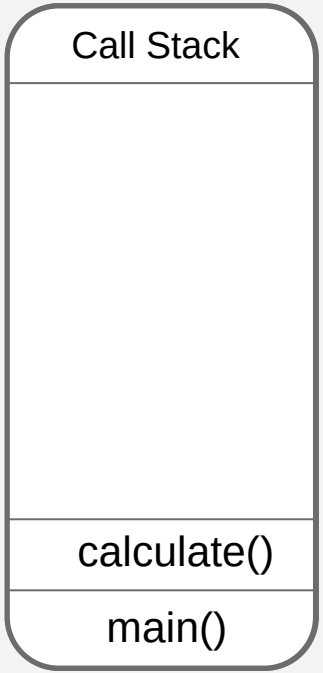




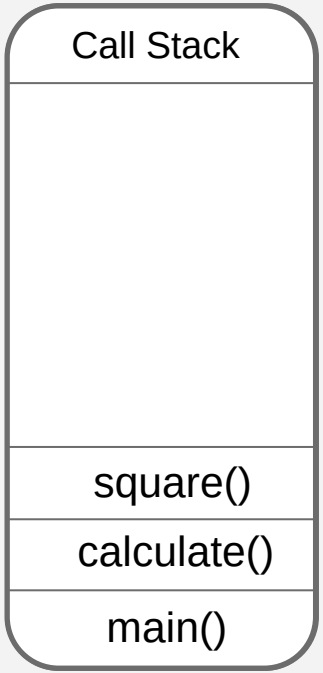

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



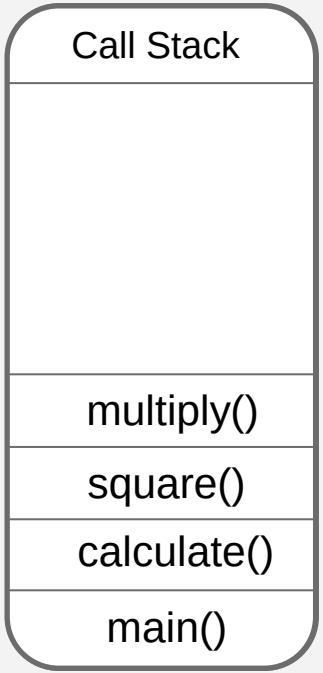

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



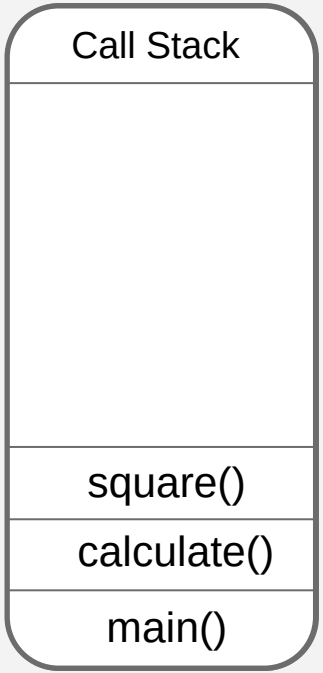

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



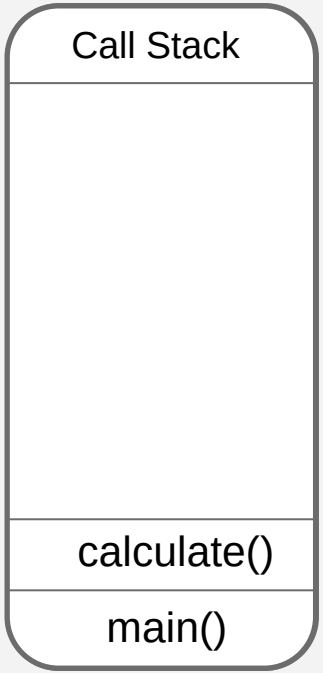

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



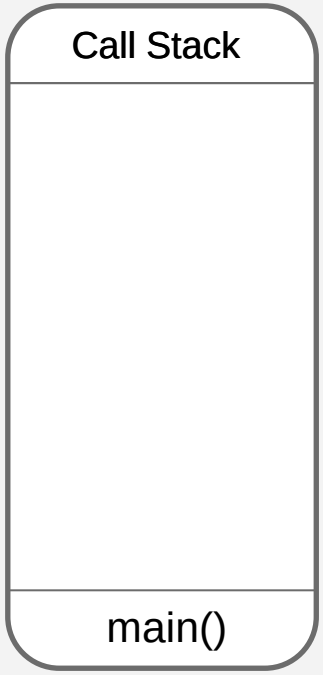

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



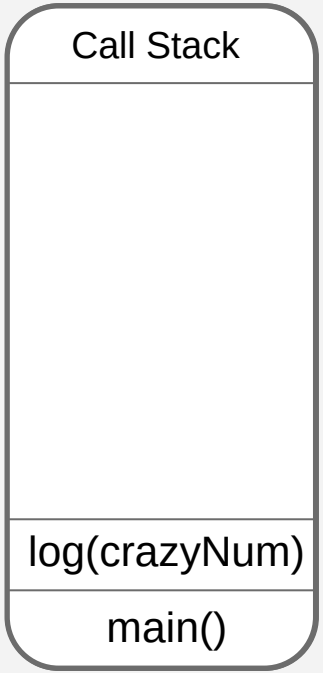

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



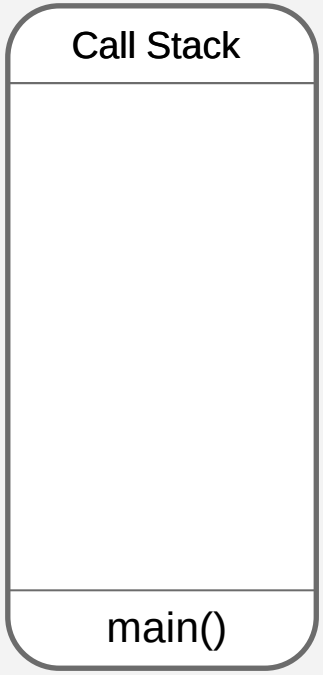

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



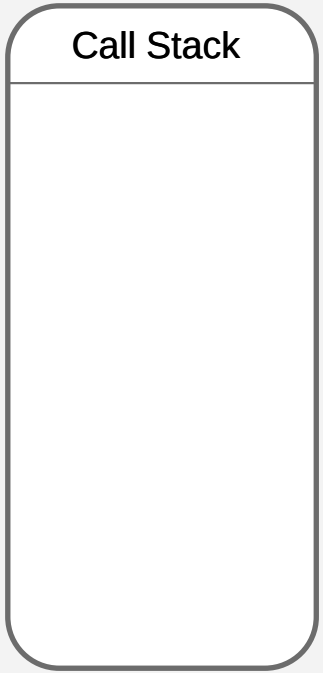

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



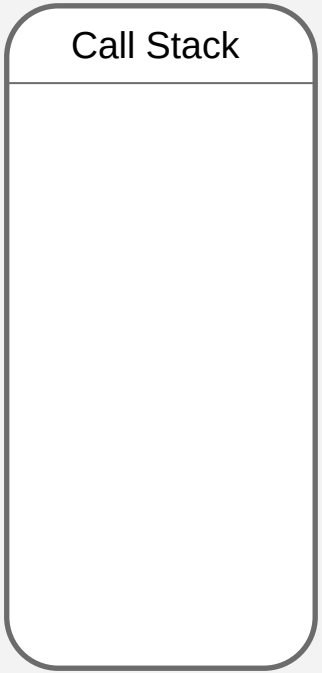

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```

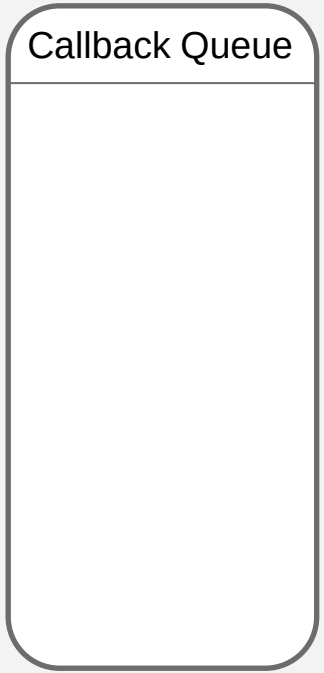



```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```



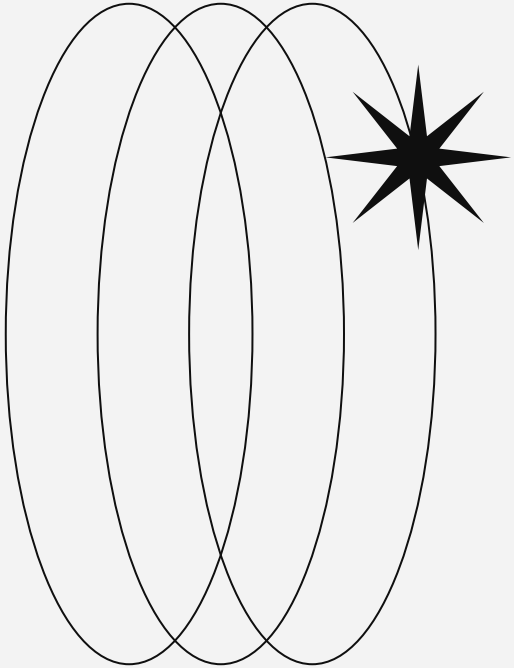

```
function multiply(num1: number, num2: number) {
    return num1 * num2;
}

function square(num: number) {
    return multiply(num, num);
}

function calculate(num: number) : number {
    let squaredNumber: number = square(num);
    return squaredNumber / 2;
}

const crazyNum = calculate(10);
console.log(crazyNum);
```


Call Stack


Callback Queue

02

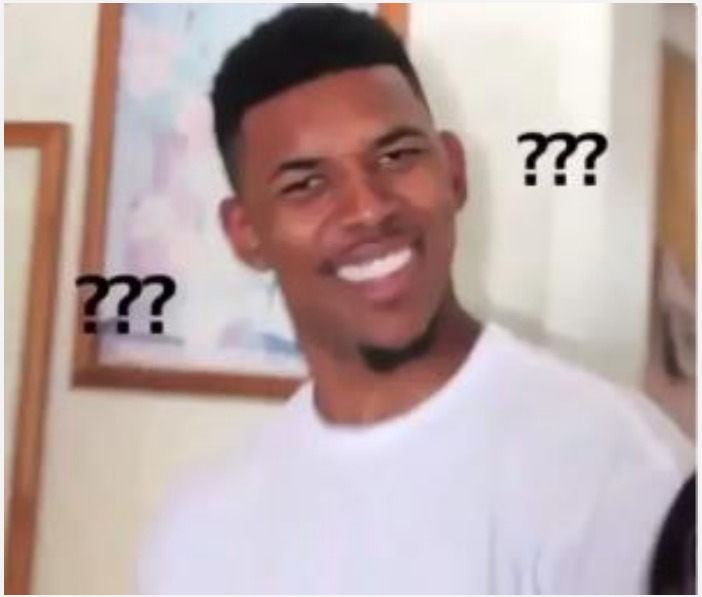
Обратни повиквания. Callbacks



Какво беше callback?

```
function callAndPrint(cb : () => void) : void {
    cb()
    console.log("Called the callback!")
}

function printMessage() : void {
    console.log("I like turtles")
}

callAndPrint(printMessage);
```

Как така обратните повиквания са свързани с асинхронното програмиране в TypeScript(JavaScript)



Асинхронни функции

Асинхронни функции

Класически примери са: setTimeout и setInterval

```
setTimeout(callback: () => void, ms?: number | undefined)
setInterval(callback: () => void, ms?: number | undefined)
```

Асинхронни функции

Класически примери са: setTimeout и setInterval

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 2000);
console.log("Third");
```

Асинхронни функции

Класически примери са: setTimeout и setInterval

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 2000);
console.log("Third");
```

```
First
Third
Second
```

Асинхронни функции

Класически примери са: setTimeout и setInterval

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 0);
console.log("Third");
```

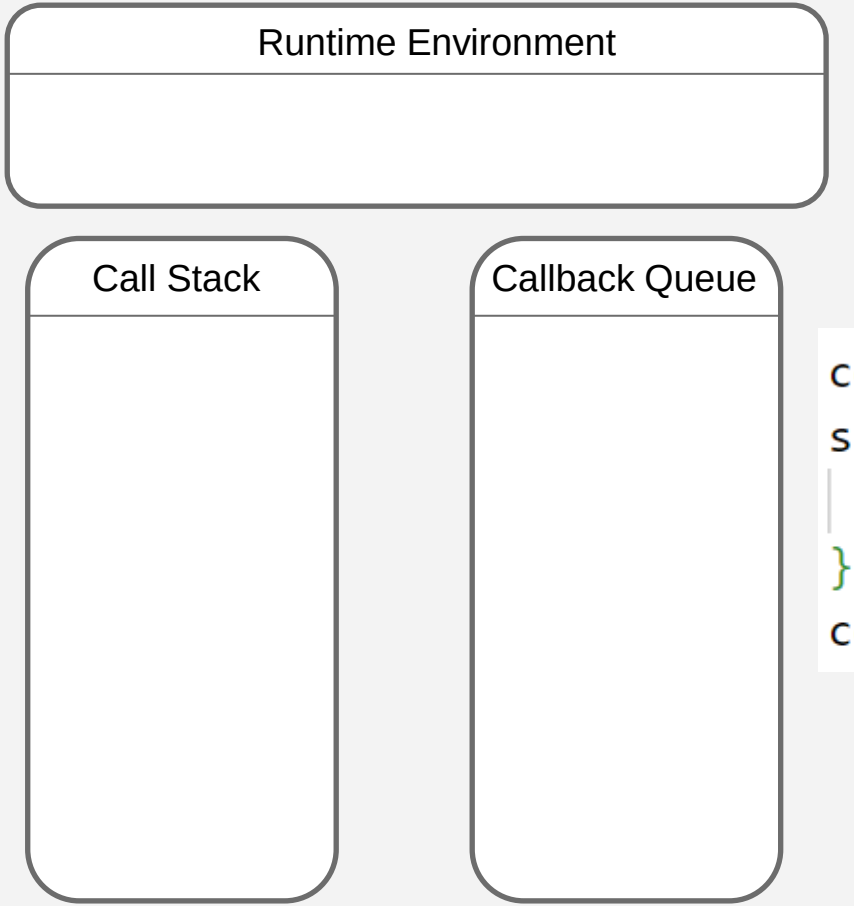
Асинхронни функции

Класически примери са: setTimeout и setInterval

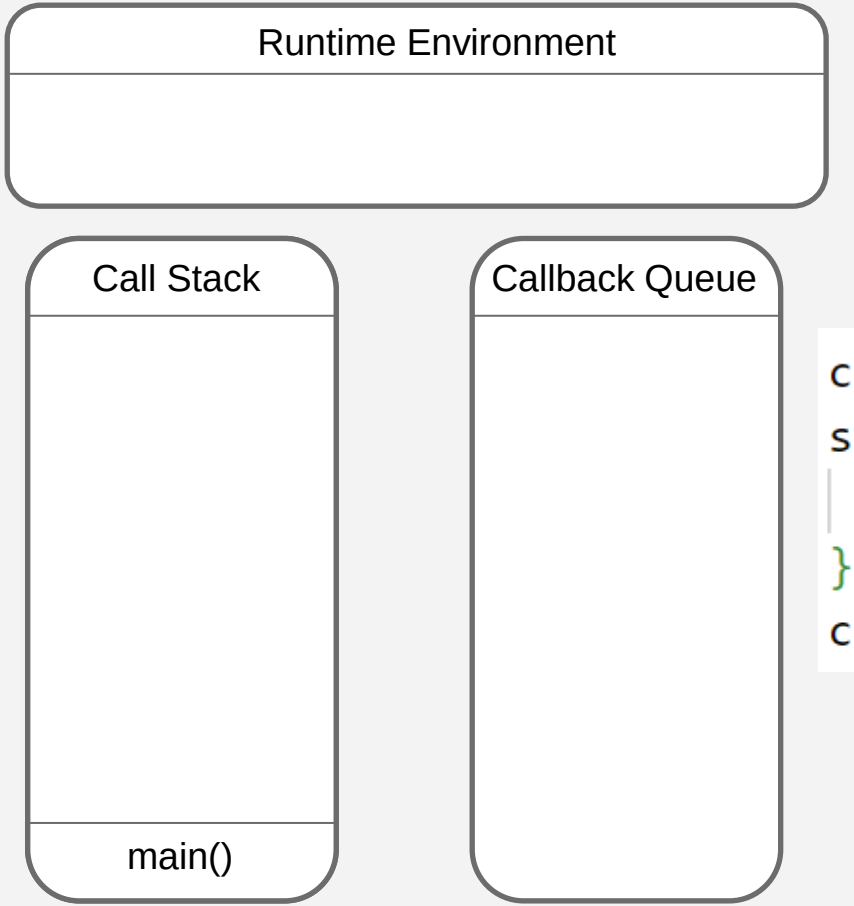
```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 0);
console.log("Third");
```

```
First
Third
Second
```

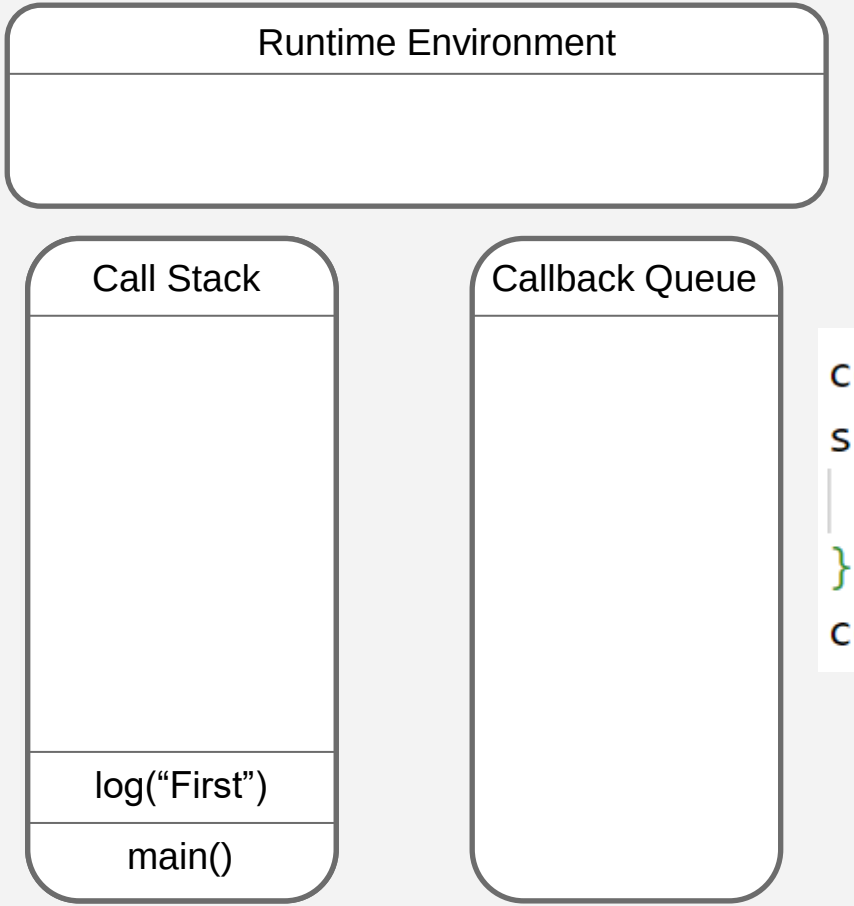
Ама нали TypeScript(JavaScript) не може да работи на няколко нишки!? Какво се случва?



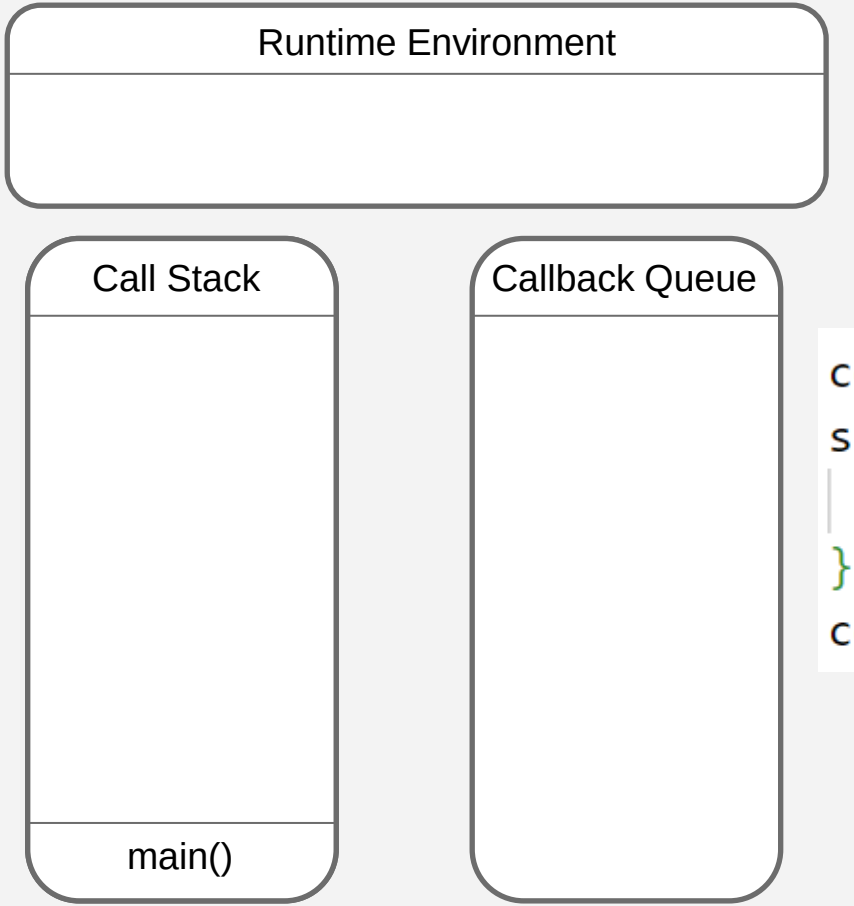

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



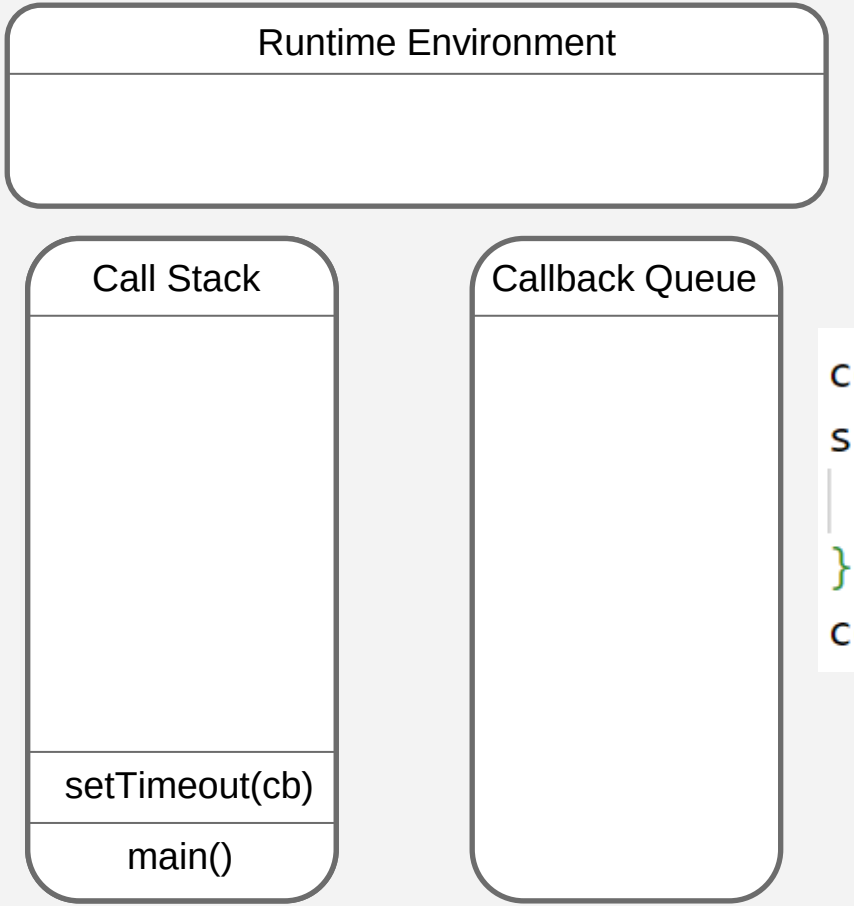

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



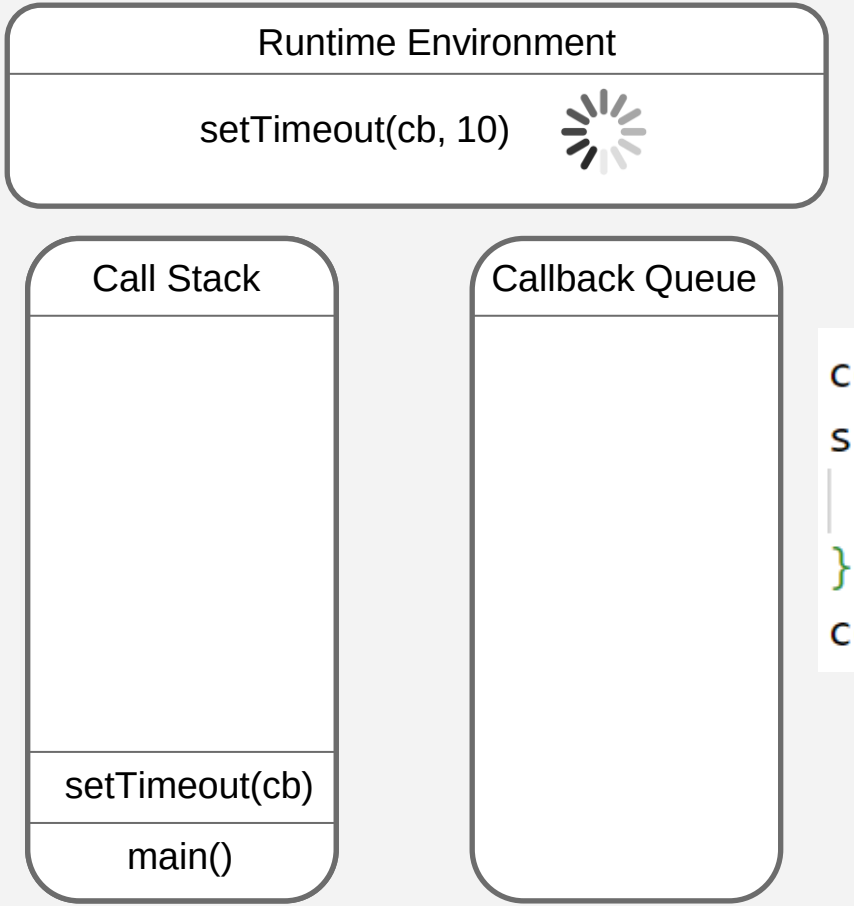

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



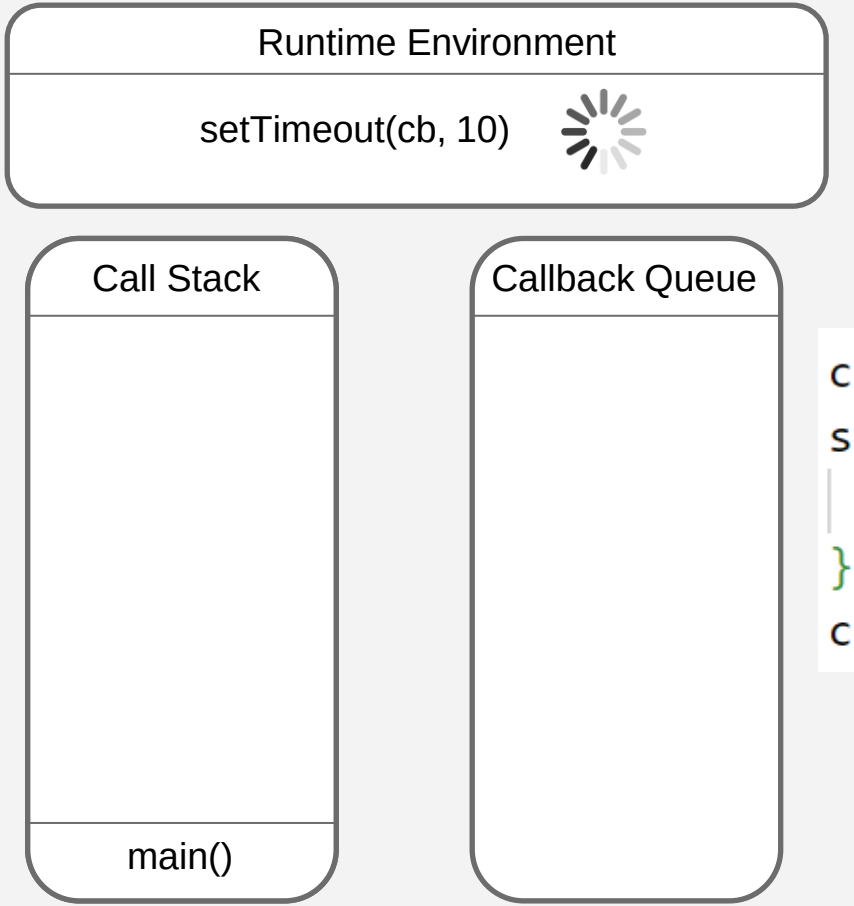

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



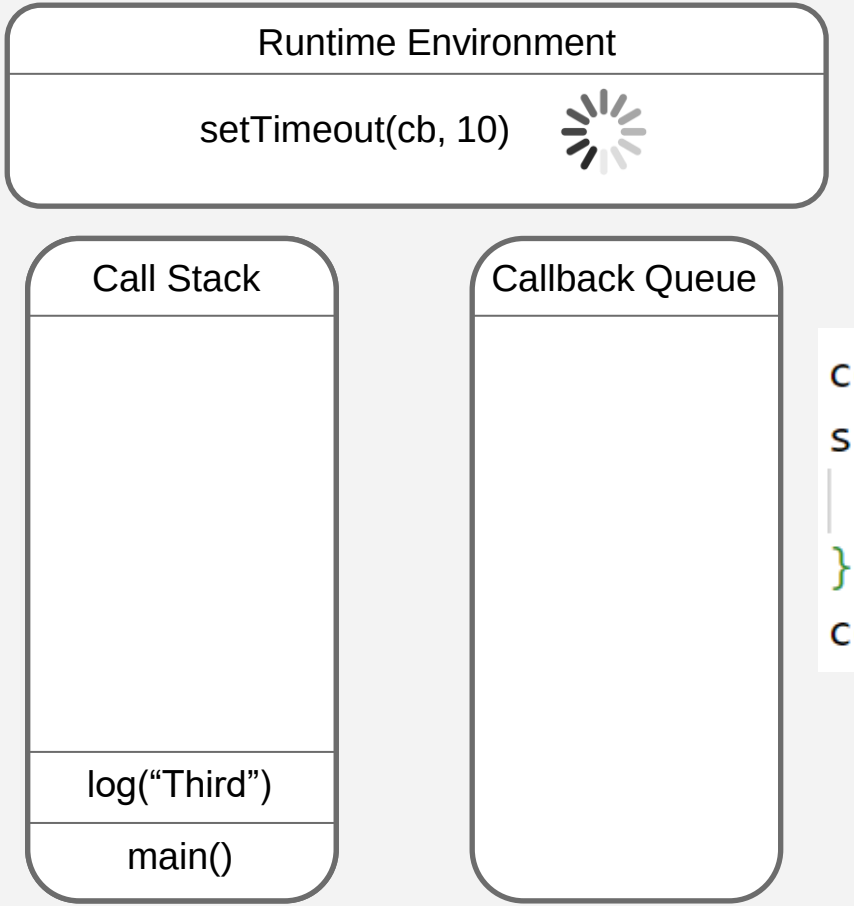

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



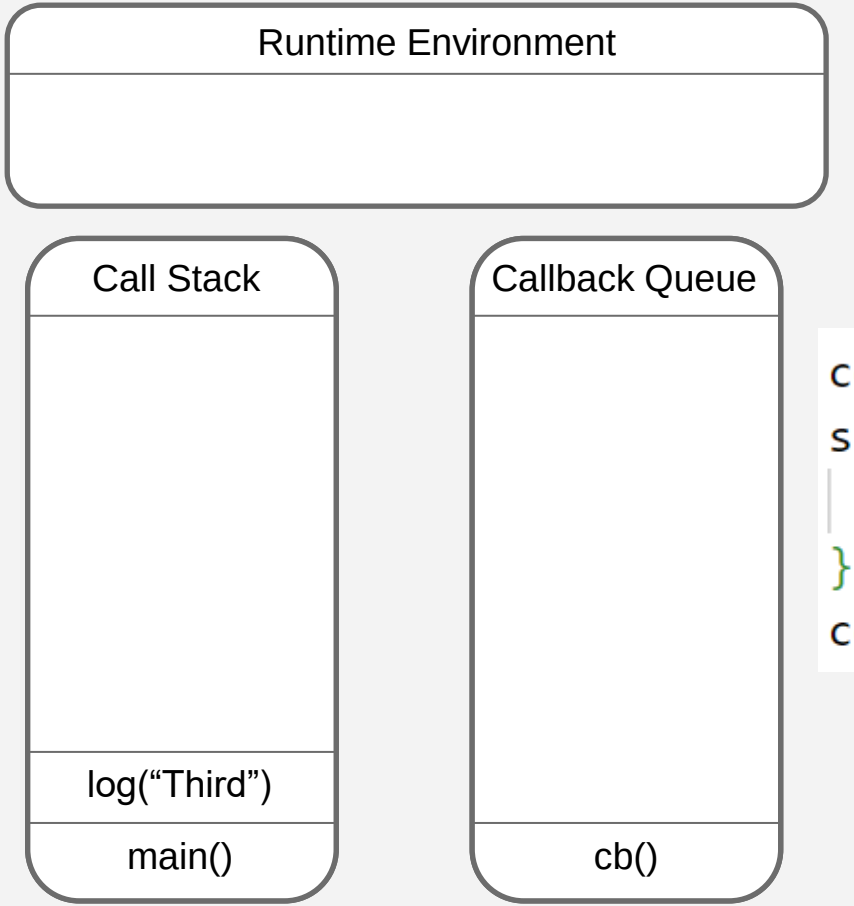

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



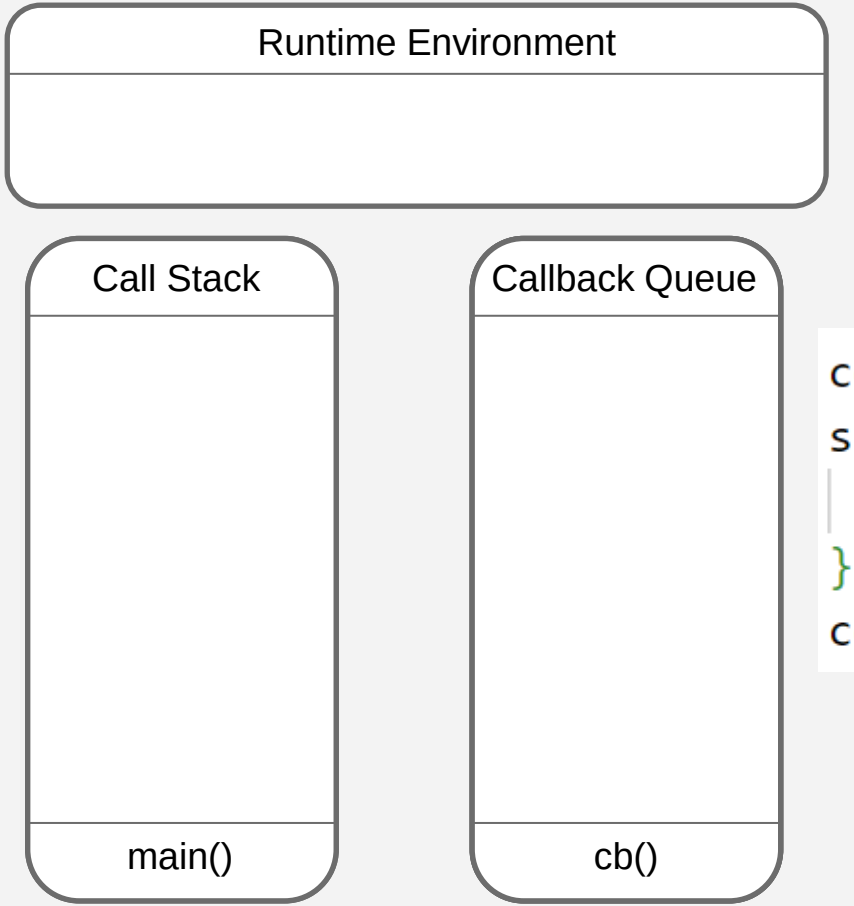

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```



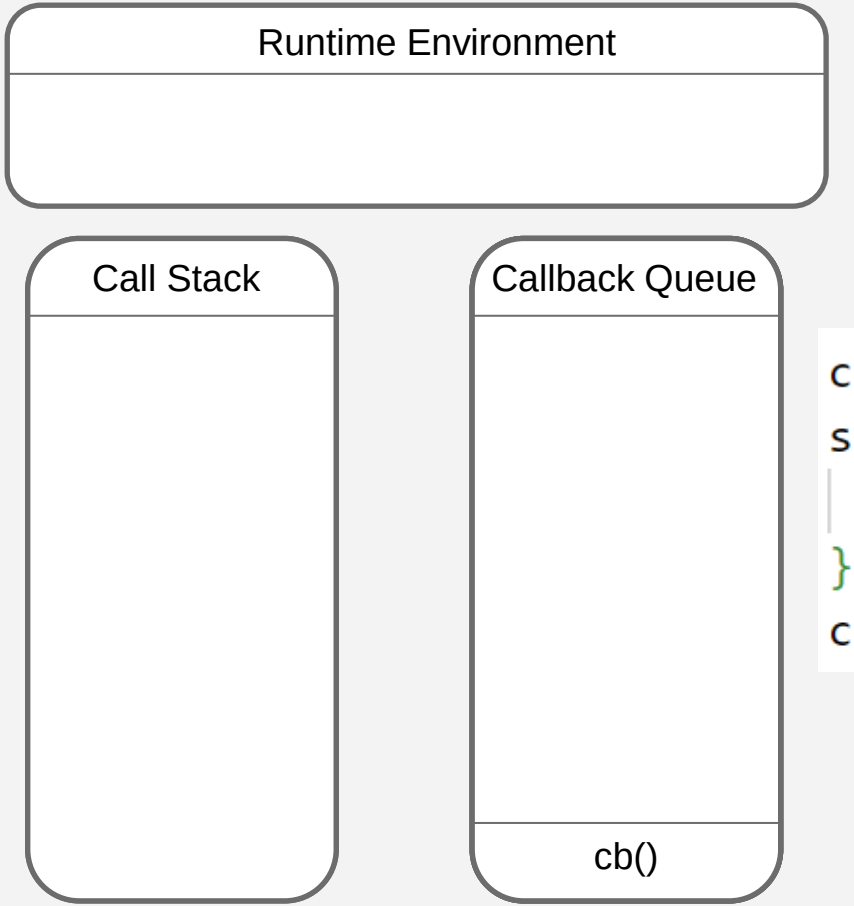

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```




```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```




```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```




```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```

Runtime Environment

Call Stack

Callback Queue

cb()

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```

Runtime Environment

Call Stack

cb()

Callback Queue

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```

Runtime Environment

Call Stack

Callback Queue

```
console.log("First");
setTimeout(() => {
    console.log("Second")
}, 10);
console.log("Third");
```

Хващане на грешки при Callback

Хващане на грешки при Callback

```
function fetchData(callback: (error: Error | null, data: any) => void) {
    const data = receiveData();

    if (data) {
        callback(null, data.message);
    }
    else {
        const error = new Error('Failed to fetch data');
        callback(error, null);
    }
}
```

```
function fetchData(callback: (error: Error | null, data: any) => void) {
    const data = receiveData();

    if (data) {
        callback(null, data.message);
    }
    else {
        const error = new Error('Failed to fetch data');
        callback(error, null);
```

```
fetchData((error, data) => {
    if (!error) {
        console.log('Success!', data);
    }
    else {
        console.log('Womp, womp...', error)
    }
})
```

```
fetchData((error, data) => {
    if (!error) {
        console.log('Success!', data);
    }
    else {
        console.log('Womp, womp...', error)
    }
})
```

```
Success! Data fetched successfully!
```

```
fetchData((error, data) => {
    if (!error) {
        console.log('Success!', data);
    }
    else {
        console.log('Womp, womp...', error)
    }
})
```

```
Womp, womp... Error: Failed to fetch data
    at fetchData (C:\Users\Kiril\Desktop\Hey, don't look here\main.js:18:23)
    at Object.<anonymous> (C:\Users\Kiril\Desktop\Hey, don't look here\main.js:22:1)
    at Module._compile (node:internal/modules/cjs/loader:1241:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1295:10)
    at Module.load (node:internal/modules/cjs/loader:1091:32)
    at Module._load (node:internal/modules/cjs/loader:938:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:83:12)
    at node:internal/main/run_main_module:23:47
```

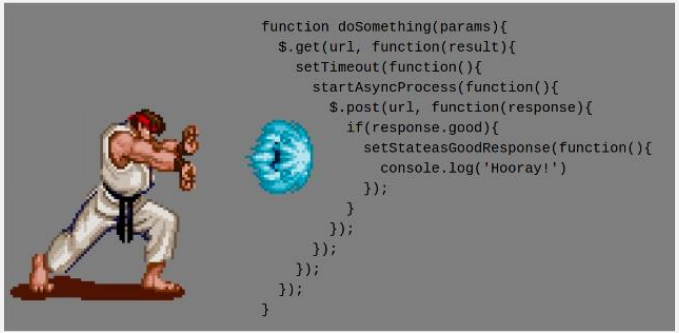
Неконсистени грешки

```
processData((err, data) => {
    if (err) {
        console.log("Big Cap Bro!");
        return;
    }
    // Do importnat stuff
})
```

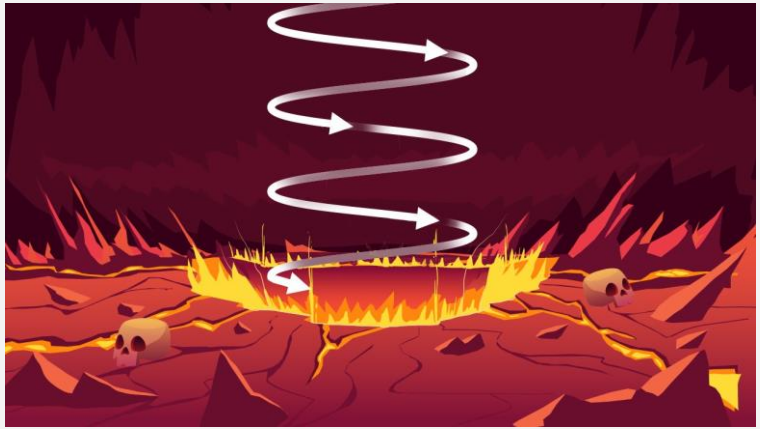
```
processData((err, data) => {
    if (err) {
        throw new Error("Big 💩")
    }
    // Do importnat stuff
})
```

03

Callback Hell



Callback Hell ⁉️ 🔥 🔥



Примерен код

```
function startCoffeeMachine(callback) {
    console.log('Starting the coffee machine...');
    setTimeout(function () {
        console.log('Coffee machine is ready.');
        callback('coffee machine is ready');
    }, 2000);
}

function grindCoffeeBeans(callback) {
    console.log('Grinding coffee beans...');
    setTimeout(function () {
        console.log('Coffee beans are ground. ');
        callback('ground coffee');
    }, 1000);
}

function boilWater(callback) {
    console.log('Boiling water...');
    setTimeout(function () {
        console.log('Water is boiled.');
        callback('boiled water');
    }, 1500);
}
```

Примерен код

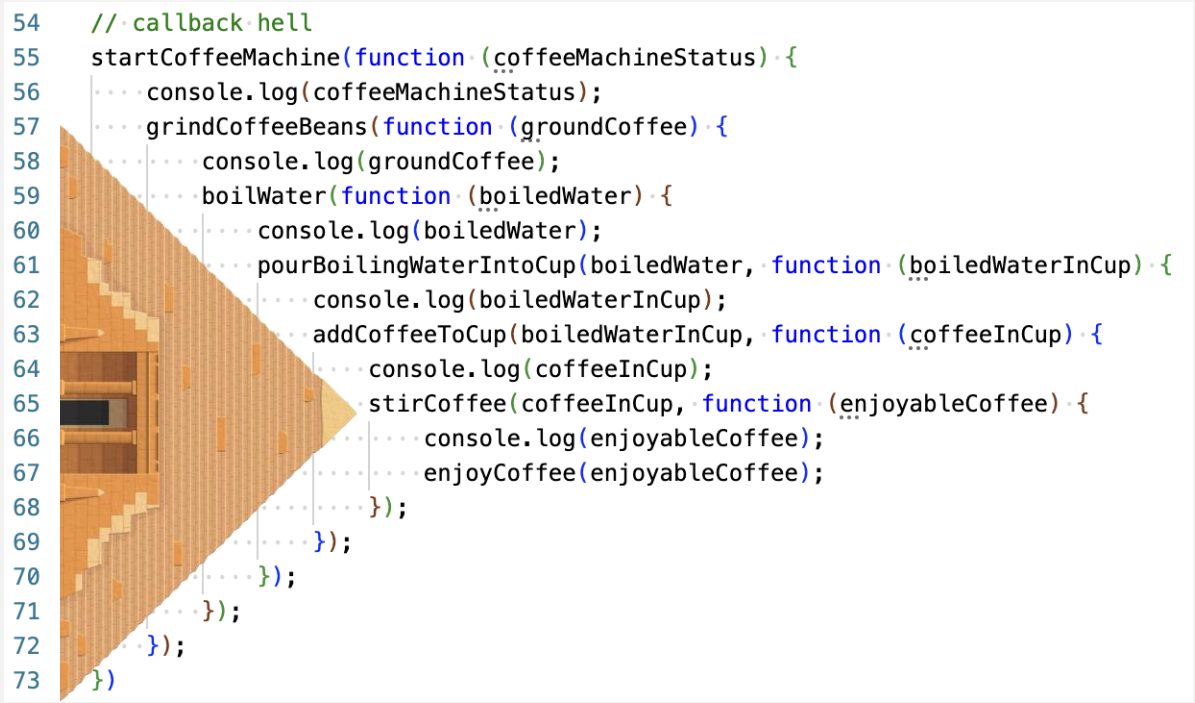
```
function pourBoilingWaterIntoCup(boiledWater, callback) {
    console.log('Pouring boiling water into a cup...');
    setTimeout(function () {
        console.log('Boiling water is in the cup. ');
        callback(boiledWater + ' in cup');
    }, 500);
}

function addCoffeeToCup(groundCoffee, callback) {
    console.log('Adding ground coffee to the cup...');
    setTimeout(function () {
        console.log('Coffee is added to the cup.');
        callback('coffee is added to ' + groundCoffee);
    }, 500);
}

function stirCoffee(coffeeInCup, callback) {
    console.log('Stirring the coffee...');
    setTimeout(function () {
        console.log('Coffee is stirred.');
        callback('enjoyable ' + coffeeInCup);
    }, 500);
}
```

```
function enjoyCoffee(coffee) {
    console.log('Enjoying the ' + coffee);
}
```

Callback Hell

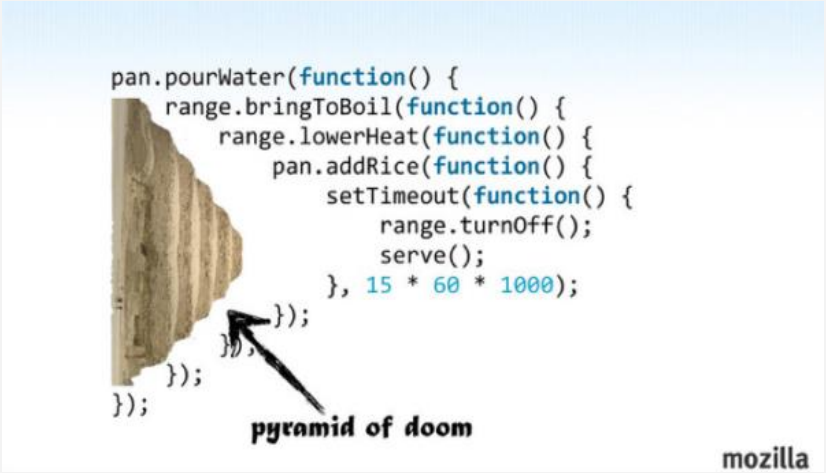
```
54  // callback hell
55  startCoffeeMachine(function (coffeeMachineStatus) {
56      console.log(coffeeMachineStatus);
57      grindCoffeeBeans(function (groundCoffee) {
58          console.log(groundCoffee);
59          boilWater(function (boiledWater) {
60              console.log(boiledWater);
61              pourBoilingWaterIntoCup(boiledWater, function (boiledWaterInCup) {
62                  console.log(boiledWaterInCup);
63                  addCoffeeToCup(boiledWaterInCup, function (coffeeInCup) {
64                      console.log(coffeeInCup);
65                      stirCoffee(coffeeInCup, function (enjoyableCoffee) {
66                          console.log(enjoyableCoffee);
67                          enjoyCoffee(enjoyableCoffee);
68                      });
69                  });
70              });
71          });
72      });
73  })
```

Callback Hell (Pyramid of Doom)



```
// callback hell
startCoffeeMachine(function (coffeeMachineStatus) {
    console.log(coffeeMachineStatus);
    grindCoffeeBeans(function (groundCoffee) {
        console.log(groundCoffee);
        boilWater(function (boiledWater) {
            console.log(boiledWater);
            pourBoilingWaterIntoCup(boiledWater, function (boiledWaterInCup) {
                console.log(boiledWaterInCup);
                addCoffeeToCup(boiledWaterInCup, function (coffeeInCup) {
                    console.log(coffeeInCup);
                    stirCoffee(coffeeInCup, function (enjoyableCoffee) {
                        console.log(enjoyableCoffee);
                        enjoyCoffee(enjoyableCoffee);
                    });
                });
            });
        });
    });
})
```

Основни проблеми при callback hell

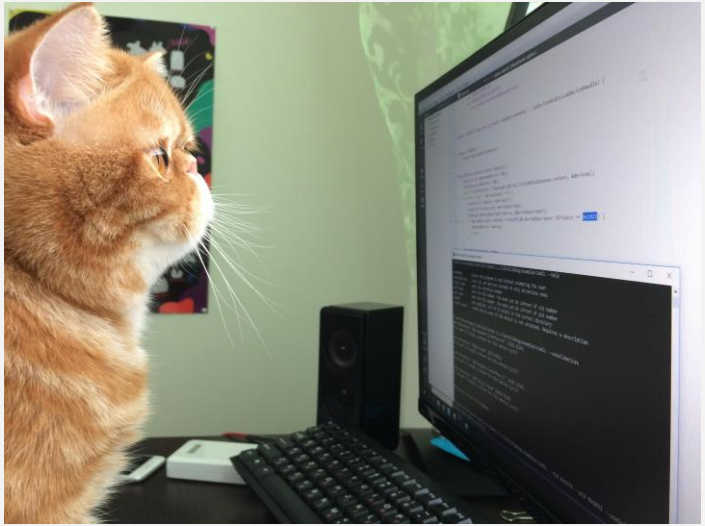


Вложени повиквания

Основни проблеми при callback hell

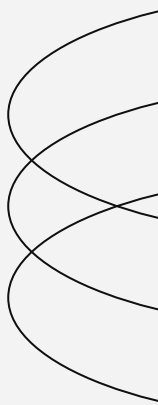
Вложени повиквания
Четимост и Поддръжка

Основни проблеми при callback hell



Вложени повиквания
Четимост и Поддръжка
Дебъгване и Рефакториране

Начини да избегнем Callback Hell



Да използваме добре именувани функции

Разбиване на по-големи функции към по-малки

Използвайки обещания (Promises)

Начини да избегнем Callback Hell

Разбиване на по-големи функции към по-малки

```
function randomFunction3(callback) {
    setTimeout(() => {
        console.log('Function 3');
        callback();
    }, 1000);
}


// Example without named functions
randomFunction1(() => {
    randomFunction2(() => {
        randomFunction3(() => {
            // More nested callbacks...
        });
    });
});
```

```
// Example with named functions
function handleResult1() {
    randomFunction2(handleResult2);
}

function handleResult2() {
    randomFunction3(handleResult3);
}

function handleResult3() {

}

randomFunction1(handleResult1);
```

04

Обещания

Какво представлява едно обещание?

Какво представлява едно обещание?

```
const myPromise = new Promise<string>((resolve, reject) => {
    const success = false;
    if (success) {
        resolve("Yay we made it")
    } else {
        reject(new Error("Booo! Didn't make it!"))
    }
})
```

Какво представлява едно обещание?

```
const myPromise = new Promise<string>((resolve, reject) => {
    const success = false;
    if (success) {
        resolve("Yay we made it")
    } else {
        reject(new Error("Booo! Didn't make it!"))
    }
})
```

```
myPromise
    .then(data => {
        console.log(data);
    })
    .catch((error : Error) => {
        console.log("Boss... We have a problem:");
        console.log(error.message);
    });
```

Какво представлява едно обещание?

```
const myPromise = new Promise<string>((resolve, reject) => {
    const success = true;
    if (success) {
        resolve("Yay we made it!")
    } else {
        reject(new Error("Booo! Didn't make it!"))
    }
})
```

```
myPromise
    .then(data => {
        return data + " The best day";
    })
    .then(data => {
        console.log(data, "of my life!")
    })
```

Какво ще изкара на конзолата следният код?

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise
.then(() => console.log(3))
.then(() => console.log(4));
```

Какво ще изкара на конзолата следният код?

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise
.then(() => console.log(3))
.then(() => console.log(4));
```

```
1
3
2
4
```

Runtime Environment

Call Stack

Callback Queue

Microtask Queue

Call Stack

Callback Queue

Microtask Queue

Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

Call Stack

main()

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

Call Stack

new Promise()

main()

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

Call Stack

myPromise.then

main()

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

log(1)

Call Stack

main()

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

log(1)

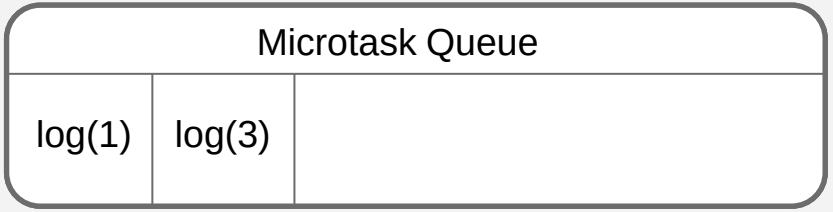
Call Stack

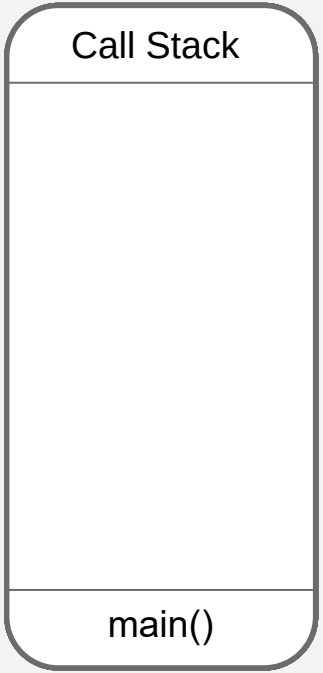
myPromise1.then

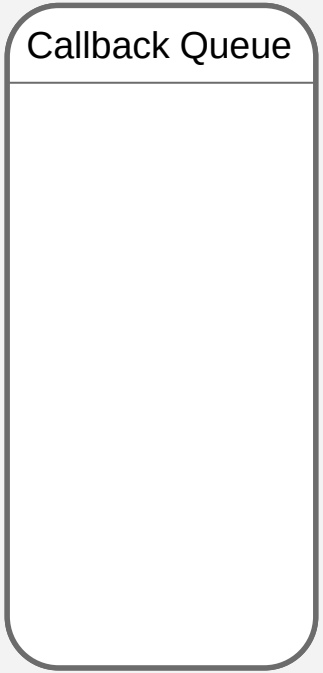
main()

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```







```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

log(1)	log(3)	

Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

log(1)	log(3)	

Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

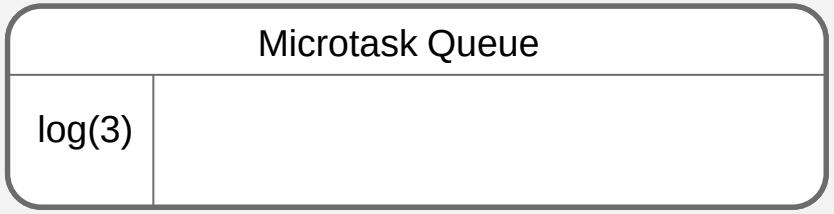
log(3)

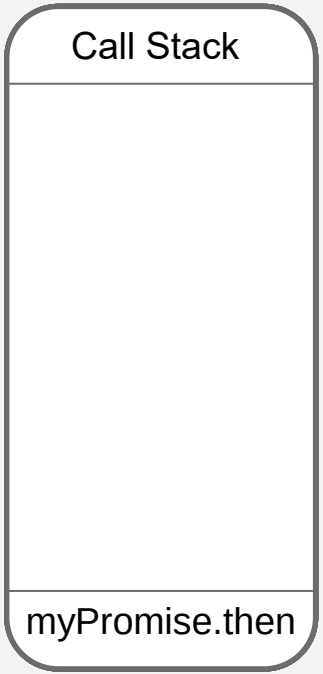
Call Stack

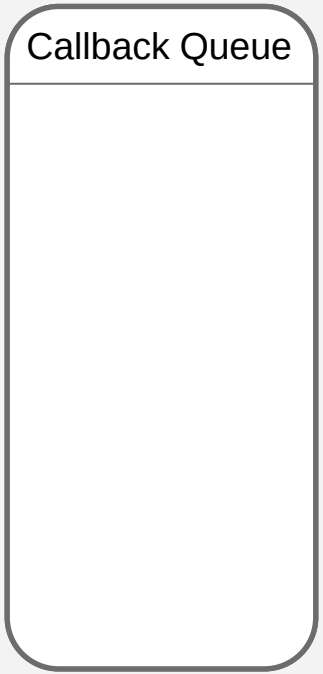
log(1)

Callback Queue

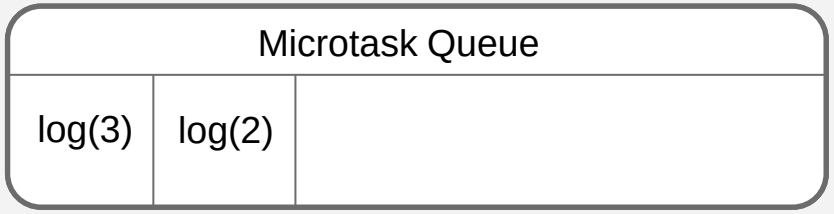
```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

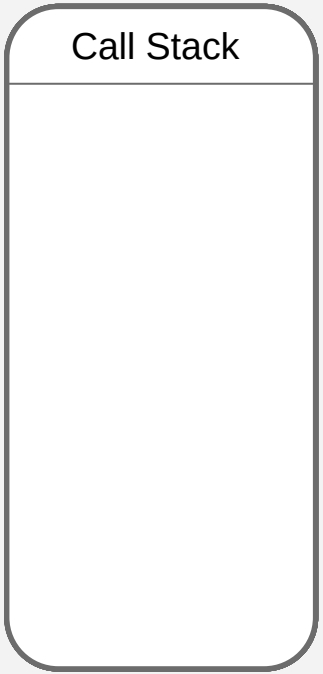


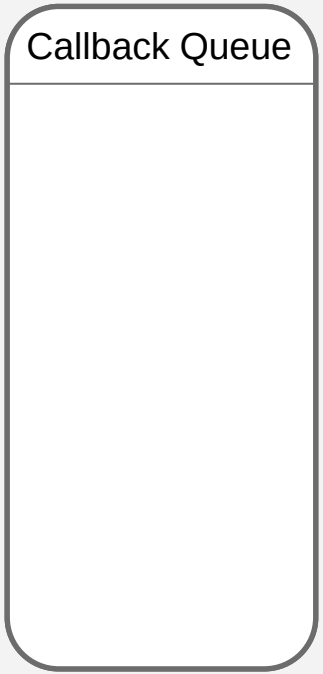




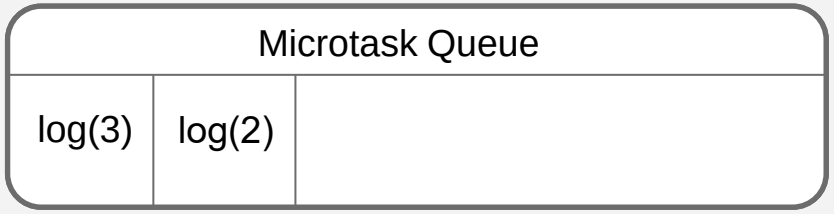
```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

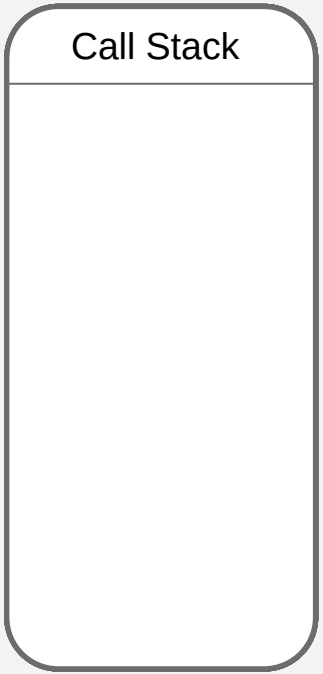


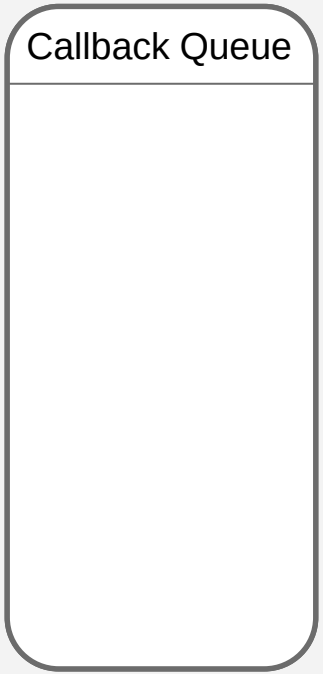




```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```







```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

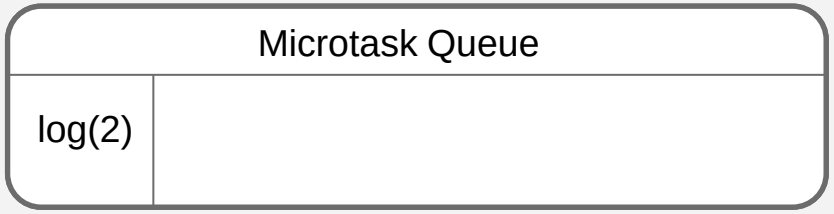
log(2)

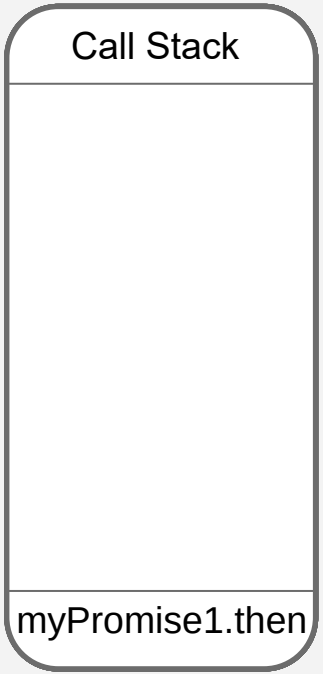
Call Stack

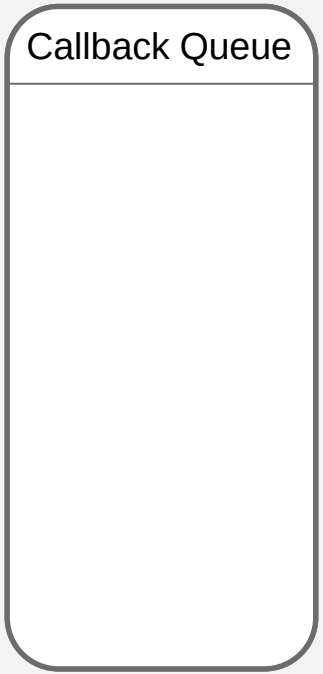
log(3)

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```







```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

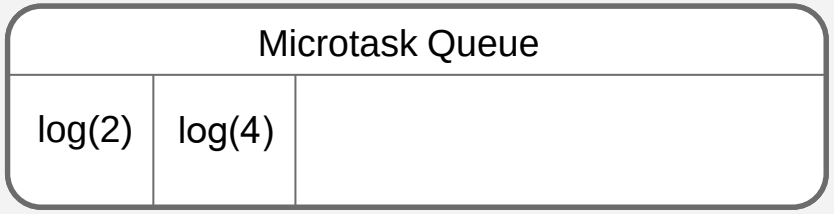
Microtask Queue

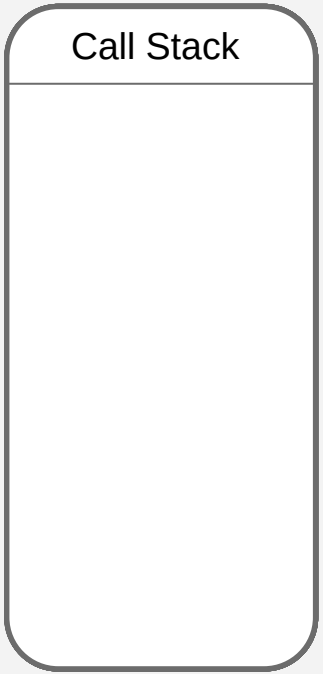
log(2)	log(4)	

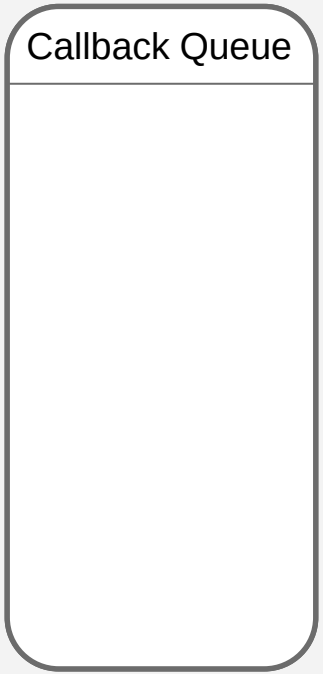
Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```







```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

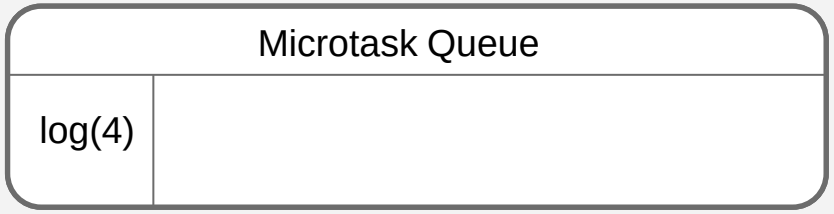
log(4)

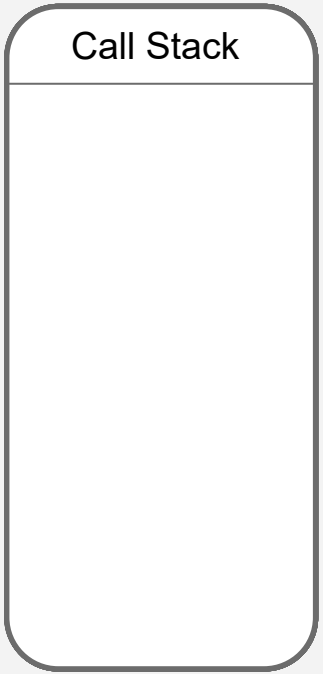
Call Stack

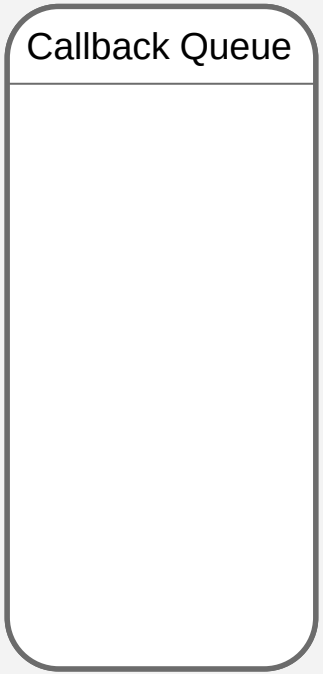
log(2)

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```







```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

log(4)

Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

Call Stack

log(4)

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

Microtask Queue

Call Stack

Callback Queue

```
myPromise
.then(() => console.log(1))
.then(() => console.log(2));

myPromise1
.then(() => console.log(3))
.then(() => console.log(4));
```

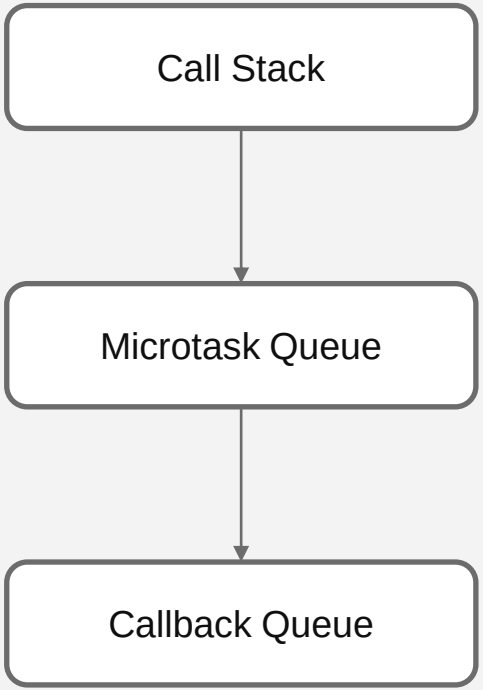
А този?

```
setTimeout(() => console.log(1), 2000);

myPromise
.then(() => console.log(2))
.then(() => console.log(3));

setTimeout(() => console.log(4), 500);
```

А този?

```
setTimeout(() => console.log(1), 2000);

myPromise
.then(() => console.log(2))
.then(() => console.log(3));

setTimeout(() => console.log(4), 500);
```

```
2
3
4
1
```



Опашката на микрозадачите се изпълнява винаги преди тази на обратните повиквания!

* Реално съществуват няколко вида Callback и Microtask опашки

Хващане на грешки при Promises

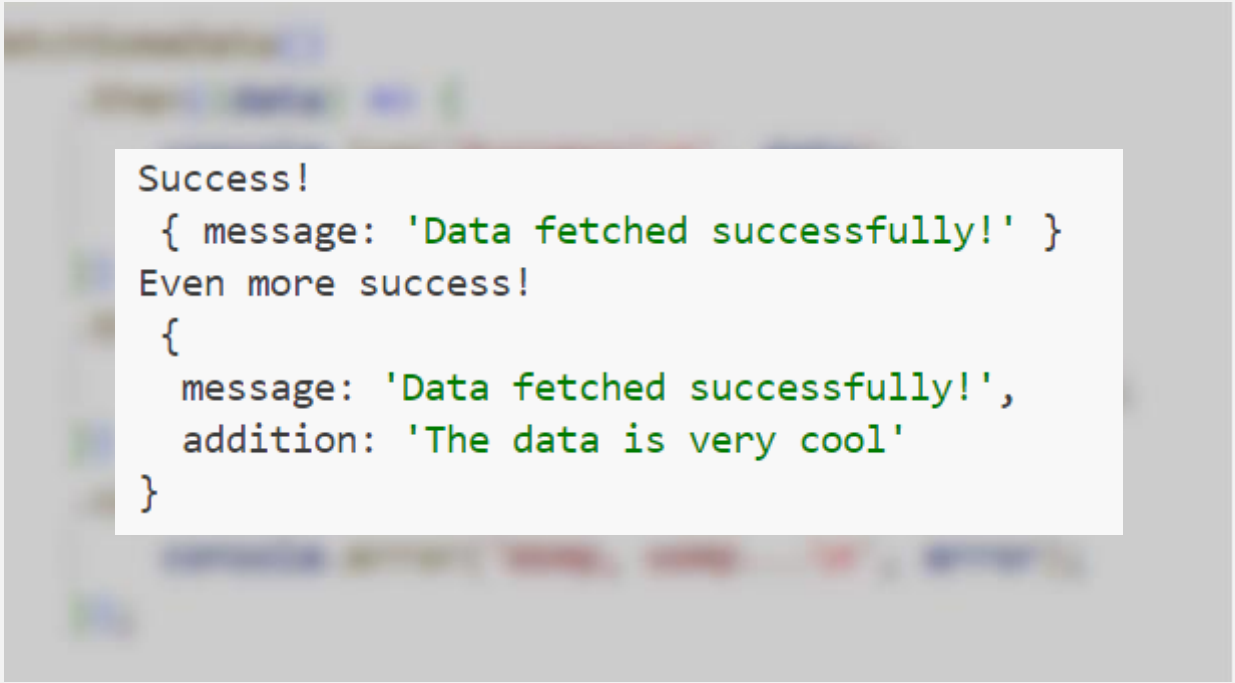
Хващане на грешки при Promises

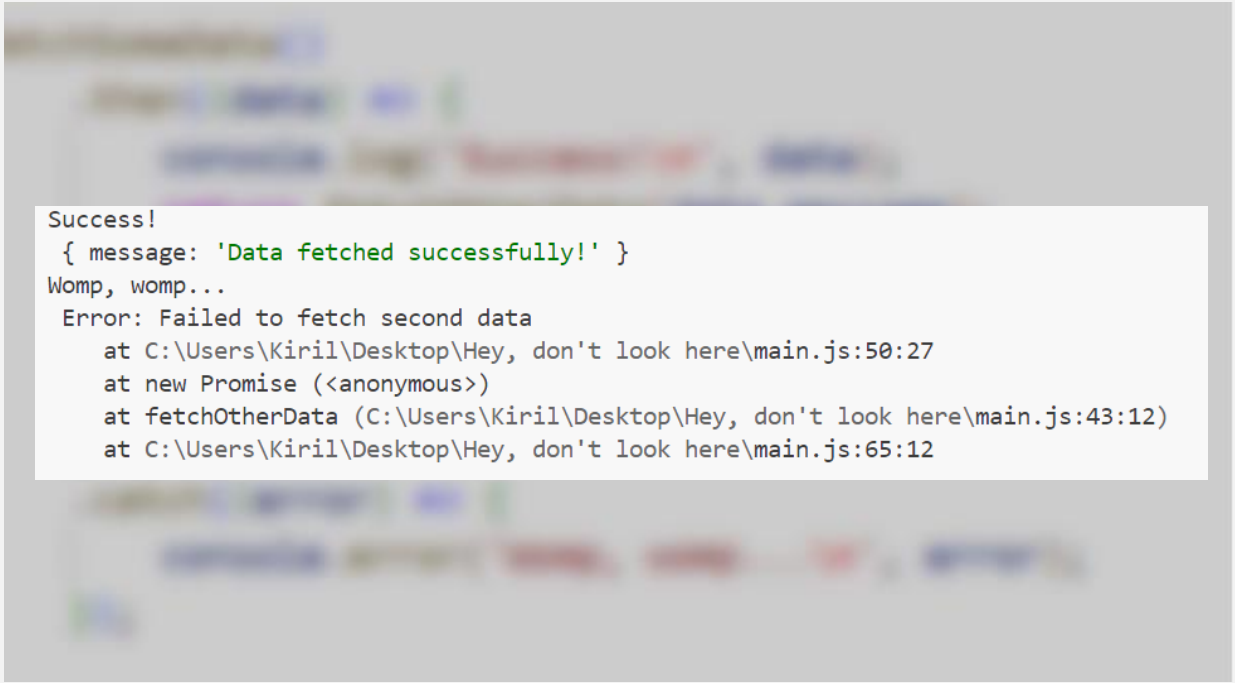
```
function fetchSomeData(): Promise<any> {
    return new Promise((resolve, reject) => {
        const data = receiveData();
        if (data) {
            resolve(data);
        } else {
            const error = new Error('Failed to fetch first data');
            reject(error);
        }
    });
}
```

```
function fetchSomeData(): Promise<any> {
    return new Promise((resolve, reject) => {
        const data = receiveData();
        if (data) {
            resolve(data);
        } else {
            const error = new Error('Failed to fetch first data');
            reject(error);
        }
```

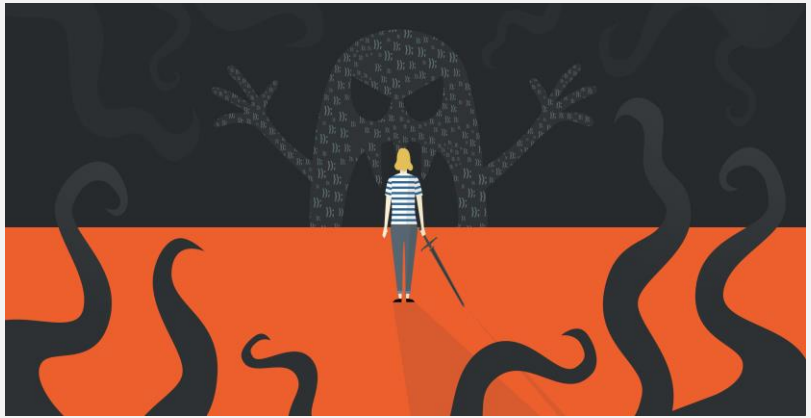
```
fetchSomeData()
    .then((data) => {
        console.log('Success!', data);
    })
    .catch((error) => {
        console.error('Womp, womp...', error)
    })
```

```
fetchSomeData()
    .then((data) => {
        console.log('Success!\n', data);
        return fetchOtherData(data.message);
    })
    .then((data) => {
        console.log('Even more success!\n', data);
    })
    .catch((error) => {
        console.error('Womp, womp...\n', error);
    });
```


Success!
{ message: 'Data fetched successfully!' }
Even more success!
{
message: 'Data fetched successfully!',
addition: 'The data is very cool'
}


Success!
{ message: 'Data fetched successfully!' }
Womp, womp...
Error: Failed to fetch second data
at C:\Users\Kiril\Desktop\Hey, don't look here\main.js:50:27
at new Promise (<anonymous>)
at fetchOtherData (C:\Users\Kiril\Desktop\Hey, don't look here\main.js:43:12)
at C:\Users\Kiril\Desktop\Hey, don't look here\main.js:65:12

Promise Hell

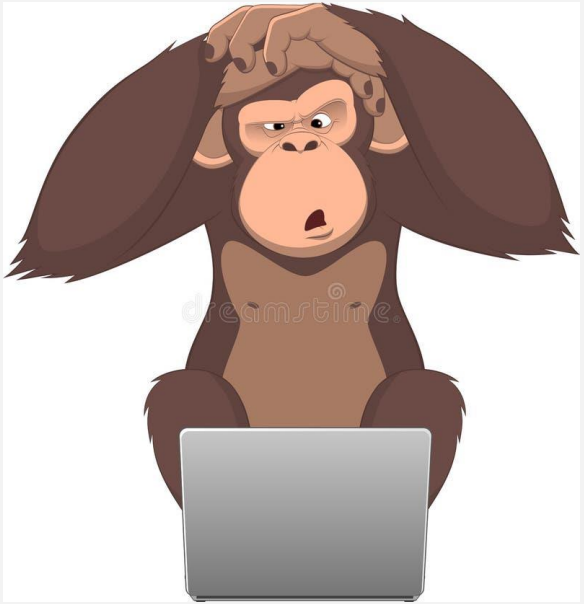


Нали обещанията бяха решението на Callback Hell!?

Как изглежда Promise Hell?

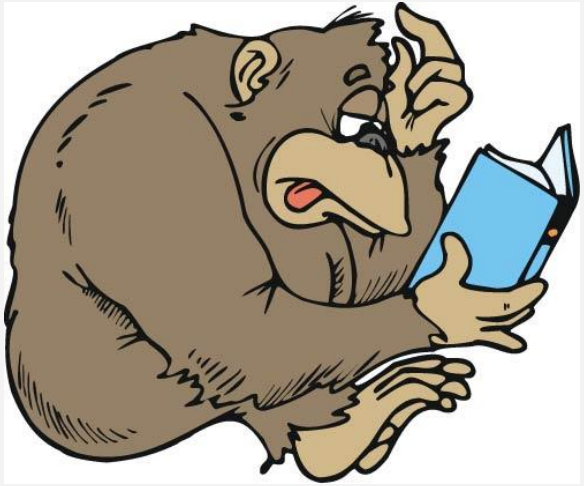
```
function func() {
    getSmth(2).then((res1: number) => {
        getSmth(3).then((res2: number) => {
            console.log(res1 + res2);
        });
    });
}
```

Проблемите с Promise Hell



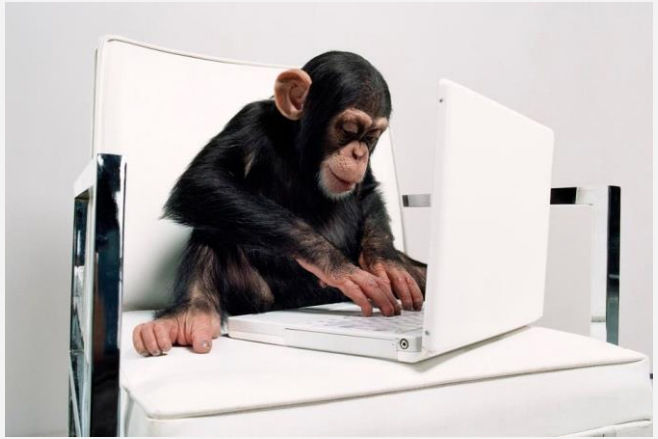
Сложност

Проблемите с Promise Hell



Четимост

Проблемите с Promise Hell



Debugging

Проблемите с Promise Hell

Поддръжка

Проблемите с Promise Hell

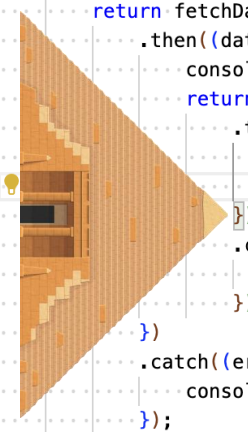
```
function func() {
    getSmth(2).then((res1: number) => {
        getSmth(3).then((res2: number) => {
            console.log(res1 + res2);
        });
    });
}
```

```
function func() {
    getSmth(2).then((res1: number) => {
        getSmth(3).then((res2: number) => {
            getSmth(4).then((res3: number) => {
                console.log(res1 + res2 + res3);
            });
        });
    });
}
```

Нещата могат да станат доста грозни

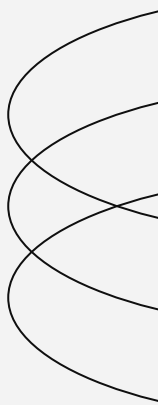
```
fetchData1()
    .then((data1) => {
        console.log("Data 1:", data1);
        return fetchData2()
            .then((data2) => {
                console.log("Data 2:", data2);
                return fetchData3()
                    .then((data3) => {
                        console.log("Data 3:", data3);
                        // More nested .then() callbacks...
                    })
                    .catch((error) => {
                        console.error("Error fetching data from API 3:", error);
                    });
            })
            .catch((error) => {
                console.error("Error fetching data from API 2:", error);
            });
    })
    .catch((error) => {
        console.error("Error fetching data from API 1:", error);
    });
```

Пирамидата на гибелта...



```
fetchData1()
    .then((data1) => {
        console.log("Data 1:", data1);
        return fetchData2()
            .then((data2) => {
                console.log("Data 2:", data2);
                return fetchData3()
                    .then((data3) => {
                        console.log("Data 3:", data3);
                        // More nested .then() callbacks...
                    })
                    .catch((error) => {
                        console.error("Error fetching data from API 3:", error);
                    });
            })
            .catch((error) => {
                console.error("Error fetching data from API 2:", error);
            });
    })
    .catch((error) => {
        console.error("Error fetching data from API 1:", error);
    });
```

Как да избегнем Promise Hell?



Разбиване на по-сложните операции

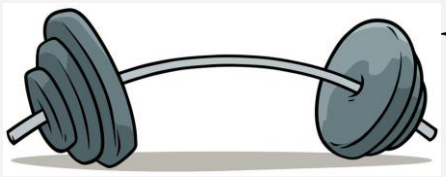
Избягване на дълбоко вложените извиквания

Async/Await

05

Async & Await

&



Async & Await


Learning
async/await
Me


CALLBACKS
PROMISES
ASYNC/AWAIT
imgflip.com

Async & Await

```
function getData(): Promise<string>
async function getData() {
    return 'Yuppie, data :D';
```

Защо използваме Async/ Await?

```
// Using Promises
function fetchData1(): Promise<string> {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve("Data fetched successfully");
        }, 1000);
    });
}

function processData(): Promise<void> {
    return fetchData1().then((data) => {
        console.log(data);
    });
}
```


```
// Using async/await
async function processDataAsync(): Promise<void> {
    const data = await fetchData();
    console.log(data);
}
```

Защо Async/Await съществува всъщност?

Синтаксис наподобяващ синхронния код

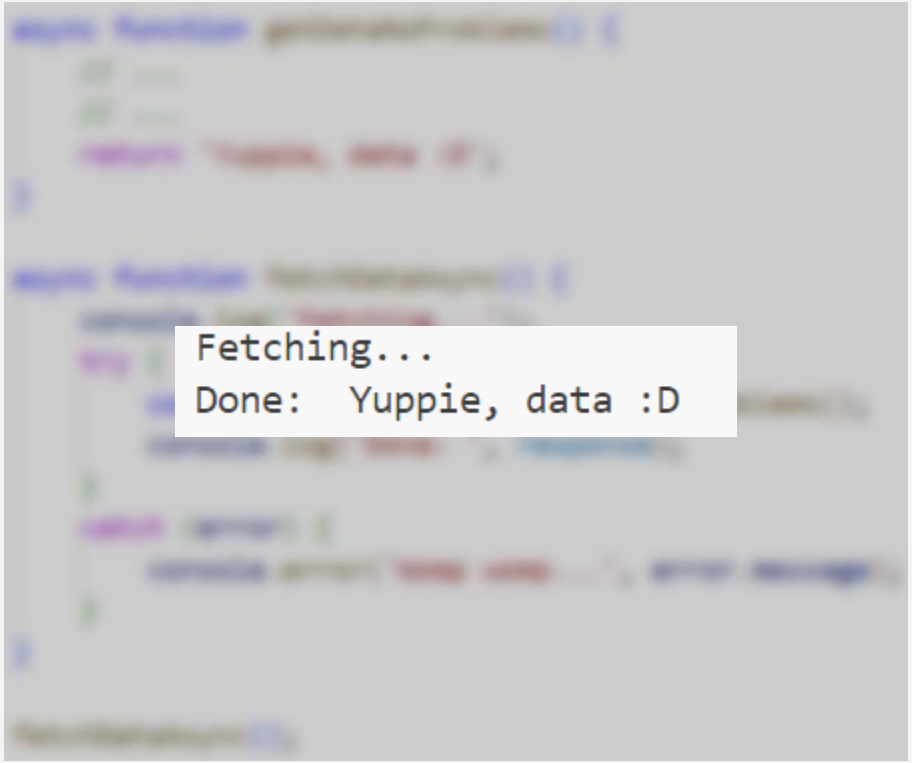
```
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Data fetched successfully');
        }, 1000);
    });
}

fetchData()
    .then((data) => {
        console.log(data);
        return processData(data);
    })
    .then((processedData) => {
        console.log(processedData);
    })
```

```
async function fetchData() {
    // ...
    return 'Data fetched successfully';
}

async function fetchDataAndProcess() {
    const data = await fetchData();
    console.log(data);
    const processedData = await processData(data);
    console.log(processedData);
}
```

Хващане на грешки при async/await

```
async function getDataNoProblems() {
    // ...
    // ...
    return 'Yuppie, data :D';
}

async function fetchDataAsync() {
    console.log('Fetching...');
    try {
        const response = await getDataNoProblems();
        console.log('Done: ', response);
    }
    catch (error) {
        console.error('Womp womp...', error.message);
    }
}

fetchDataAsync();
```

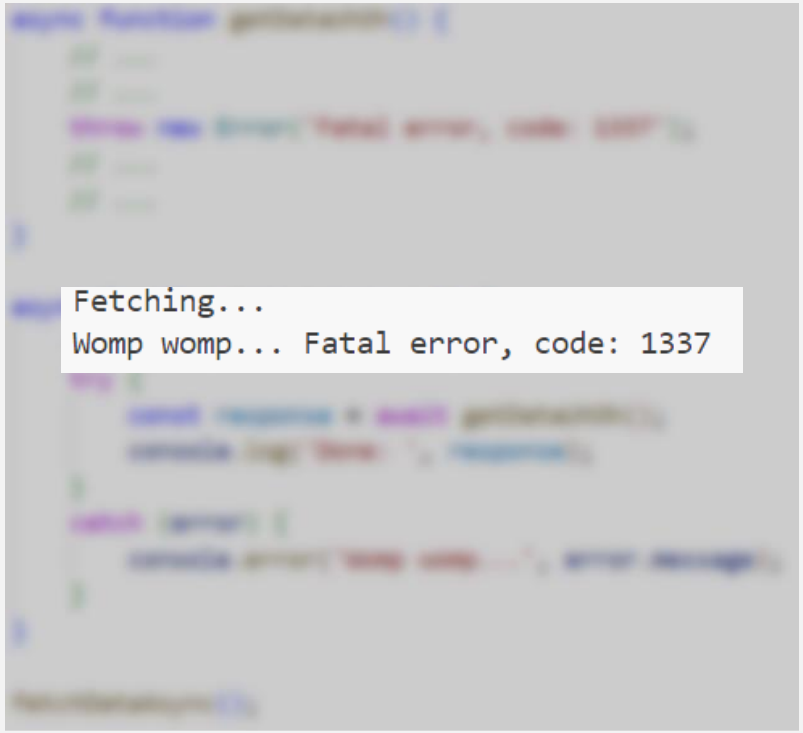
Хващане на грешки при async/await



Хващане на грешки при async/await

```
async function getDataUhOh() {
    // ...
    // ...
    throw new Error('Fatal error, code: 1337');
    // ...
    // ...
}

async function fetchDataAsync() {
    console.log('Fetching...');
    try {
        const response = await getDataUhOh();
        console.log('Done: ', response);
    }
    catch (error) {
        console.error('Womp womp...', error.message);
    }
}

fetchDataAsync();
```

Хващане на грешки при async/await



```
Fetching...
Womp womp... Fatal error, code: 1337
```

06

Еволюция на асинхронното програмиране в TypeScript (JavaScript)

Еволюция на асинхронните подходи

```
getData(a => {
    getMoreData(a, b => {
        getMoreData(b, c => {
            getMoreData(c, d => {
                getMoreData(d, e => {
                    console.log(e);
                });
            });
        });
    });
});
```

Еволюция на асинхронните подходи

```
getData()
.then(a => getMoreData(a))
.then(b => getMoreData(b))
.then(c => getMoreData(c))
.then(d => getMoreData(d))
.then(e => console.log(e));
```

Еволюция на асинхронните подходи

```
(async () => {
    const a = await getData();
    const b = await getMoreData(a);
    const c = await getMoreData(b);
    const d = await getMoreData(c);
    const e = await getMoreData(d);
    console.log(e);
})();
```

Благодарим за вниманието!

Въпроси?

+ + +

x x x