Day 4.2
Lecture Name: Async-Await

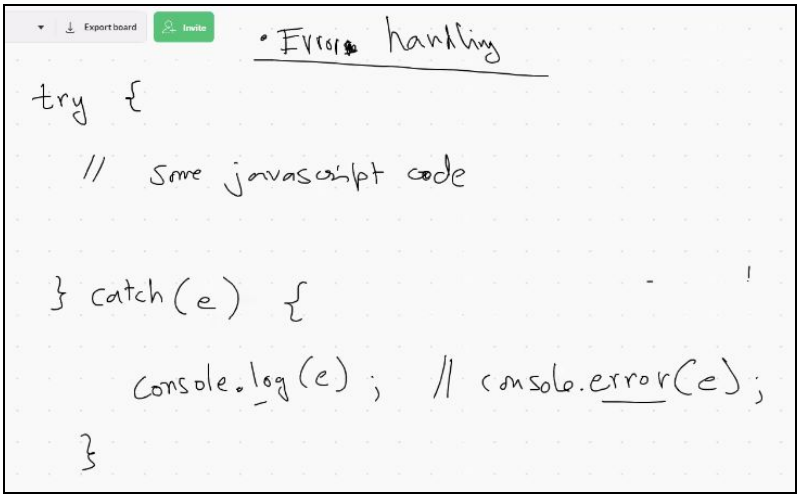
Topics Discussed:

Error Handling | Multithreading

Two ways of handling an error. Discussing the first way

Try-catch block

- The code that you know will cause an error or can cause an error is written inside the try block.
- If at any point error occurred on any statement in try block the program execution in try block stops and execution control jumps to catch block.



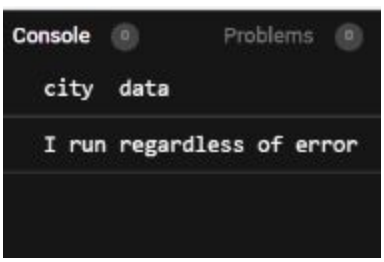
- And inside a catch block we either print error or handle it in some way.

Finally block

- This block will always run either there is an error or there is no error.
- If there is no error in try block, catch block will not execute directly finally block will execute. If the error is present program execution stops in try block and catch block executes and then finally block executes.
- This is an optional block, It is not necessary to write this block. This block is generally used for cleaning up the code from try block.
- This is a basic structure of error handling

Example

```
try {
  const city = document.getElementById("city-search").value;
  console.log("city", city);
} catch(e) {
  console.log("Got an error");
  console.error(e);
} finally {
  console.log("I run regardless of error");
}
```

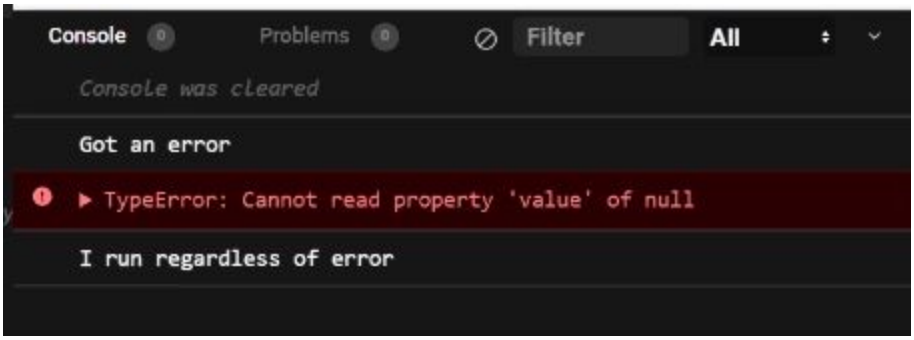


- As there was no error in the try block it executed and catch block is skipped, then finally block is executed.

Simulating an error

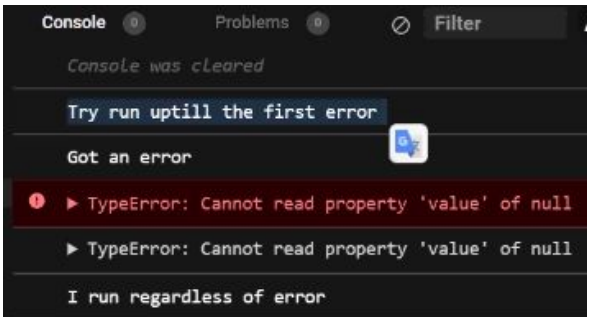
- There is no element that has an id=”city-search-2”. By doing “city-search-2” error will occur in try block.

```
try {
  const city = document.getElementById("city-search-2").value;
  console.log("city", city);
} catch (e) {
  console.log("Got an error");
  console.error(e);
} finally {
  console.log("I run regardless of error");
}
```



- In the below code 3rd line will cause an error and city will not be printed.
- console.error(e) will print the error in highlighted red format, but console.log(e) prints it normally.

```
try {
  console.log("Try run uptill the first error");
  const city = document.getElementById("city-search-2").value;
  console.log("city", city);
} catch (e) {
  console.log("Got an error");
  console.error(e);
  console.log(e);
} finally {
  console.log("I run regardless of error");
}
```



- No concept of multiple catch block present in javascript. In java there are multiple catch blocks for catching error.
- Error handling and exception handling does not have much difference. Syntactically they are the same in java and javascript.

Another Example:

- ‘n’ is not defined still we are printing this will cause an error.

```
try {
  console.log("Try run uptill the first error");
  const city = document.getElementById("city-search").value;
  console.log(n);
  console.log("city", city);
} catch (e) {
  console.log("Got an error");
  console.error(e);
  console.log(e);
} finally {
  console.log("I run regardless of error");
}
```

```
Console  Problems
Console was cleared
Try run uptill the first error
Got an error
▶ ReferenceError: n is not defined
▶ ReferenceError: n is not defined
I run regardless of error
```

- You can observe the city is not printed this shows that once try block detects one error it will not execute remaining statements and jumped to catch block

Reassigning const value

```
try {
  const a = 23;
  console.log("Try run uptill the first error");
  const city = document.getElementById("city-search").value;
  // console.log(n);
  console.log("city", city);
  a = 45;
} catch (e) {
  console.log("Got an error");
  console.error(e);
  console.log(e);
} finally {
  console.log("I run regardless of error");
}
```

```
Console  Problems  Filter  All
Console was cleared
Try run uptill the first error
city  test 2
Got an error
▶ TypeError: Assignment to constant variable.
▶ TypeError: Assignment to constant variable.
I run regardless of error
```

- Finally can be used for some cleanup you want to do.
- Finally block is recommended for cleanup.

Creating a function with error

- ‘b’ is not defined.

```
function fn(a) {
  console.log(b);
}
```

- Called the function in try block.

```
Console  Problems  Filter
Got an error
▶ ReferenceError: b is not defined
▶ ReferenceError: b is not defined
I run regardless of error
follow up code
```

- When you call a function in a try block and that function has an error at a specific statement. The remaining statements in the function will not execute and error will be thrown outside. This error is catched in the catch block.

Defining your own error

```
<script>
  function fn(a) {
    new Error("NewtonSchool is down");
  }
```

- ‘new’ is a constructor which will create a new type of object here its Error object.
- Error is defined not thrown it yet, In order to throw the error see the below code. It shows when at any point value of a is 42 the custom error we created will be thrown.

```
<script>
  function fn(a) {
    const customError = new Error("NewtonSchool is down");
    if(a === 42) {
      throw customError;
      // throw new Error("Some custome Error");
    }
  }
```

Way of throwing an error

- Nothing will run from the point where the error is thrown.

fn(42) is called

```
<script>
  function fn(a) {
    const customError = new Error("NewtonSchool is down");
    if (a === 42) {
      throw customError;
      // throw new Error("Some custome Error");
    }
    console.log("After error");
  }
```

“After error” is not printed because an error is thrown before that

Another Example:

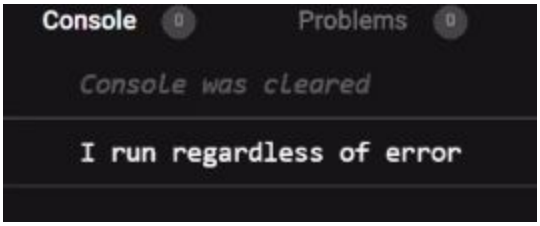
- This try catch block is inside the above function fn(a){.....}.

```
try {
    // const a = 23;
    // console.log("Try run uptill the first er
    // const city = document.getElementById("ci
    // // console.log(n);
    // console.log("city", city);
    // a = 45;

    // show a loader
    // do some calculation
  const b = 45;
  return b;
} catch (e) {
  console.log("Got an error");
  console.error(e);
  console.log(e);
} finally {
  console.log("I run regardless of error");
    // // hide the loader
}
```

Will finally execute?

- As there is a return statement in try block. We think that finally block will not execute. But this is wrong finally block runs even if you write return statement in try block.
- This is how it works



- First, the value of b=45 is saved somewhere temporarily and finally block runs. Once finally block is completed the value 45 saved in the temporary variable is returned by the function.

```
const b = 45;
try {
    // const a = 23;
    // console.log("Try run uptill the first
    // const city = document.getElementById("
    // // console.log(n);
    // console.log("city", city);
    // a = 45;

    // show a loader
    // do some calculation
  b = b + 2;
  return b;
} catch (e) {
  console.log("Got an error");
  console.error(e);
  console.log(e);
} finally {
  b = b + 4;
  console.log("I run regardless of error");
    // // hide the loader
}
console.log("follow up code");
```

```
Got an error
▶ TypeError: Assignment to constant variable.
▶ TypeError: Assignment to constant variable.
```

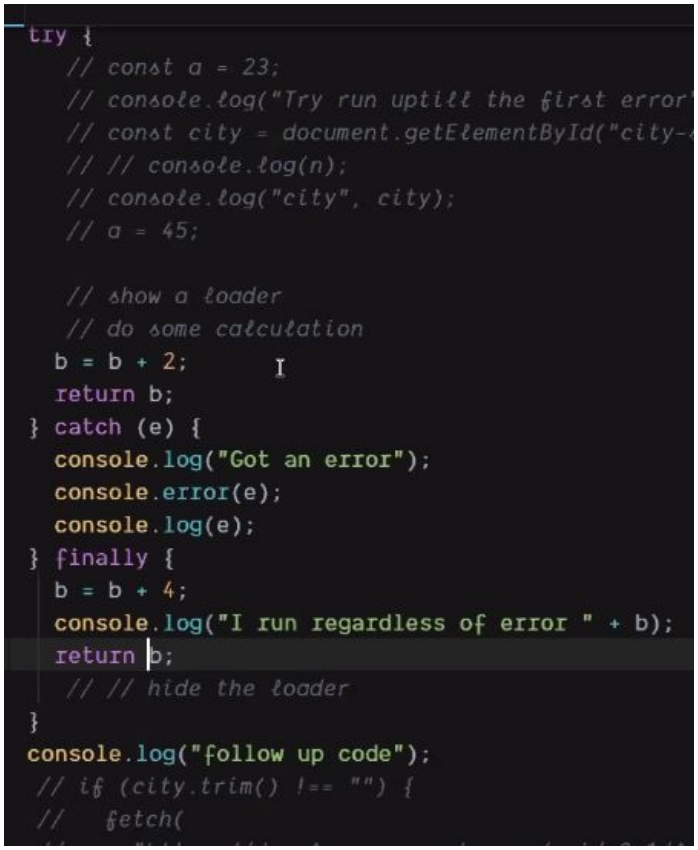
- Error is present in try block (b=b+2). So it never reached the (return b) statement. It went inside catch block.
- finally block is not running, no its not like that its is runnnig but when the error is detected it stop executing the remaining statements of finally block. If you write (b=b+4) inside try catch block the remaining statement would have executed and we could have seen “I run regardless of error”.
- In the below code ‘b’ is defined using let and defined before the try block. So first try block returned the value of b. As there is no error in try block catch block is not executed. And then finally block is executed.

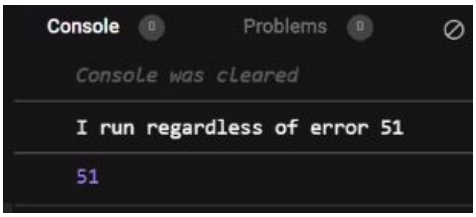
```
64          b = b + 2;
65          return b;
66        } catch (e) {
67          console.log("Got an error");
68          console.error(e);
69          console.log(e);
70        } finally {
71          b = b + 4;                                  let b: number
72          console.log("I run regardless of error " + b);
73           // // hide the loader
74        }
75        console.log("follow up code");
```

```
Console 0   Problems 0   Filter
Console was cleared
I run regardless of error 51
47
```

- Inside finally block, the value of b printed is 51 that means the value of b=47 is still held by the b and printed as b=57. And after that the value which was returned by the b in try block is actually returned. Before the value was just holded up.

New return value override





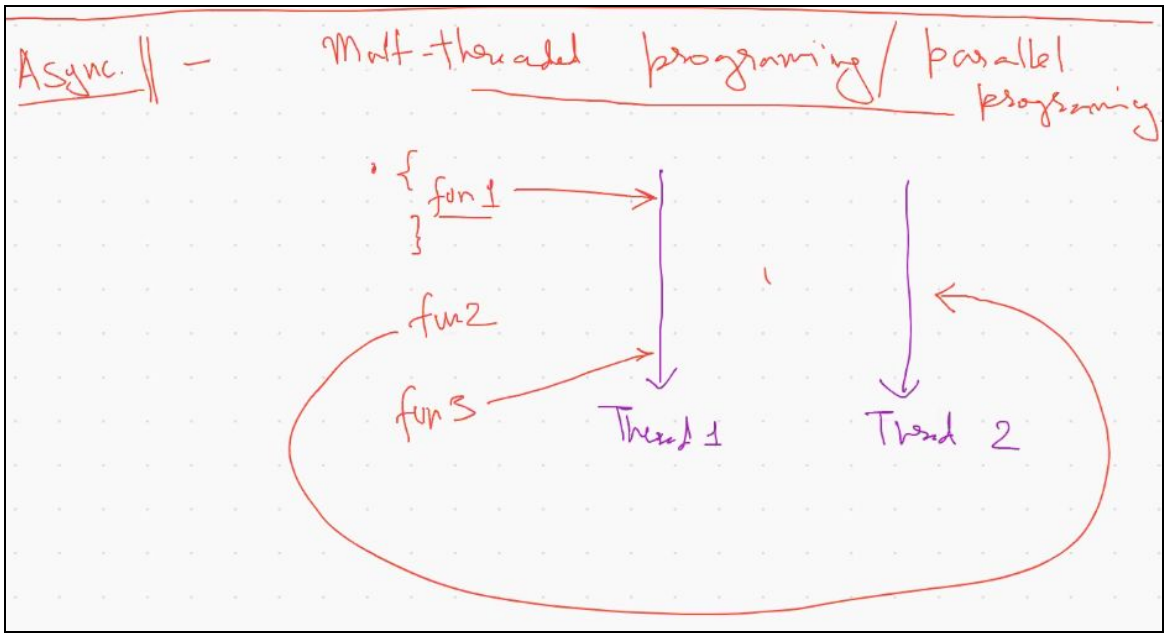
- Overwriting the previous return value.
    - If you return b in finally block, as finally block runs after try block the return value of b will be overridden by new value of b in finally and that value is written.
    - This concept is asked by most interviewers who wants to trick the candidate but this thing is not recommended it will cause confusion, their are no application of the return value overriding.
- Finally is the only case where it will execute even after return keyword.

Tips:

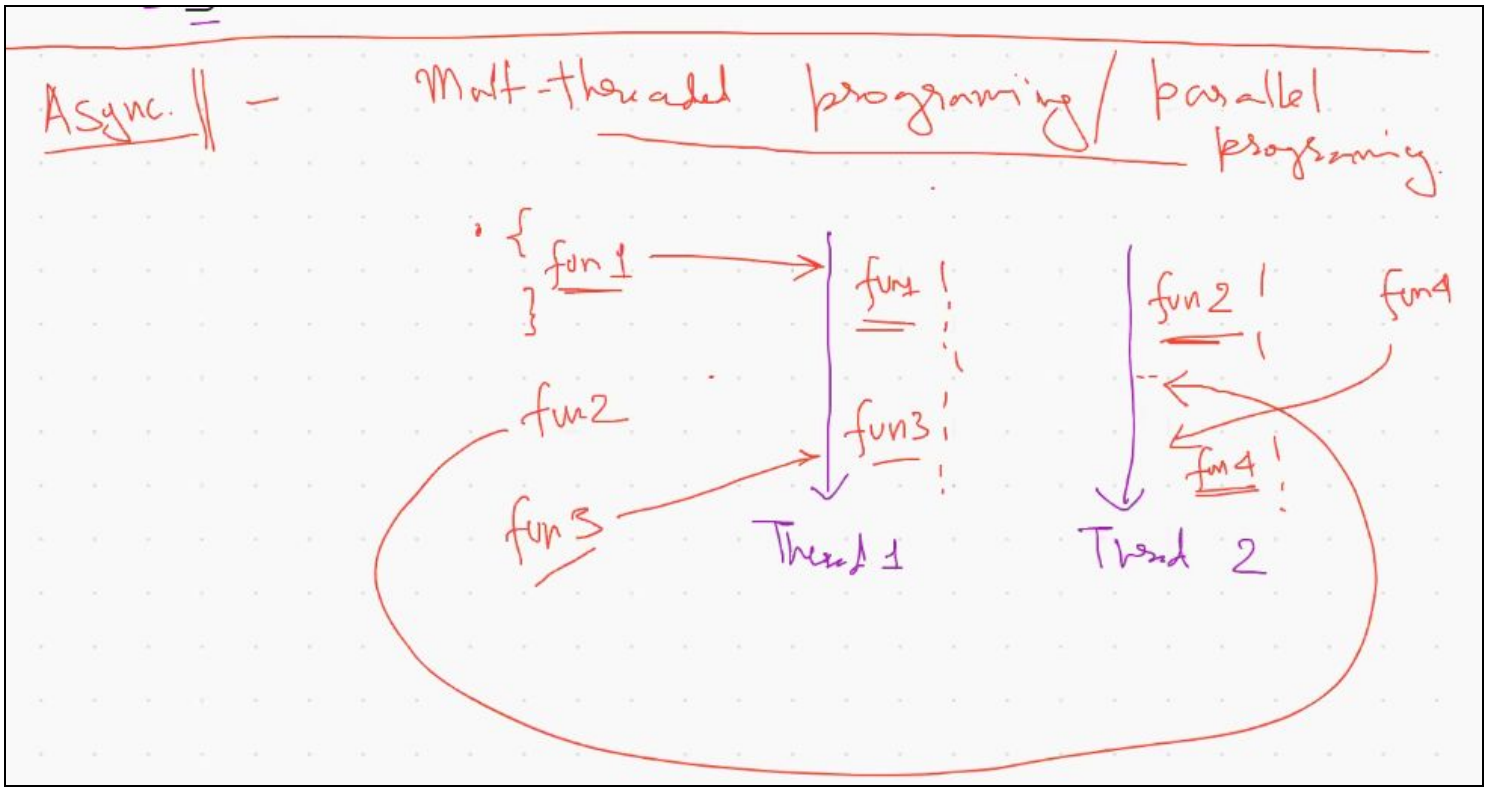
- Code cleanliness is important because employees keeps changing in a company so interviewer looks for code cleanliness skills. Interviewer will observe it

Multi threading

- Executing code in multiple threads.
- f(n) in thread 1
- f(n2) in thread 2



- We execute code one after the other in sequence In thread one

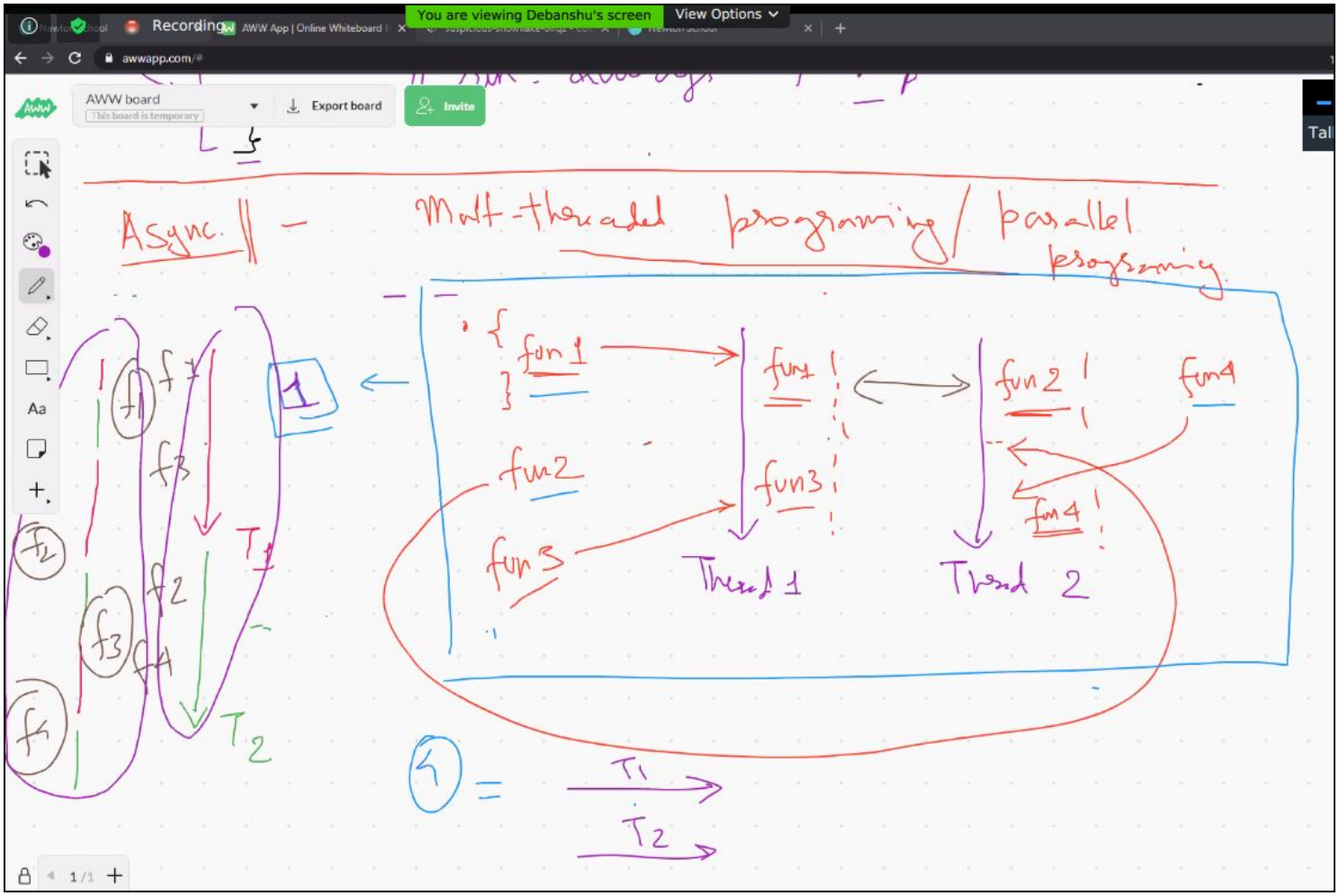


No guarantee of execution in multiple threading, we are not sure which thread will be completed first

Any statement can run at any time For single core. For multi core it will try to distribute the operations.

Even if you have 4 cores it will run parallely, its not like the code 1 thread is handled by one core and other thread will be handled by other core. Yes there are ways to do it but by default it is not happening. Thats why apple phones are optimised because they are written with respect to core. Bu android doesn't optimise the core as it has deal with so many different devices.

Relative order of threads



- Javascript is actually single thread as a default version
- There are ways to make js multithreaded but don't go there
- So the code will always run in the sequential order

```
<script>
  function getData() {
    console.log("ran step 1");
    fetch("http://www.google.com").then(() => {
      console.log("ran step 2");
    });
    console.log("ran step 3");
  }
```

Tell what is the output of get data function

Ans

Predicted ans

- Ran step 1
- Ran step 2
- Ran step 3

Calling getData(0 function

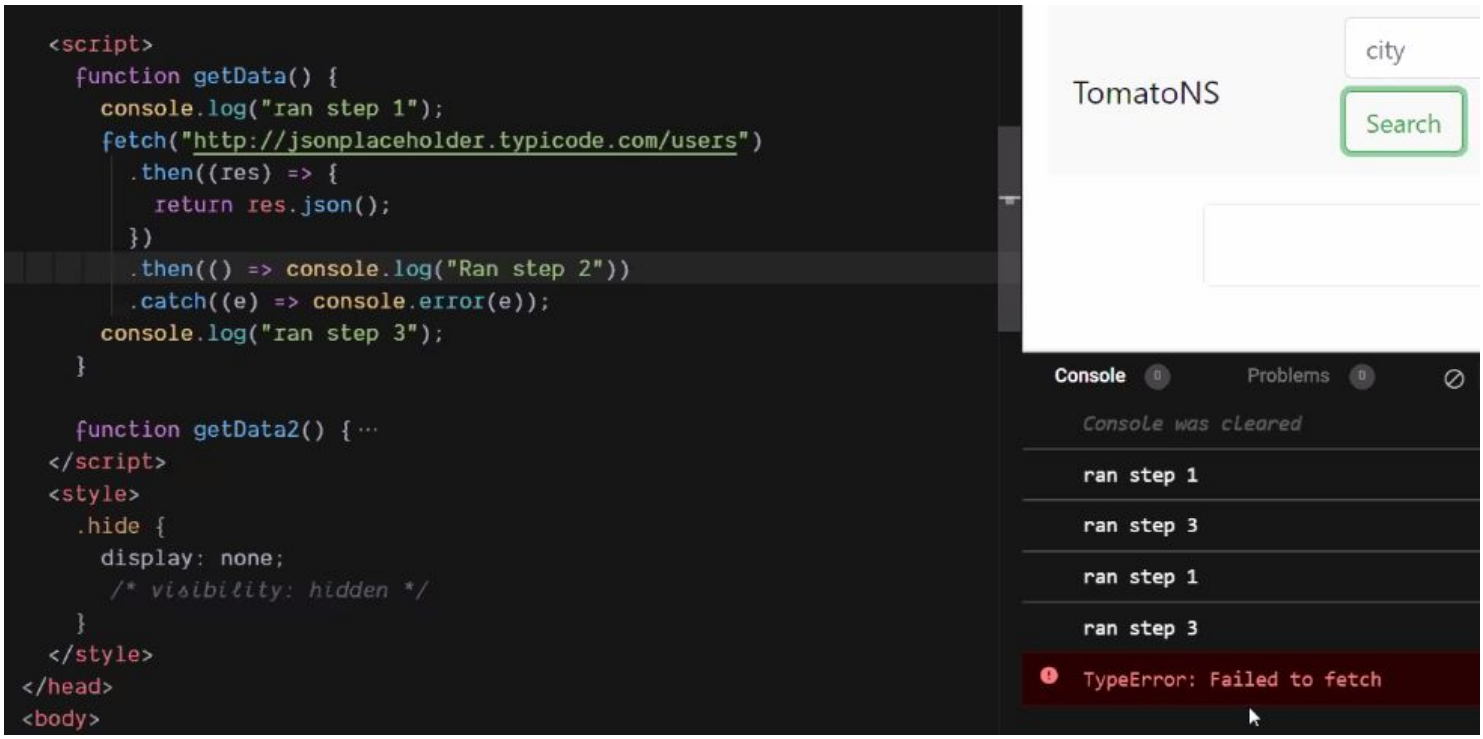
```
<script>
  function getData() {
    console.log("ran step 1");
    fetch("http://www.google.com").then(() => {
      console.log("ran step 2");
    });
    console.log("ran step 3");
  }
```

Step 2 did not come because google api can't hit using fetch function

```
<script>
  function getData() {
    console.log("ran step 1");
    fetch("http://jsonplaceholder.typicode.com/users").then(() => {
      console.log("ran step 2");
    });
    console.log("ran step 3");
  }
```

Again the same thing

Still didn't work



This introduces **.catch block** of fetch function

Try catch also not work in catching the error from fetch function. Only .catch function will catch the error from fetch function.

Handle error in api you need to write the .catch block

After writing ‘then’, then only use .catch

If any of the the then bock error occurs then .catch will catch the error

This is the second way of handling error.

Fetching our own file

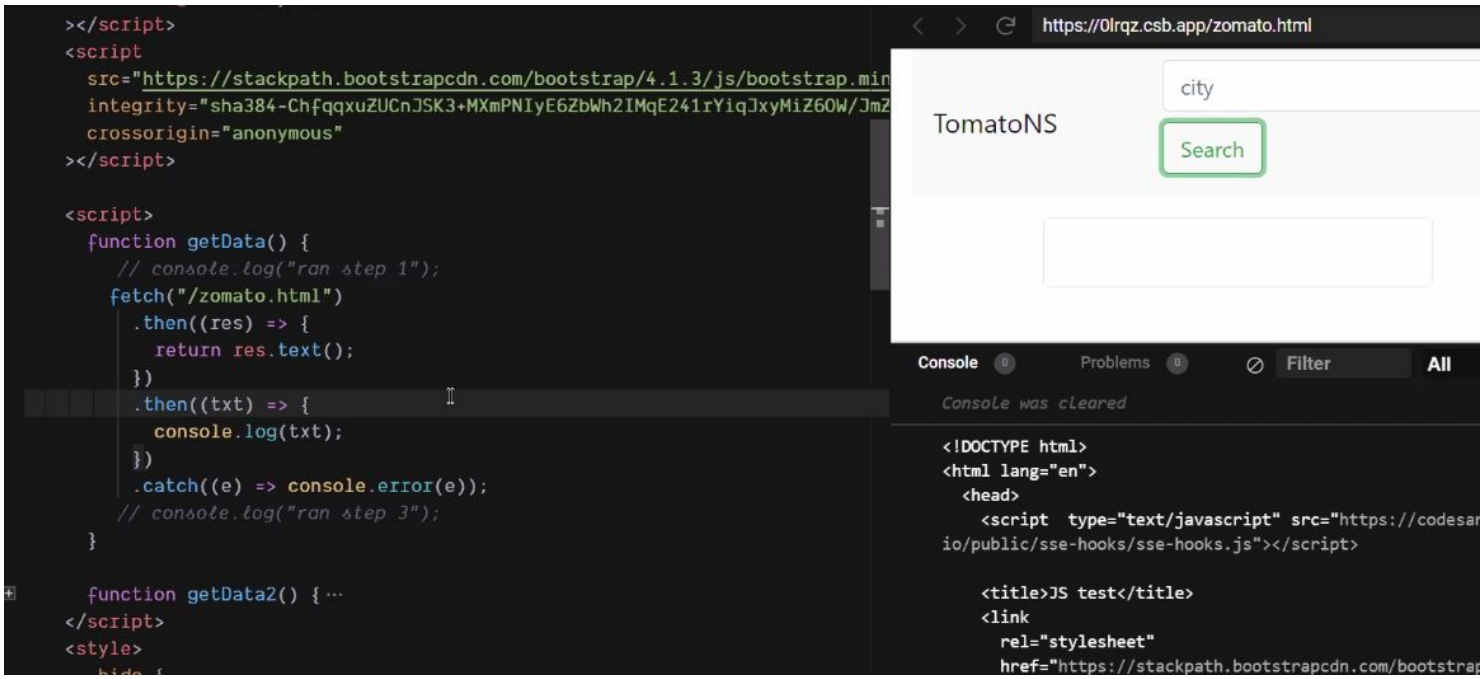
```
<script>
  function getData() {
    console.log("ran step 1");
    fetch("/zomato.html")
      .then((res) => {
        return res.json();
      })
      .then(() => {
        console.log(a);
      })
      .catch((e) => console.error(e));
    console.log("ran step 3");
  }
```

```
Console was cleared
ran step 1
ran step 3
SyntaxError: Unexpected token < in JSON at position 0
```

Working

Line 6 caused error because it is not json response it's just an html file not json.

If you try to do x.json() where x object is not json it won't work, in order to get that data of x use x.text()



Tried to fetch our current file

```
function getData() {
  console.log("ran step 1");
  fetch("/zomato.html")
    .then((res) => {
      return "some result";
    })
    .then(() => {
      console.log(a);
    })
    .catch((e) => console.error(e));
  console.log("ran step 3");
}
```

- Json.stringify converts the js object to json
- Html content cannot be converted to json so just convert it to text..
- x.content() is not a well recognized operation. There is no such this as res.content().

Function

- JSON.parse(). It takes a text and converts to js object

Example

```
const a = '{ "status": true} ';
console.log(a);
```

```
const a = '{ "status": true} ';
console.log(a);
const b = JSON.parse(a);
```

‘a’ have simple text. This converts the string to js object

```
const a = '{ "status": true} ';
console.log(a);
const b = JSON.parse(a);
```

It is just a reverse of stringify

- Hosting our own json in our server observe the file in fetch function /sample_json.json <- we created this file.

```
    // const a = '{ "status": true} ';
    // console.log(a);
    // const b = JSON.parse(a);
    // console.log(b);
    // console.log("ran step 1");
    fetch("/sample_json.json")
      .then((res) => {
        return res.json();
      })
      .then((body) => {
        console.log(body);
      })
      .catch((e) => console.error(e));
    // console.log("ran step 3");
  }

  function getData2() {···
</script>
<style>
  .hide {
```

```
<script>
  function getData() {
    // const a = '{ "status": true} ';
    // console.log(a);
    // const b = JSON.parse(a);
    // console.log(b);
    // console.log("ran step 1");
    fetch("/sample_json.json")
      .then((re (parameter) res: Response
        return res.text();
      })
      .then((body) => JSON.parse(body))
      .then((body) => {
        console.log(body);
      })
      .catch((e) => console.error(e));
    // console.log("ran step 3");
  }

  function getData2() {···
</script>
```

```
<script>
  function getData() {
    // const a = '{ "status": true} ';
    // console.log(a);
    // const b = JSON.parse(a);
    // console.log(b);
    console.log("ran step 1");
    fetch("/sample_json.json").then((res) => {
      console.log("ran step 2");
    });
    // .then((body) => {
    //   console.log(body);
    // })
    // .catch((e) => console.error(e));
    console.log("ran step 3");
  }

  function getData2() {···
</script>
<style>
```

```
Console  0  Proble
Console was cleared
ran step 1
ran step 3
ran step 2
```

Why this unsequential behavviour

- Ran step 1
- Ran step 3
- Ran step 2

Is not possible at all as javascript is single threaded. So ans to this will be given in next class. This is a concept of asynchronous behaviour.