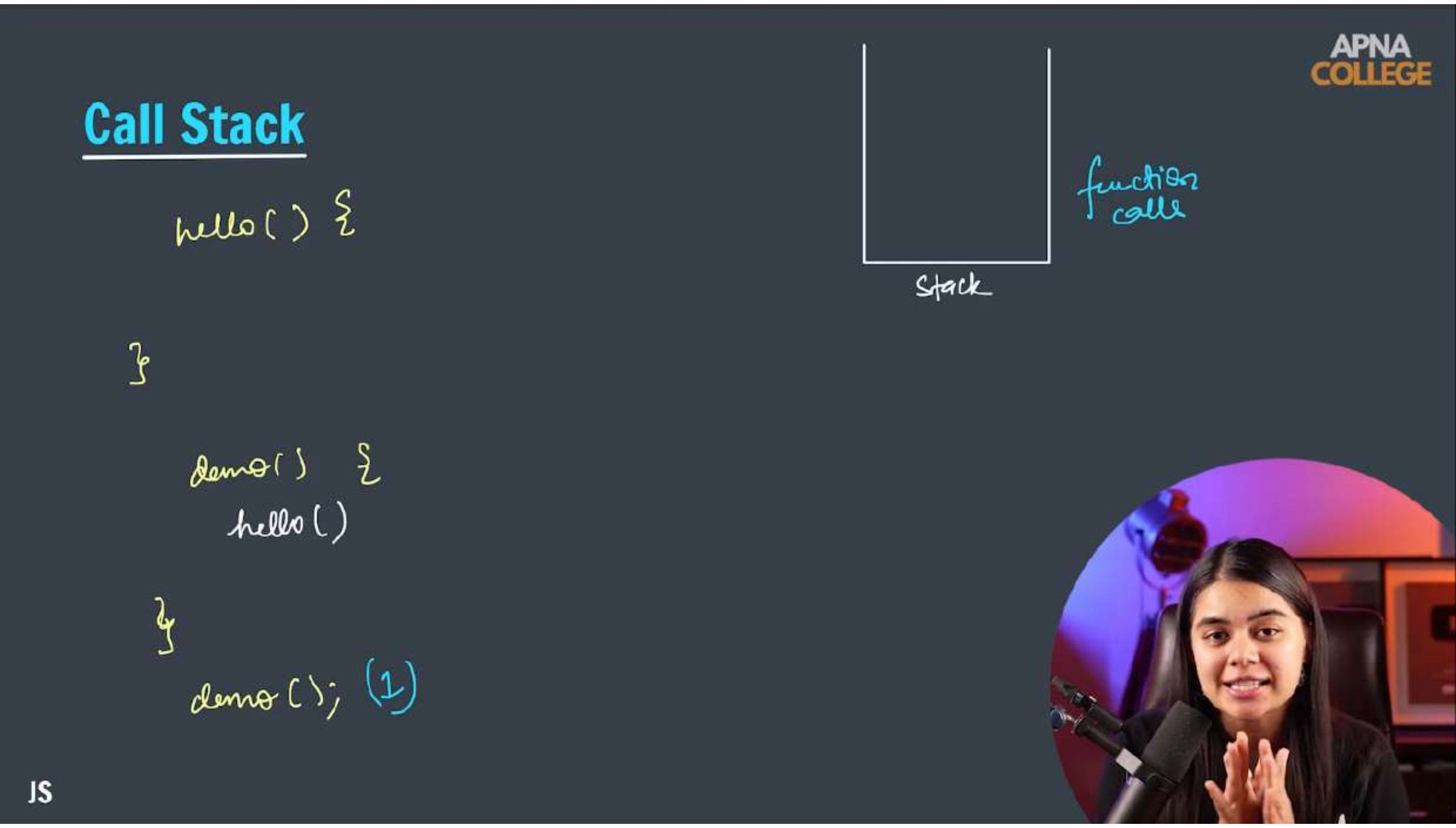


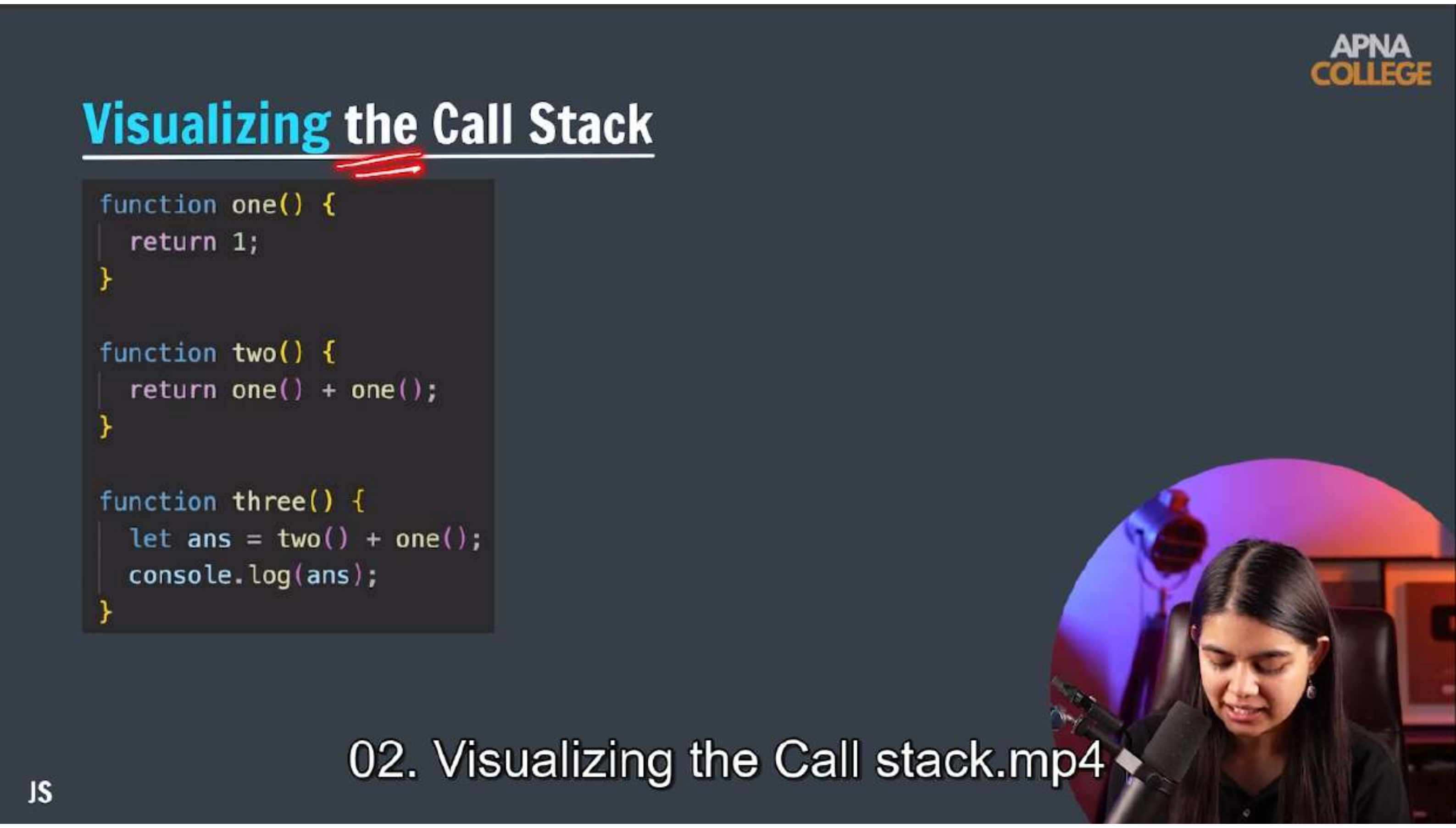
Call Stack

```
hello() {


}

 demo() {
   hello()


 }
 demo(); (1)
```



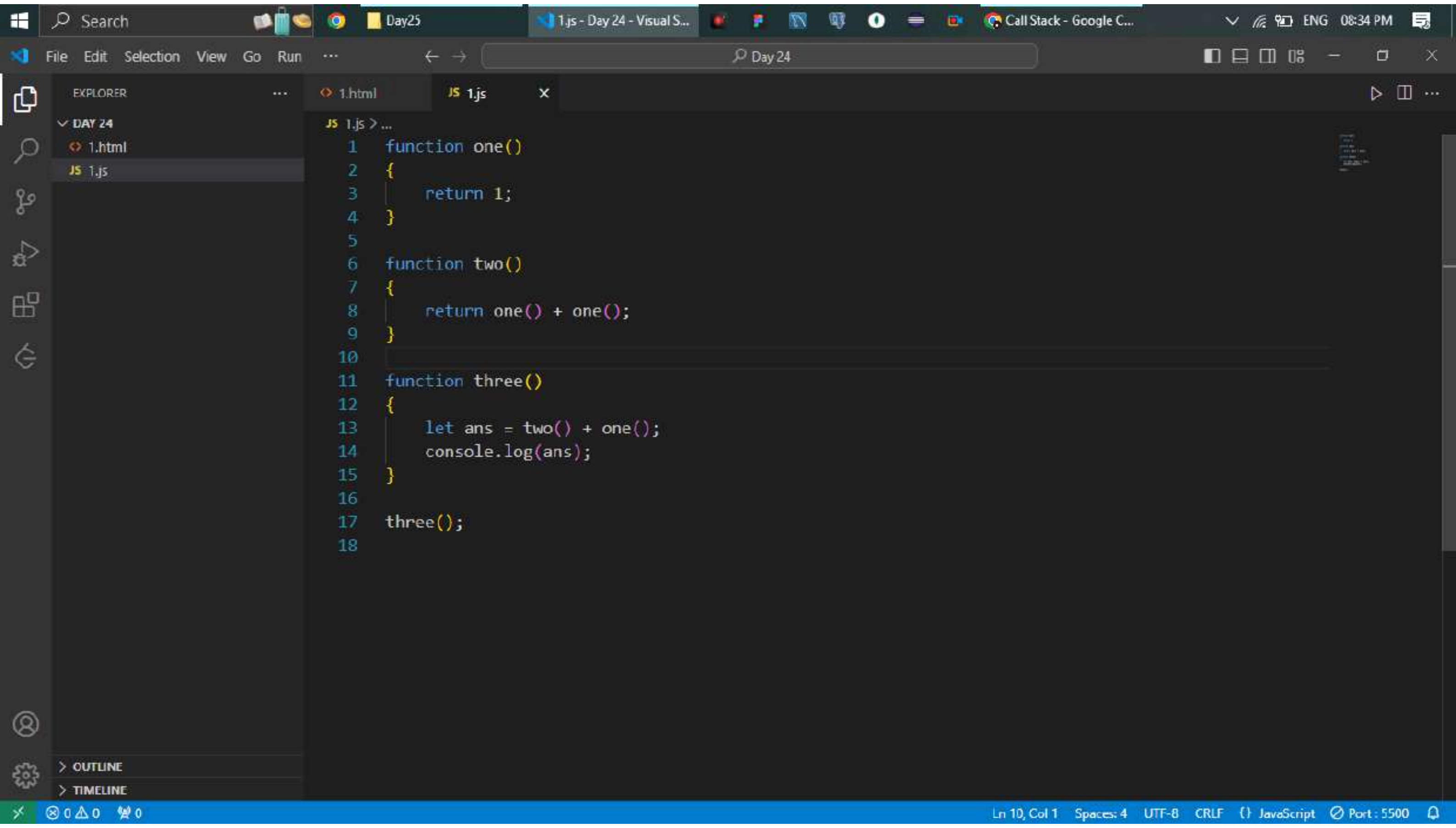
Visualizing the Call Stack

```
function one() {
  return 1;
}

function two() {
  return one() + one();
}

function three() {
  let ans = two() + one();
  console.log(ans);
}
```

02. Visualizing the Call stack.mp4



Search
Day25
1.js - Day 24 - Visual S...
Call Stack - Google C...
ENG 08:34 PM
File Edit Selection View Go Run ···
Day 24
EXPLORER
DAY 24
1.html
1.js
1.html
1.js
1.js > ...
OUTLINE
TIMELINE
⊗ 0 ⚠ 0 0
Ln 10, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript Port : 5500

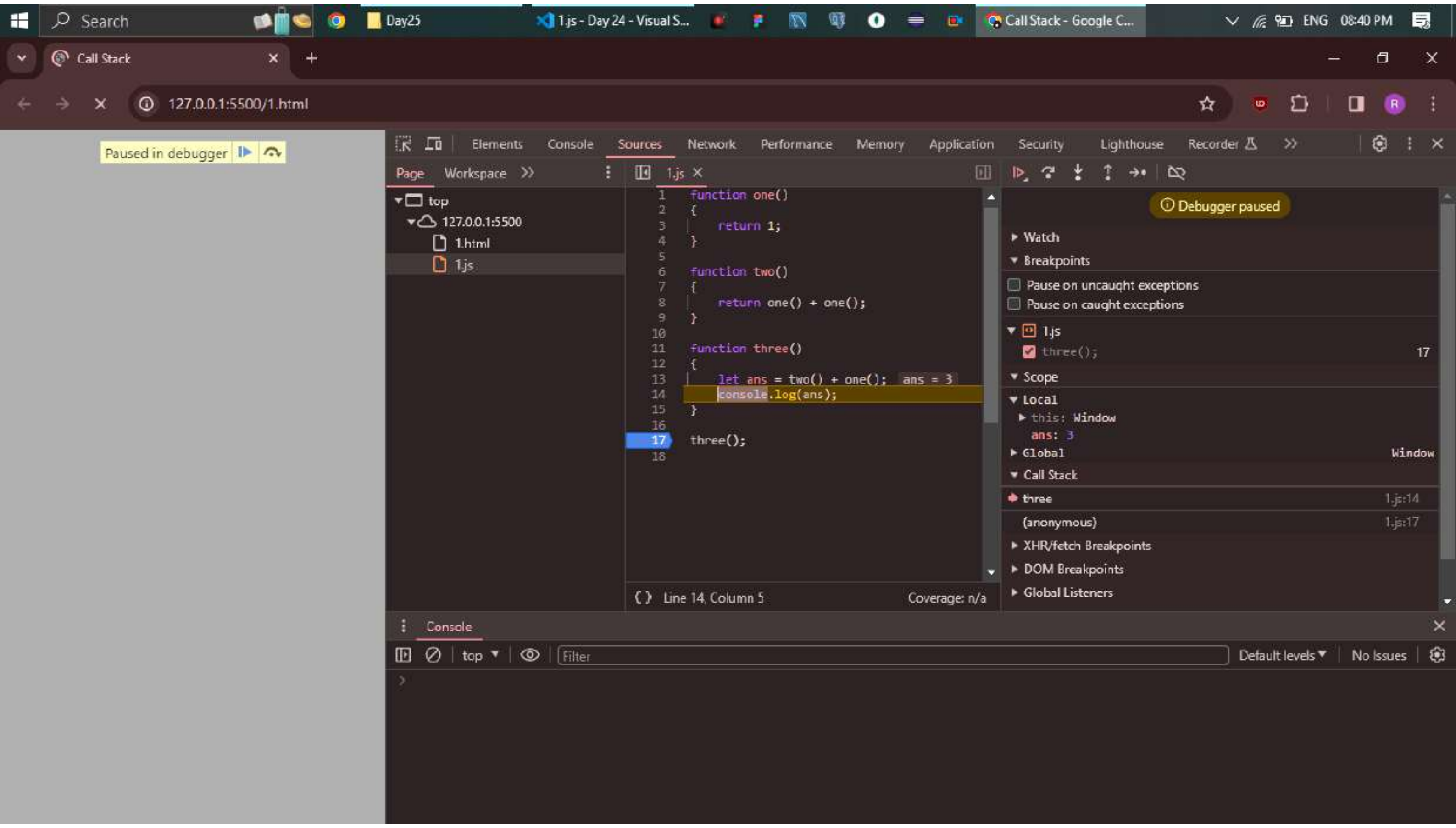
```
function one()
{
    return 1;
}

function two()
{
    return one() + one();
}

function three()
{
    let ans = two() + one();
    console.log(ans);
}

three();

```



```
function one()
{
    return 1;
}

function two()
{
    return one() + one();
}

function three()
{
    let ans = two() + one();  ans = 3
    console.log(ans);
}

three();
```

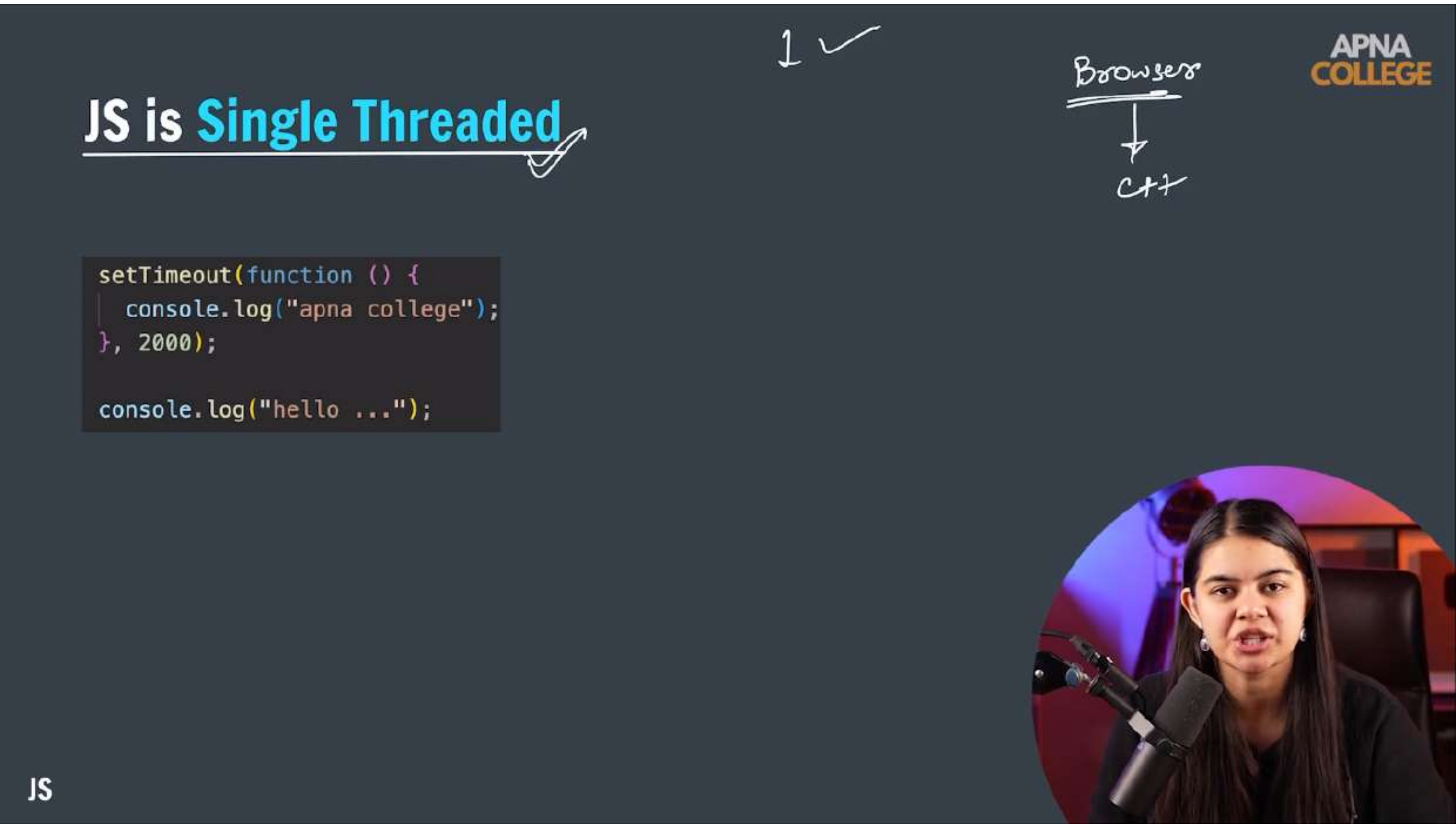
Line 14, Column 5 Coverage: n/a

□ Pause on uncaught exceptions
□ Pause on caught exceptions

1.js
☑ three(); 17

Scope
Local
this: Window
ans: 3
Global Window

Call Stack
three 1.js:14
(anonymous) 1.js:17



1 ✓

Browser

↓

C++

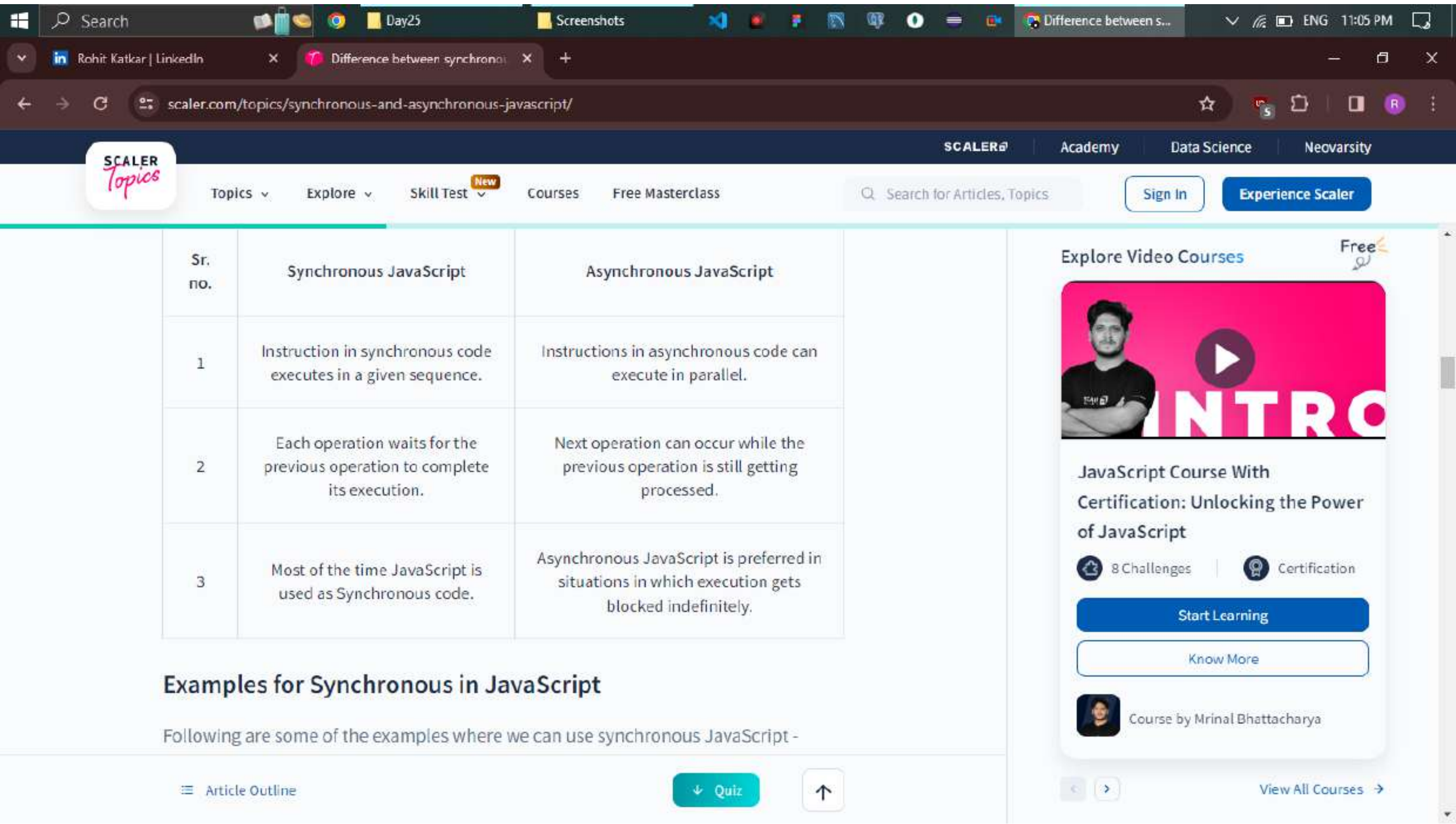
APNA COLLEGE

JS is Single Threaded ✓

```
setTimeout(function () {
  console.log("apna college");
}, 2000);

console.log("hello ...");
```

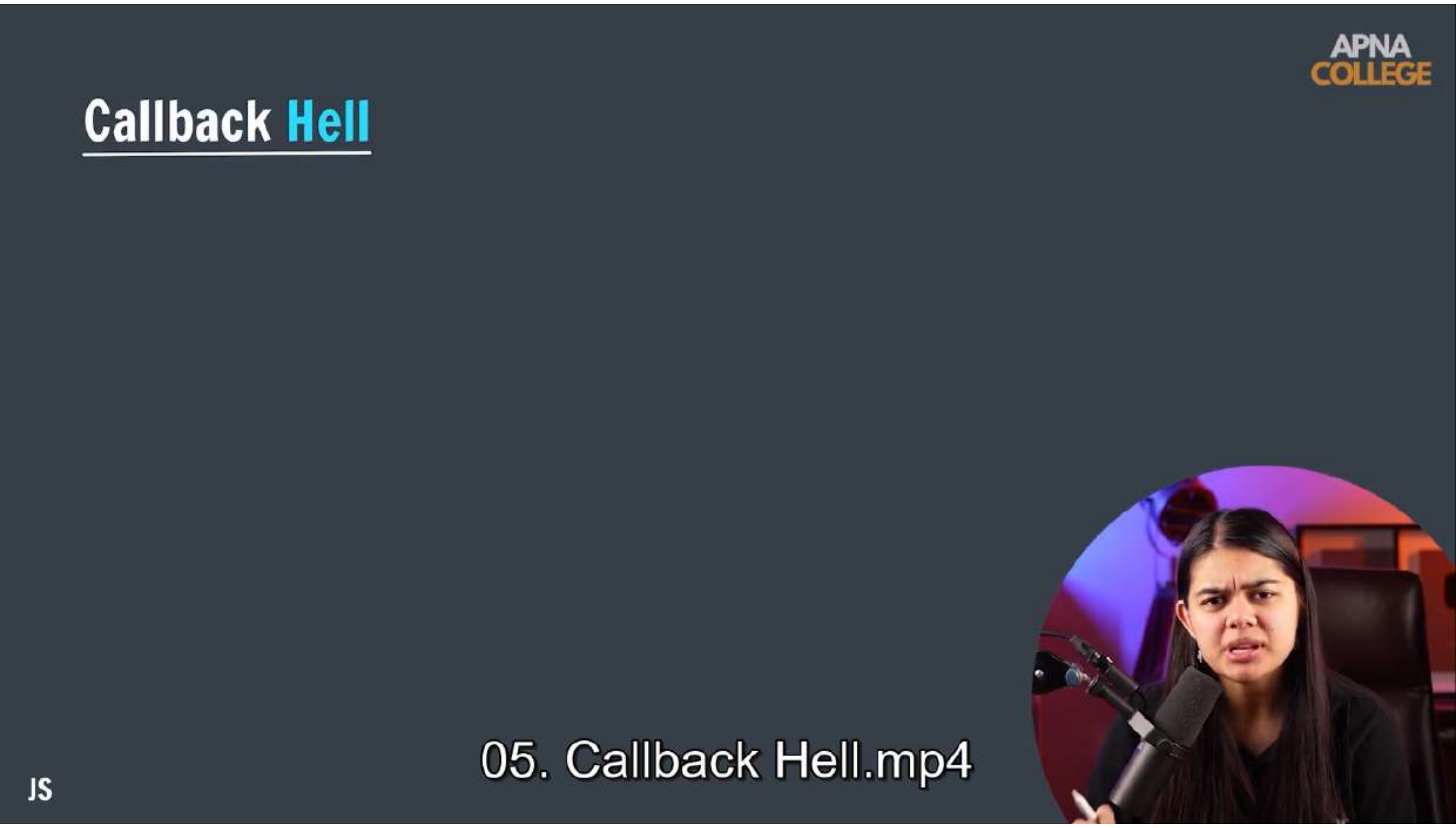
JS



Sr. no.	Synchronous JavaScript	Asynchronous JavaScript
1	Instruction in synchronous code executes in a given sequence.	Instructions in asynchronous code can execute in parallel.
2	Each operation waits for the previous operation to complete its execution.	Next operation can occur while the previous operation is still getting processed.
3	Most of the time JavaScript is used as Synchronous code.	Asynchronous JavaScript is preferred in situations in which execution gets blocked indefinitely.

Examples for Synchronous in JavaScript

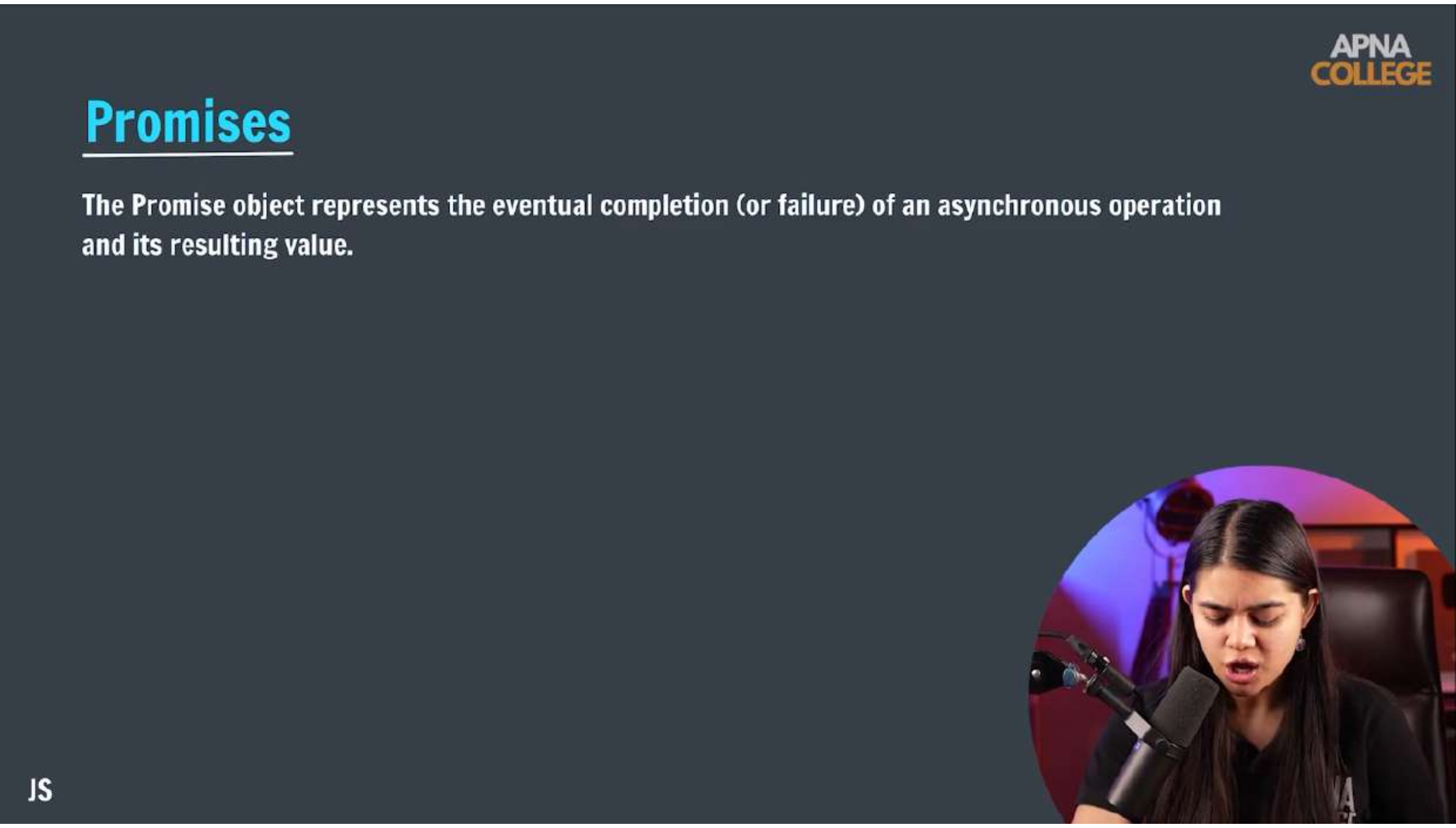
Following are some of the examples where we can use synchronous JavaScript -



Callback Hell

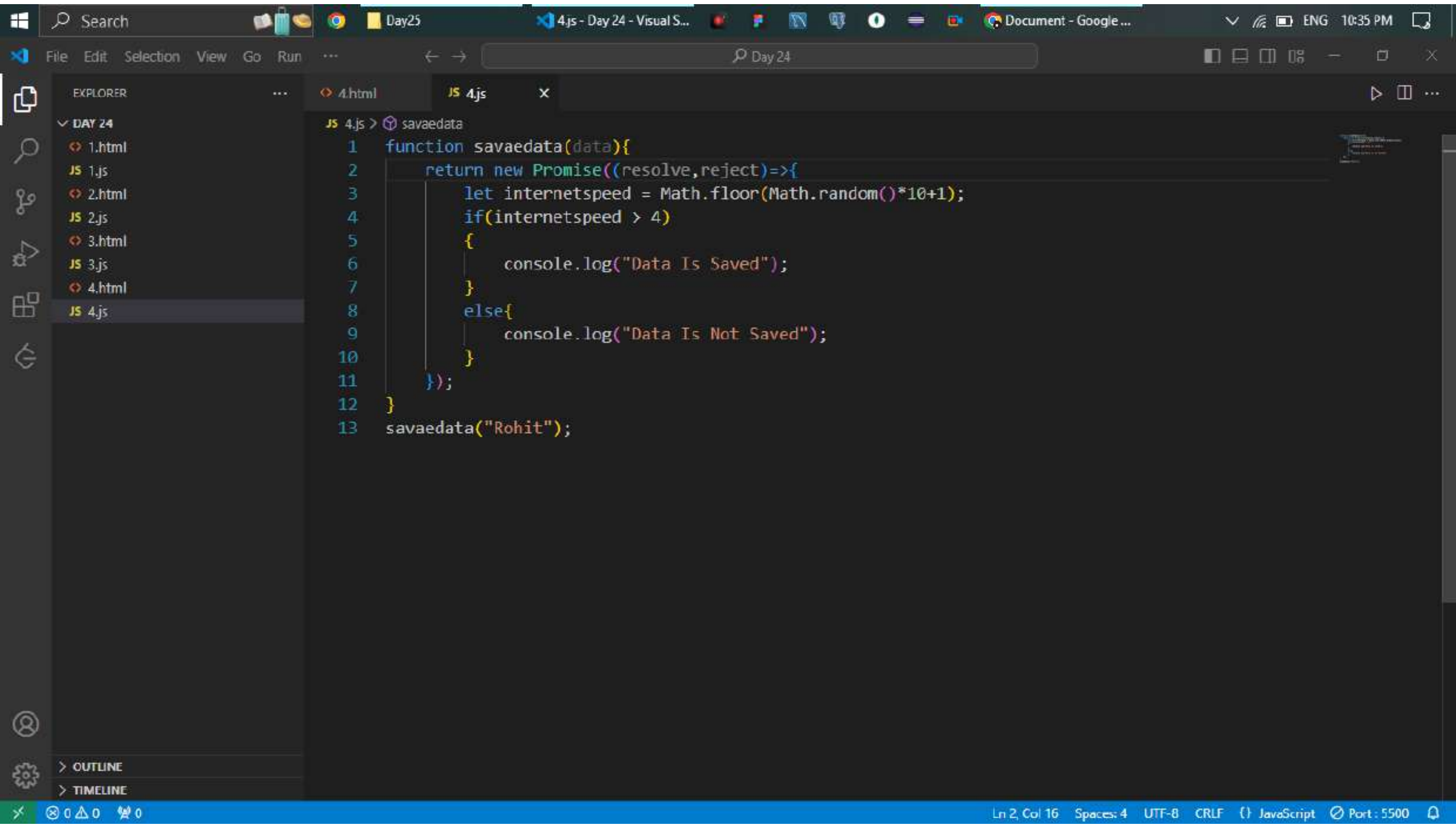
05. Callback Hell.mp4

JS

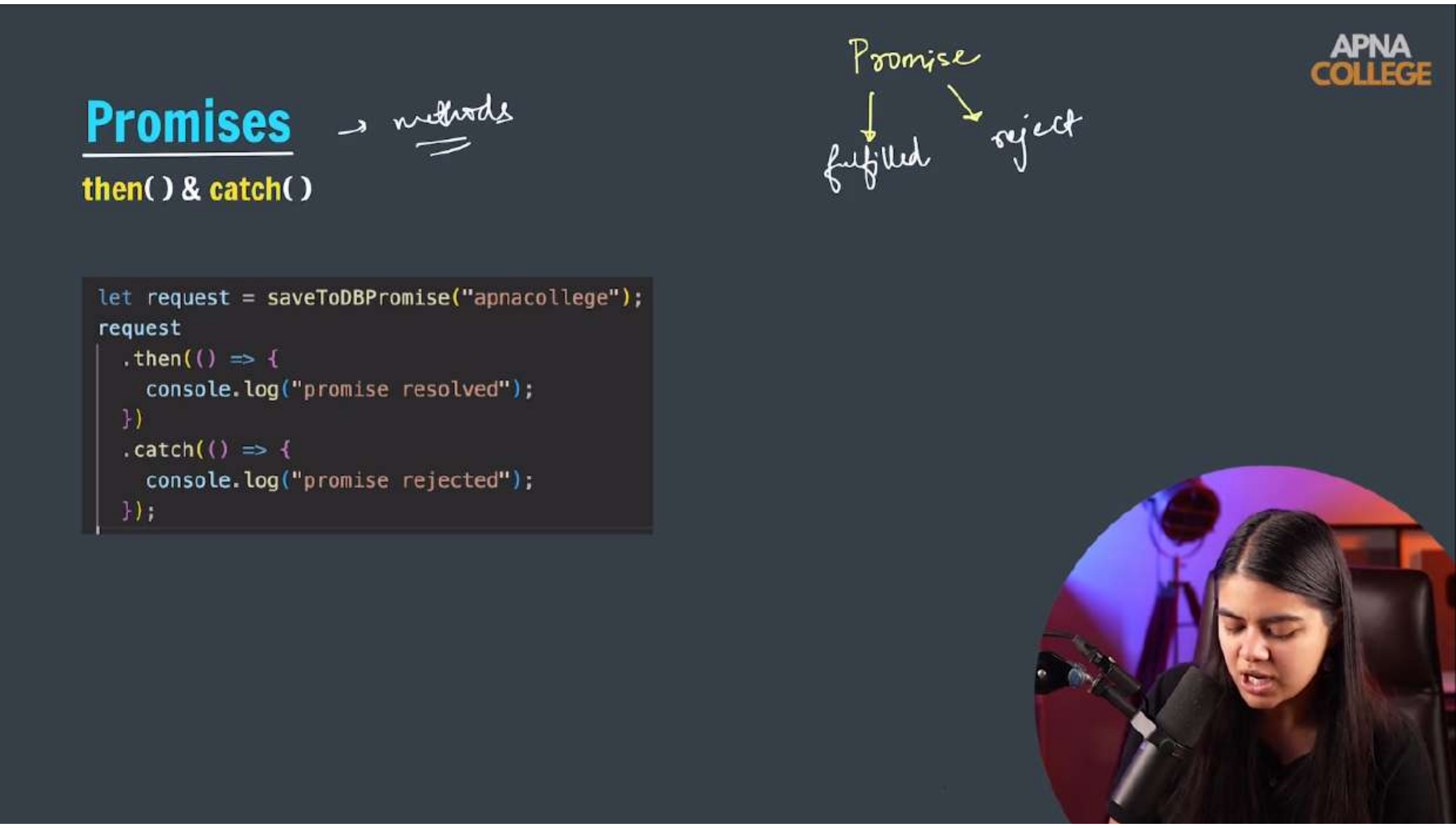


Promises

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



```javascript
function savaedata(data){
    return new Promise((resolve,reject)=>{
        let internetspeed = Math.floor(Math.random()*10+1);
        if(internetspeed > 4)
        {
            console.log("Data Is Saved");
        }
        else{
            console.log("Data Is Not Saved");
        }
    });
}
savaedata("Rohit");
```



Promises

→ methods

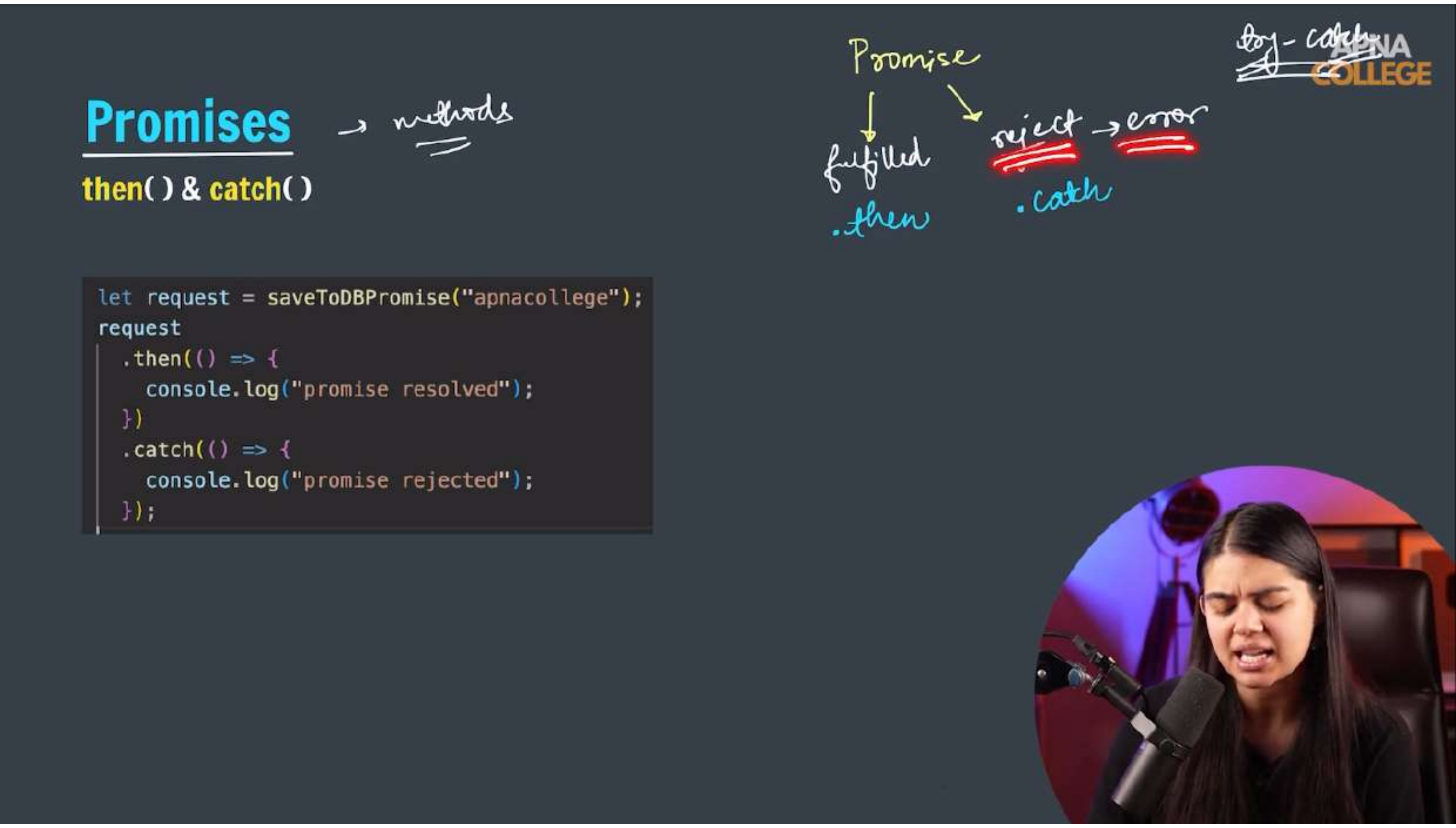
then() & catch()

```
let request = saveToDBPromise("apnacollege");
request
  .then(() => {
    console.log("promise resolved");
  })
  .catch(() => {
    console.log("promise rejected");
  });
```

Promise

↓ fulfilled

↘ reject

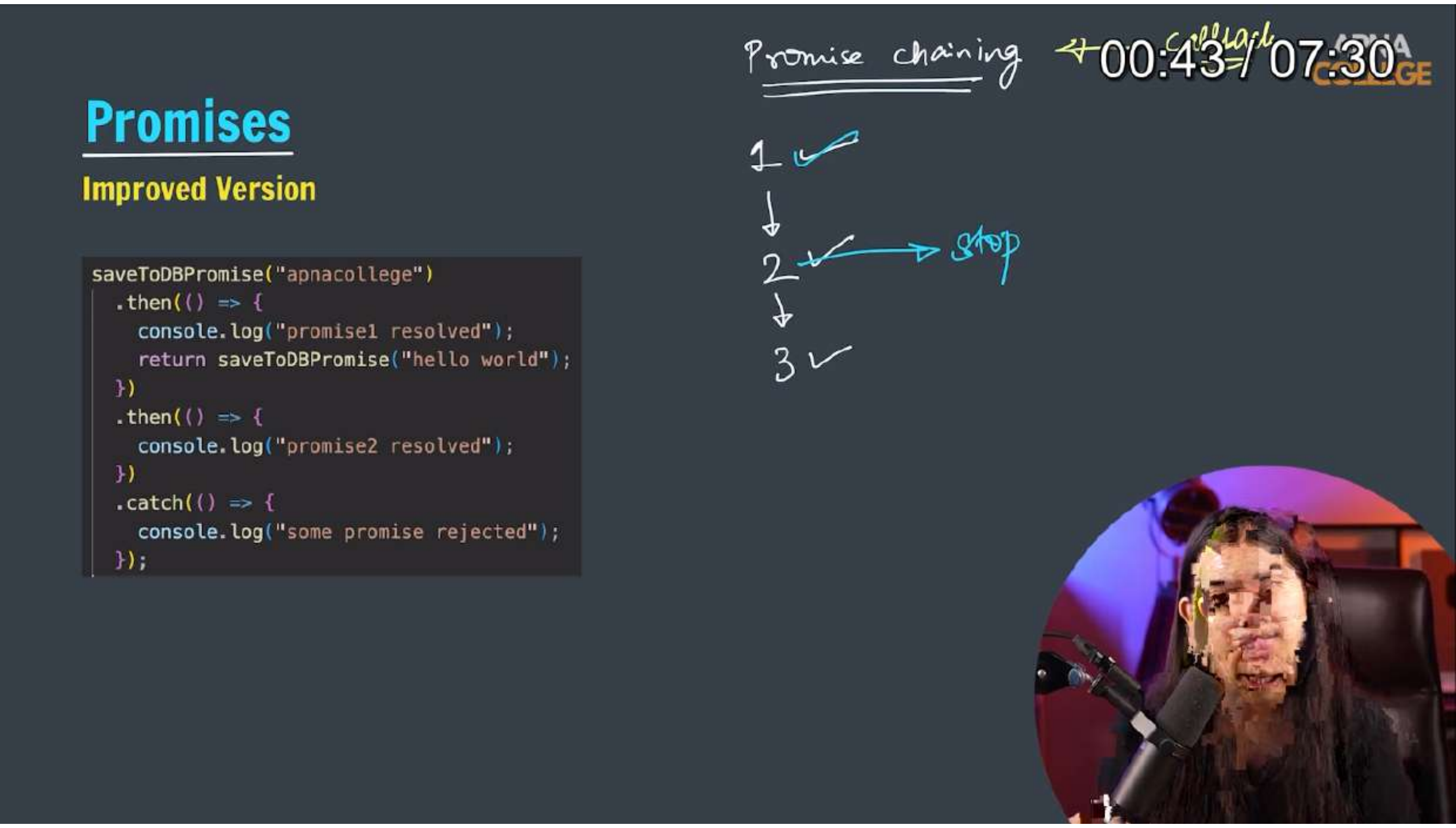


Promises

→ methods

then() & catch()

```
let request = saveToDBPromise("apnacollege");
request
  .then(() => {
    console.log("promise resolved");
  })
  .catch(() => {
    console.log("promise rejected");
  });
```



Promises

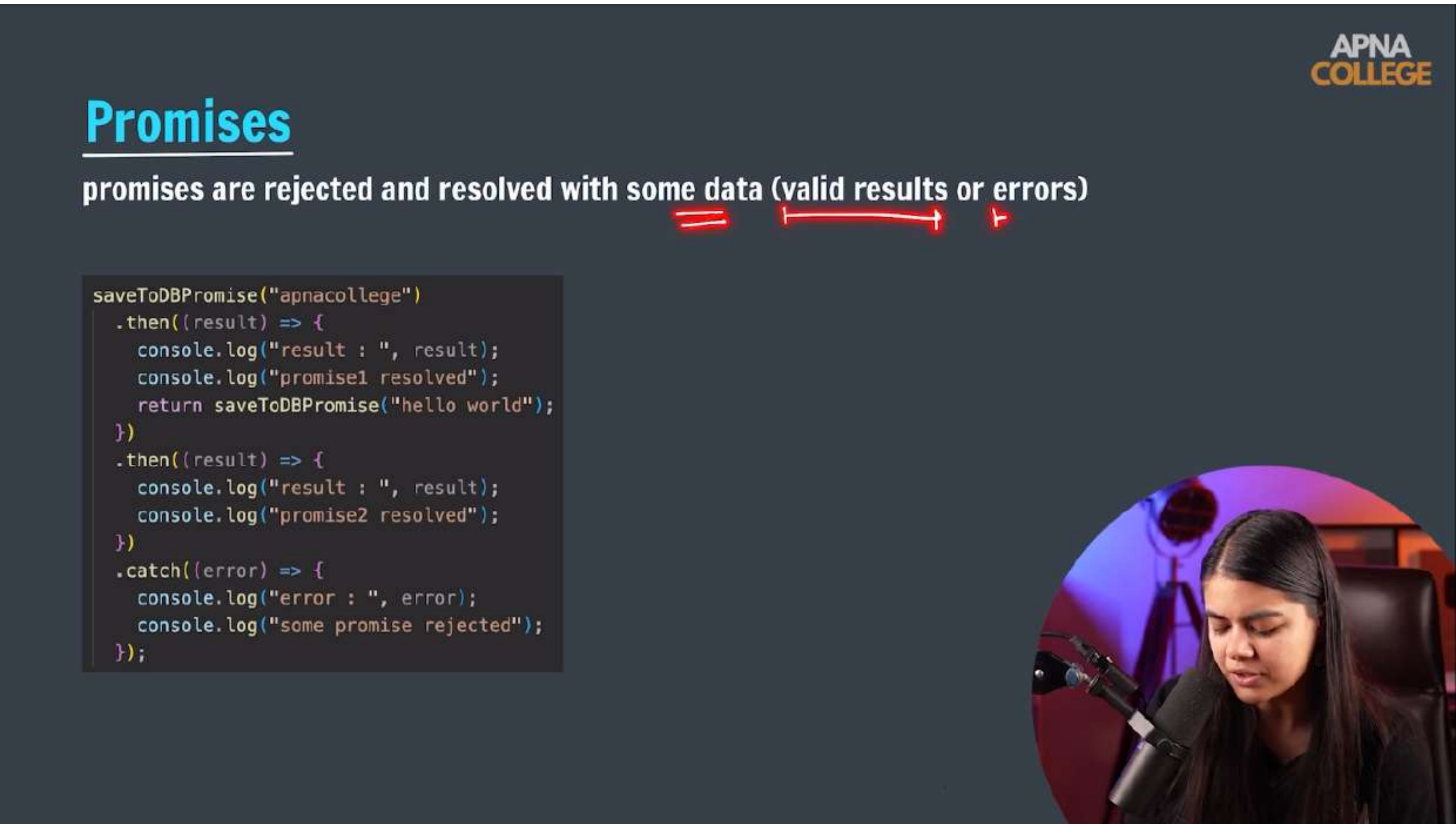
Improved Version

```
saveToDBPromise("apnacollege")
  .then(() => {
    console.log("promise1 resolved");
    return saveToDBPromise("hello world");
  })
  .then(() => {
    console.log("promise2 resolved");
  })
  .catch(() => {
    console.log("some promise rejected");
  });
```

Promise chaining ← callback

1 ✓
↓
2 ✓ → stop
↓
3 ✓

00:43 / 07:30



Promises

promises are rejected and resolved with some data (valid results or errors)

```
saveToDBPromise("apnacollege")
  .then((result) => {
    console.log("result : ", result);
    console.log("promise1 resolved");
    return saveToDBPromise("hello world");
  })
  .then((result) => {
    console.log("result : ", result);
    console.log("promise2 resolved");
  })
  .catch((error) => {
    console.log("error : ", error);
    console.log("some promise rejected");
  });
```