

Scopes

undefined vs undeclared

undefined is a variable state when the scopes already know about it but in the execution phase we have not allocated it a value.

undeclared is a variable state when we never formally declared a variable & before assigning

it a value so that it has chance to become
antiglobal, we try to use it.

```

1  var teacher = "Sanket";
2  function fun() {
3      var teacher = "Pulkit";
4      console.log(teacher, teachingAssistant);
5      teachingAssistant = "vibhav"; // eligible for becoming a
6
7  }
8
9  function gun() { // scope of gun
10     console.log(subject);
11     var subject = "JS";
12     console.log(teacher);
13 }
14
15 console.log(teacher); // Sanket
16 fun();
   gun();

```

Sanket → global

Pulkit → scope of fun

error →

phase → 1 → parsing

phase → 2 → execution

this is the case
where teachingAssistant
is undeclared.

```
1  var teacher = "Sanket";
2  function fun() {
3      var teacher = "Pulkit";
4      console.log(teacher, teachingAssistant);
5      teachingAssistant = "vibhav"; // eligible for becoming a
6  }
7
8  function gun() {
9      console.log(subject);
10     var subject = "JS";
11     console.log(teacher);
12 }
13
14 console.log(teacher); // Sanket
15 var gun();
```

Handwritten annotations on the code:

- Red arrow from line 1 to "global" (written in red).
- White arrow from line 2 to "scope of fun" (written in white).
- White arrow from line 9 to "scope of gun" (written in white).
- White arrow from line 11 to "subject" (written in white).
- Purple box around "subject" with "undefined" written above it.
- Purple text "Sanket" next to line 15.
- Purple text "gun();" next to line 16.

Phone → 1

Execution

This is the case
when subject is
declared but
undefined.

Node-JS-Backend-Batch / Oct 2 - Scopes 2 > Js autoGlobalist.js

```
1 → function fun() { // scope of fun ←  
2     teachingAssistant = "Vibhav";  
3     console.log(teachingAssistant);  
4     var teachingAssistant = "JD";  
5 }  
6  
7 fun();
```

global

JD

teachingAssistant

Node-JS-Backend-Batch > Oct 2 - Scores 2 > J

```
1  var fun = 10;
2  function gun() {
3      console.log("hello world");
4  }
5
6  gun(); → helloworld
7  fun(); → error → type error
```

Handwritten notes on the code:

- A red box around the number `10` in line 1, with an arrow pointing to the word `global` written in red above it.
- A red arrow pointing from the `10` in line 1 to the `fun` parameter in line 2.
- A red arrow pointing from the `gun` function name in line 2 to the `gun()` call in line 6.

Phase → 1
Phase-2

ode-JS-Backend-Batch > Oct 2 - Scopes 2

```
1  gun();  
2  function gun() {  
3      console.log("Shoooot");  
4  }  
5
```

global

Phase → 1

Phase → 2

Hoisting is a direct consequence of lexical parsing that happens in JS, due to which we are able to access some funcⁿ & variables before declaring them. which gives us a notion that they are moved up in the file.

Pulkit

global

```
1 → var teacher = "Sanket";  
2 function fun() {  
3     teacher = "pulkit";  
4     console.log("HI", teacher);  
5 }  
6  
7 fun();  
8 console.log(teacher); // pulkit
```

Phase → 1

Phase → 2

↳ HI pulkit

unhybrid

