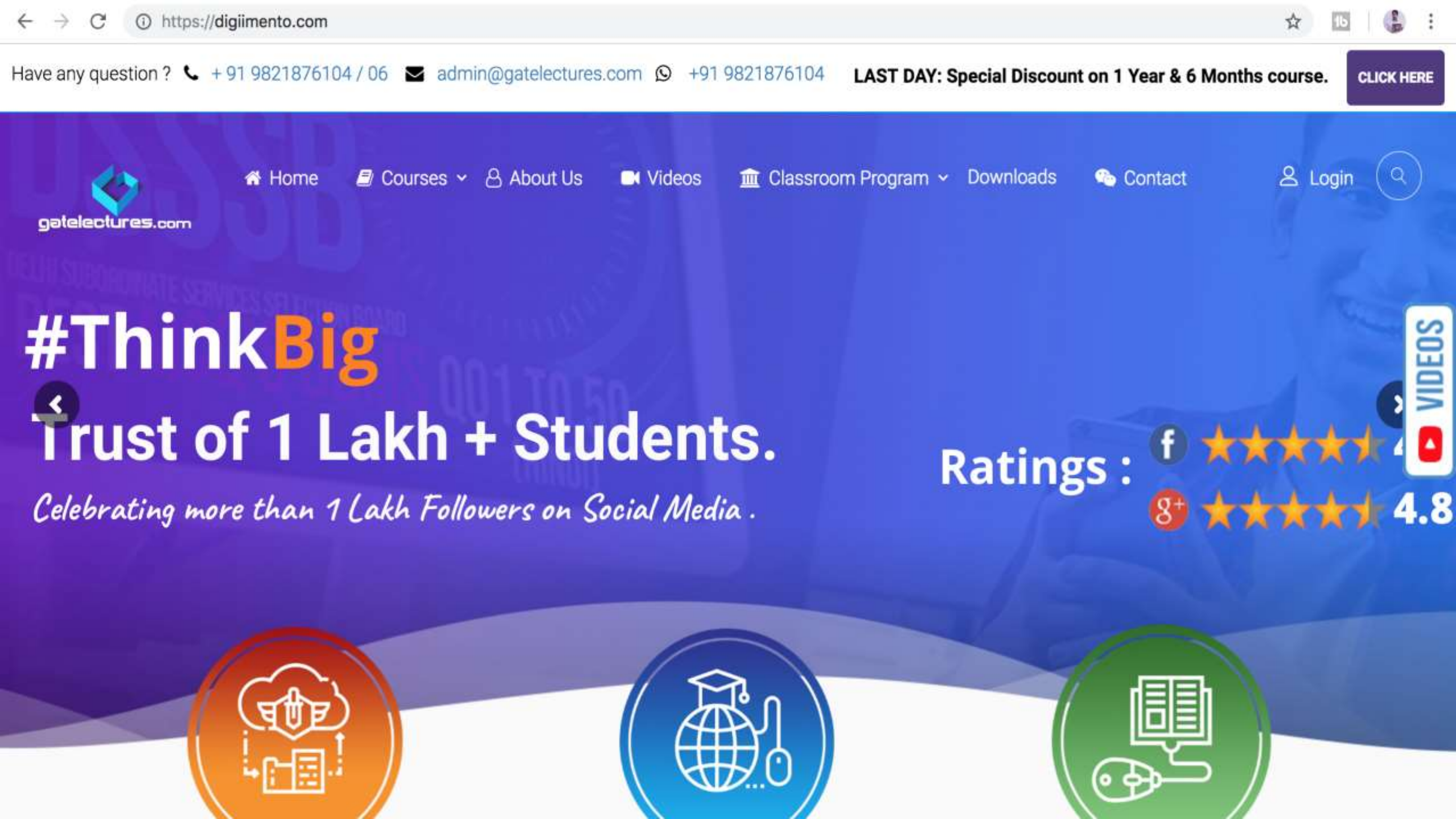
digimento

EDUCATING ON GO...

JAVA SCRIPT



https://digiimento.com

Have any question ? + 91 9821876104 / 06 admin@gatelectures.com +91 9821876104 **LAST DAY: Special Discount on 1 Year & 6 Months course.** CLICK HERE

gatelectures.com

Home · Courses · About Us · Videos · Classroom Program · Downloads · Contact · Login

#ThinkBig

Trust of 1 Lakh + Students.

Celebrating more than 1 Lakh Followers on Social Media .

Ratings : 4.8

three ways of variable declaration:

- let
- const
- Var

- *let and const behave exactly the same way in terms of Lexical Environments.*
- *But var is a very different, it originates from very old times. It's generally not used in modern scripts, but still lurks in the old ones*

- *Credits - https://javascript.info*

Variables

- To create a variable in JavaScript, we need to use the let keyword.
- The statement below creates (in other words: *declares* or *defines*) a variable with the name “message”:

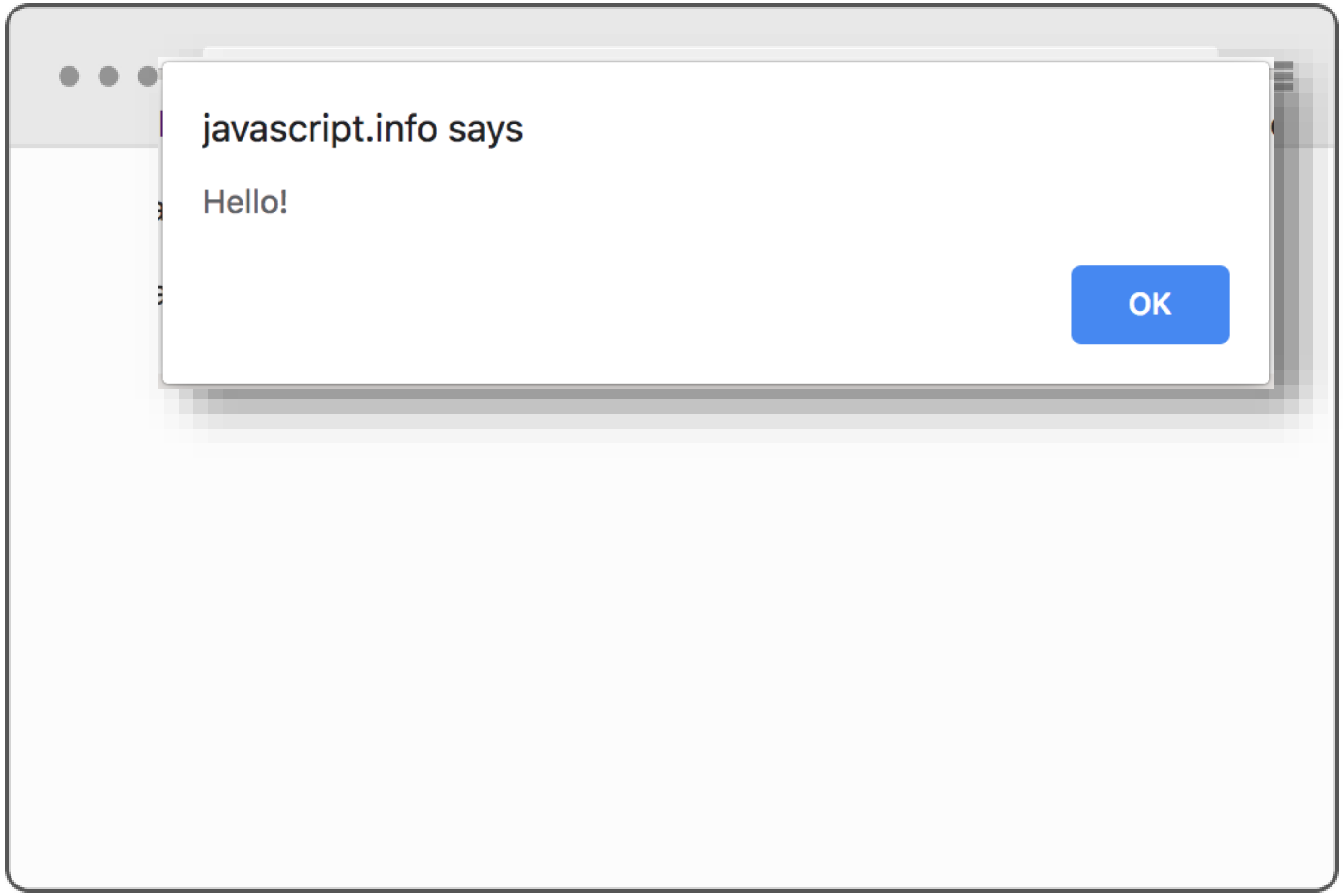
```
let message;
```

- Now we can put some data into it by using the assignment operator =:

```
let message;

message = 'Hello'; // store the string
```

Now we can put some data into it by using the assignment operator =:

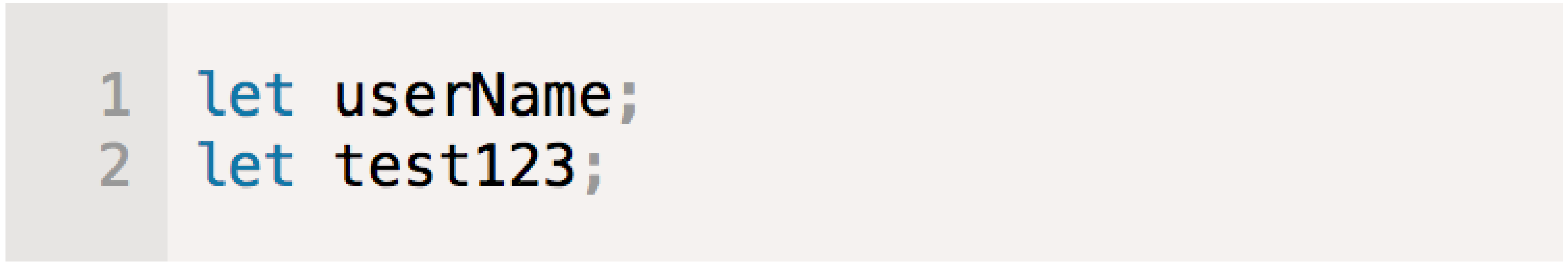
```
<!DOCTYPE html>
<html>
<body>
  <script>
    'use strict';
    let message;
    message = 'Hello!';

    alert(message);
  </script>
</body>
</html>
```


javascript.info says

Hello!

Variable naming

- There are two limitations for a variable name in JavaScript:
- The name must contain only letters, digits, symbols $ and _.
- The first character must not be a digit.
- Valid names, for instance:



```
let userName;
let test123;
```

```
<html>

<body>
  <script>
    'use strict';
    let $ = 1; // declared a variable with the name "$"
    let _ = 2; // and now a variable with the name "_"

    alert($ + _); // 3
  </script>
</body>

</html>
```

Examples of incorrect variable names:

```
let 1a; // cannot start with a digit

let my-name; // a hyphen '-' is not allowed in the name
```

Case matters

Variables named `apple` and `AppLE` – are two different variables.

⚠ Reserved names

There is a list of reserved words, which cannot be used as variable names, because they are used by the language itself.

For example, words `let`, `class`, `return`, `function` are reserved.

The code below gives a syntax error:

```
let let = 5; // can't name a variable "let", error!
let return = 5; // also can't name it "return", error!
```

⚠ An assignment without `use strict`

Normally, we need to define a variable before using it. But in the old times, it was technically possible to create a variable by a mere assignment of the value, without `let`. This still works now if we don't put `use strict`. The behavior is kept for compatibility with old scripts.

```javascript
// note: no "use strict" in this example

num = 5; // the variable "num" is created if didn't exist

alert(num); // 5
```

That's a bad practice, it gives an error in the strict mode:

```javascript
"use strict";

num = 5; // error: num is not defined
```

Constants

To declare a constant (unchanging) variable, one can use `const` instead of `let`:

```javascript
const myBirthday = '18.04.1982';
```

Variables declared using `const` are called “constants”. They cannot be changed. An attempt to do it would cause an error:

```javascript
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

When a programmer is sure that the variable should never change, they can use `const` to guarantee it, and also to clearly show that fact to everyone.

Uppercase constants

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.

Such constants are named using capital letters and underscores.

Like this:

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Benefits:

- `COLOR_ORANGE` is much easier to remember than `"#FF7F00"`.
- It is much easier to mistype in `"#FF7F00"` than in `COLOR_ORANGE`.
- When reading the code, `COLOR_ORANGE` is much more meaningful than `#FF7F00`.

Being a “constant” just means that the value never changes. But there are constants that are known prior to execution (like a hexadecimal value for red), and there are those that are *calculated* in run-time, during the execution, but do not change after the assignment.

For instance:

```javascript
const pageLoadTime = /* time taken by a webpage to load */;
```

The value of `pageLoadTime` is not known prior to the page load, so it’s named normally. But it’s still a constant, because it doesn’t change after assignment.

`var` instead of `let`

In older scripts you may also find another keyword: `var` instead of `let`:

```
var message = 'Hello';
```

The `var` keyword is *almost* the same as `let`. It also declares a variable, but in a slightly different, "old-school" fashion.

From the first sight, `var` behaves similar to `let`. That is, declares a variable:

```
function sayHi() {
  var phrase = "Hello"; // local variable, "var" instead of "let"

  alert(phrase); // Hello
}

sayHi();

alert(phrase); // Error, phrase is not defined
```

...But here are the differences.

"var" has no block scope

`var` variables are either function-wide or global, they are visible through blocks.

For instance:

```
if (true) {
  var test = true; // use "var" instead of "let"
}

alert(test); // true, the variable lives after if
```

If we used `let test` on the 2nd line, then it wouldn't be visible to `alert`. But `var` ignores code blocks, so we've got a global `test`.

The same thing for loops: `var` cannot be block- or loop-local:

```
for (var i = 0; i < 10; i++) {
  // ...
}

alert(i); // 10, "i" is visible after loop, it's a global variable
```

If a code block is inside a function, then `var` becomes a function-level variable:

```
function sayHi() {
  if (true) {
    var phrase = "Hello";
  }

  alert(phrase); // works
}

sayHi();
alert(phrase); // Error: phrase is not defined
```

As we can see, `var` pierces through `if`, `for` or other code blocks. That's because a long time ago in JavaScript blocks had no Lexical Environments. And `var` is a reminiscence of that.

“var” are processed at the function start

`var` declarations are processed when the function starts (or script starts for globals).

In other words, `var` variables are defined from the beginning of the function, no matter where the definition is (assuming that the definition is not in the nested function).

So this code:

```
function sayHi() {
  phrase = "Hello";

  alert(phrase);

  var phrase;
}
```

...Is technically the same as this (moved `var phrase` above):

```
function sayHi() {
  var phrase;

  phrase = "Hello";

  alert(phrase);
}
```

...Or even as this (remember, code blocks are ignored):

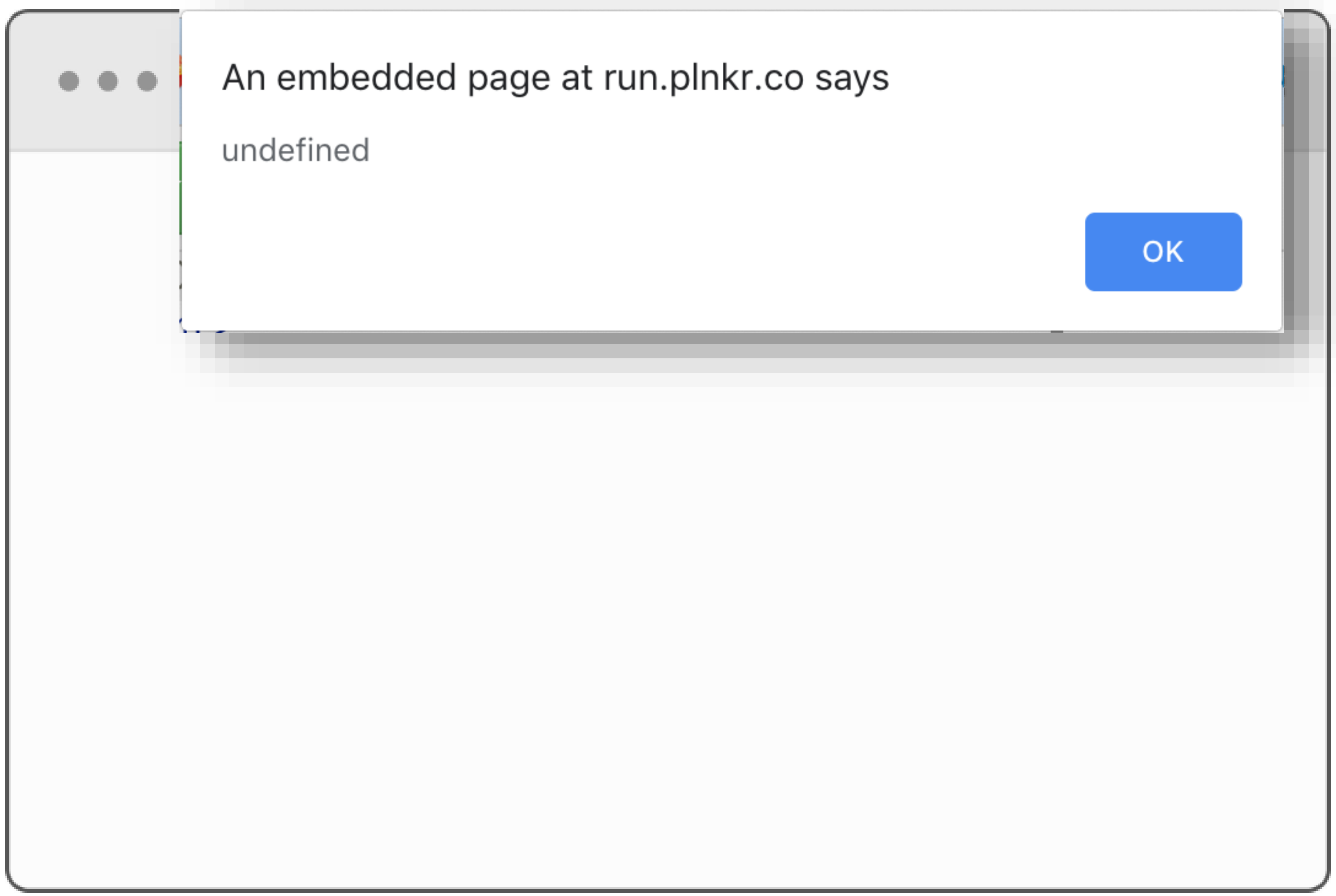
```
function sayHi() {
  phrase = "Hello"; // (*)

  if (false) {
    var phrase;
  }

  alert(phrase);
}
```

People also call such behavior “hoisting” (raising), because all `var` are “hoisted” (raised) to the top of the function.

So in the example above, `if (false)` branch never executes, but that doesn’t matter. The `var` inside it is processed in the beginning of the function, so at the moment of `(*)` the variable exists.

Declarations are hoisted, but assignments are not.

```
<html>

<body>
  <script>
    'use strict';
    function sayHi() {
      alert(phrase);

      var phrase = "Hello";
    }

    sayHi();
  </script>
</body>

</html>
```



An embedded page at run.plnkr.co says

```
undefined
```

- The line var phrase = "Hello" has two actions in it:
- Variable declaration var
- Variable assignment =.
- The declaration is processed at the start of function execution (“hoisted”), but the assignment always works at the place where it appears

```
<body>
  <script>
    'use strict';
    function sayHi() {
      var phrase; // declaration works at the start...

      alert(phrase); // undefined

      phrase = "Hello"; // ...assignment - when the execution reaches it.
    }

    sayHi();
  </script>
</body>
```