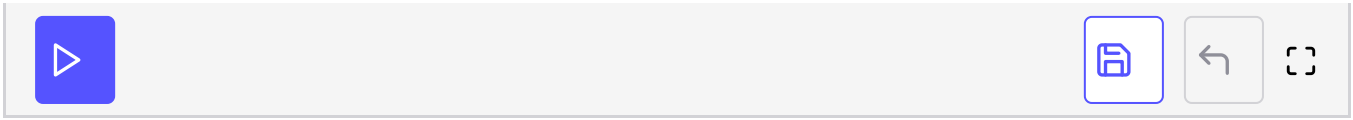
Variable Hoisting

Understand variable hoisting in JavaScript. Learn what the engine does behind the scenes to our variable and function declarations and how this results in unexpected behavior. Learn the difference between variable hoisting and function hoisting.

Variable Hoisting

What do you think the following code will log to the console? Take a guess and then run the code to check.

```
console.log(x); // -> ?
var x = 5;
console.log(x); // -> ?
console.log(y); // -> ?
```



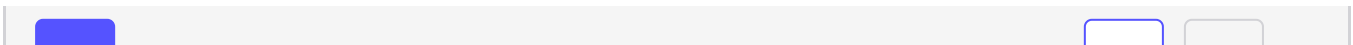
The second console statement prints `5` as expected, but the first prints `undefined`. But the last statement prints an error. What gives? Why do we see an error for `y` but not for `x`?

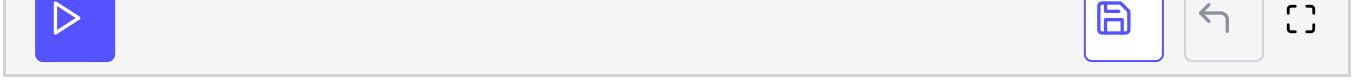
The JavaScript Engine

This is because the JavaScript engine alters the structure of our code before it runs it.

Variable declarations are hoisted, or moved, to the top of their available scope. We're currently in the global scope, so they'll be hoisted to the top. The code that our Javascript engine runs looks more like this.

```
var x;
console.log(x); // -> undefined
x = 5;
console.log(x); // -> 5
console.log(y); // -> Uncaught ReferenceError: y is not defined
```





Note that only the variable *declaration*, not the *assignment*, is hoisted. Assignment means actually giving it a value, while declaration means creating the variable. The variable `x` is declared at the beginning of our code block, but doesn't receive the value of `5` until later. A declared but unassigned variable automatically gets the value of `undefined`, so that's what we see printed to the console with the first statement.

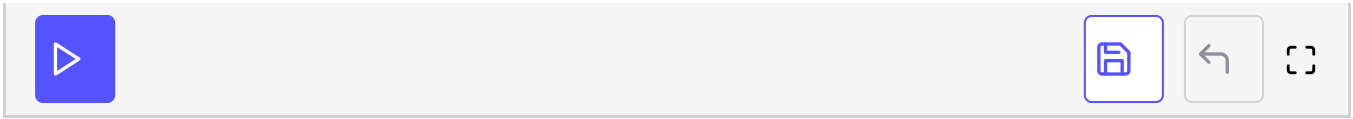
We never declare or assign a value to `y`, so there's nothing to hoist and we see a reference error.

Hoisting and `var`

A variable that is not declared using `var` will not be hoisted. We'll get an error.

```
console.log(x);
console.log(y);

var x = 'This will log "undefined"!';
y = 'This will throw an error :(';
```



Best Practices

Because of hoisting, some developers consider it a best practice to declare all variables at the top of their scope. By doing so, the code they write is closer to the engine's interpretation of it. All four of these code blocks are the same to the engine.

```
x = 17;
var y = 20;
var x;


x = 17;
y = 20;
var x;
var y;
```

```
var x = 17;
var y = 20;


var x;
var y;
x = 17;
y = 20;
```

The last block is the most accurate representation of our code. You might argue that the 2nd-to-last block is shorter and more readable, and you'd be right. It's up to you to decide which style you prefer. Just don't ever write code that looks like one of the first two blocks!

Function Hoisting

Function hoisting is a little different. There are two way to write a function. The first is referred to as a *function expression*.

```
var fn = function() {
    // do something...
}
```

The second is called a *function declaration*.

```
function fn() {
    // do something...
}
```

Note that this definition of declaration doesn't match the definition of variable declaration.

A function expression gets hoisted just like a normal variable. The variable goes to the top of the scope and gets the value `undefined`, later being set to the function itself at the expected place in the code.

A function declaration, on the other hand, gets hoisted in its entirety. The variable gets moved to the top and immediately is equal to the function. It even gets hoisted higher than normal variable initializations.

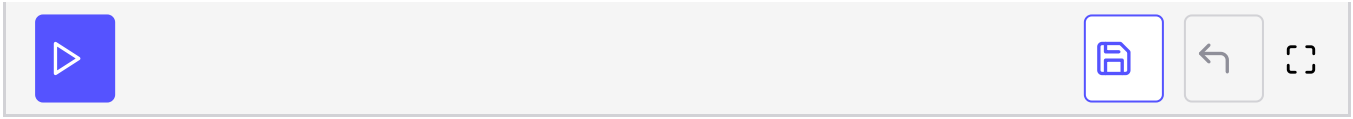
Examples

This explains the following behavior.

```
fnDeclaration(); // -> This works!
fnExpression(); // -> Uncaught TypeError: fnExpression is not a function

function fnDeclaration() {
    console.log('This works!');
}

var fnExpression = function() {
    console.log("This won't work :(");
}
```



Full Example

Here's a sample of code and what the compiler turns it into.

```
var a = 123;
var b = 'abc';

var fnExpression = function() {
    var c = 456;
    var d = 'def';
}

function fnDeclaration() {
    var e = 789;
}
```

The final version of this code that the compiler will run looks more like this.

```
function fnDeclaration() {
    var e;

    e = 789;
}

var a;
var b;
var fnExpression;

a = 123;
b = 'abc';

fnExpression = function() {
    var c;
    var d;

    c = 456;
    d = 'def';
```

```
}
```

That’s it.