

Variables: var, let, const

Declare a variable in JS with one of three keywords:

```
// Function scope variable
```

```
var x = 15;
```

```
// Block scope variable
```

```
let fruit = 'banana';
```

```
// Block scope constant; cannot be reassigned
```

```
const isHungry = true;
```

You do not declare the datatype of the variable before using it
("dynamically typed")

Function parameters

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
}
```

Function parameters are **not** declared with `var`, `let`, or `const`

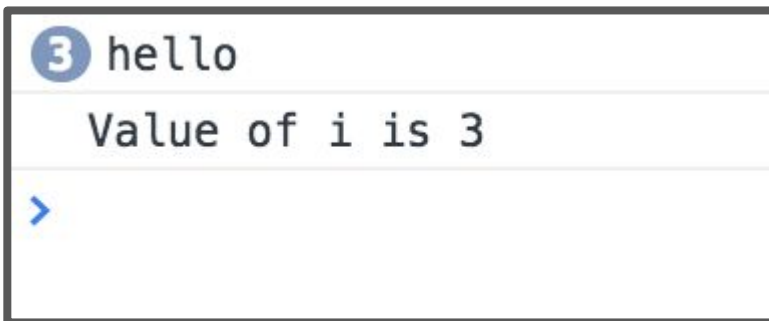
Understanding var

```
function printMessage(message, times) {  
  for (var i = 0; i < times; i++) {  
    console.log(message);  
  }  
  console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Q: What happens if we try to print "i" at the end of the loop?

Understanding var

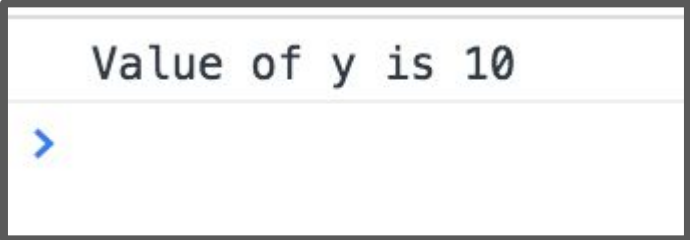
```
function printMessage(message, times) {  
  for (var i = 0; i < times; i++) {  
    console.log(message);  
  }  
  console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```



The value of "i" is readable outside of the for-loop because variables declared with var have function scope.

Function scope with var

```
var x = 10;  
if (x > 0) {  
    var y = 10;  
}  
console.log('Value of y is ' + y);
```



Value of y is 10
>

- Variables declared with "var" have function-level scope and do not go out of scope at the end of blocks; only at the end of functions
- Therefore you can refer to the same variable after the block has ended (e.g. after the loop or if-statement in which they are declared)

Function scope with var

```
function meaningless() {  
  var x = 10;  
  if (x > 0) {  
    var y = 10;  
  }  
  console.log('y is ' + y);  
}  
meaningless();  
console.log('y is ' + y); // error! ❌
```

y is 10

❌ ▶ Uncaught ReferenceError: y is not defined
at script.js:9

But you can't refer to a variable outside of the function in which it's declared.

Understanding **let**

```
function printMessage(message, times) {  
  for (let i = 0; i < times; i++) {  
    console.log(message);  
  }  
  console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Q: What happens if we try to print "i" at the end of the loop?

Understanding **let**

```
function printMessage(message, times) {  
  for (let i = 0; i < times; i++) {  
    console.log(message);  
  }  
  console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

A screenshot of a JavaScript console window. At the top, it shows '3 hello'. Below that, there is a red error message: 'Uncaught ReferenceError: i is not defined'. The error message is followed by the stack trace: 'at printMessage (script.js:5)' and 'at script.js:8'. A blue prompt character '>' is visible at the bottom of the console.

3 hello

✖ ▶ Uncaught ReferenceError: i is not defined
at printMessage (script.js:5)
at script.js:8

>

let has
block-scope so
this results in
an error

Understanding `const`

```
let x = 10;  
if (x > 0) {  
    const y = 10;  
}  
console.log(y); // error!
```



Like `let`, `const` also has block-scope, so accessing the variable outside the block results in an error

Understanding `const`

```
const y = 10;  
y = 0;           // error!  
y++;             // error!  
const list = [1, 2, 3];  
list.push(4);    // OK
```

`const` declared variables cannot be reassigned.

However, it doesn't provide true const correctness, so you can still modify the underlying object

- (In other words, it behaves like Java's `final` keyword and not C++'s `const` keyword)

Contrasting with **let**

```
let y = 10;  
y = 0;           // OK  
y++;             // OK  
let list = [1, 2, 3];  
list.push(4);    // OK
```

let can be reassigned, which is the difference between
const and let

Variables best practices

- Use `const` whenever possible.
- If you need a variable to be reassignable, use `let`.
- **Don't use `var`.**
 - You will see a ton of example code on the internet with `var` since `const` and `let` are relatively new.
 - However, `const` and `let` are [well-supported](#), so there's no reason not to use them.

(This is also what the [Google](#) and [AirBnB](#) JavaScript Style Guides recommend.)

Variables best practices

- Use `const` whenever possible.
- If you need a variable to be reassignable, use `let`.
- **Don't use `var`.**
 - You will see a ton of example code on the internet with `var` since `const` and `let` are relatively new.
 - However, `const` and `let` are **well-supported** so

Aside: The internet has a **ton** of misinformation about JavaScript!

Including several "accepted" StackOverflow answers, tutorials, etc. Lots of stuff online is years out of date.

Tread carefully.