Copyright © 2022 Krystal Institute Limited. All rights reserved.

前端網絡開發人員課程
（二）進階網絡程式設計

8. ES6 JS I: Syntax I

Presented by Krystal Institute

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Learning Objective

- Understand what ES6 is, and how it tackles issues from ES5
- Learn new features on ES6, and how to use them

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Content

8.1
Introduction to
ES6

8.2
ES6 Syntax pt.1

8.3
Default
Parameters

Copyright © 2022 Krystal Institute Limited. All rights reserved.

8.1 Introduction to ES6

Copyright © 2022 Krystal Institute Limited. All rights reserved.

What is ES6?

- Also known as ECMAScript 6 / ECMAScript 2015
- It is a programming language standard (e.g. Jscript, ActionScript)

- ES6 is created to standardize JavaScript
- Makes JS code more modern and readable

Copyright © 2022 Krystal Institute Limited. All rights reserved.

ES6 vs. ES5

- ES5 is the fifth edition of ECMAScript, and ES6 is the next major enchancement to ES5
- There is new operators and methods in ES6 that can greatly reduce the complexity of the codes using ES5
- Throughout your programming time, you will see lots of ES5 codes
- Lots of them isn't updated to ES6

Copyright © 2022 Krystal Institute Limited. All rights reserved.

ES6 usage

- ES6 JS works similar to the JS that you've learned before, but includes new syntaxes, functions and features that makes it much easier to write websites and create complex logic with just a few lines of code
- Some additions of ES6 also improves upon older versions of JS and solves some of the issues.

Copyright © 2022 Krystal Institute Limited. All rights reserved.

8.2 ES6 Syntax pt.1

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let

- Before ES6, var is used for declaring variables

- However, it has many flaws and issues which will be talked about later

```
var variable_name;
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let

- In previous lessons, you have seen let used in codes
- It is a new feature implemented in ES6
- You can use the let keyword to declare a new variable

```
let variable_name;
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- Variables declared using the let keyword is block-scoped
- A block is the space inside a pair of curly braces { }

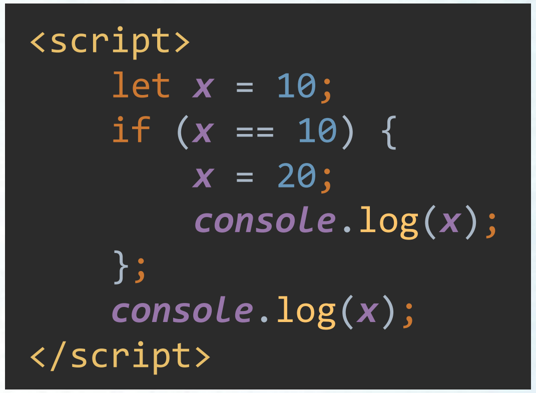
```
<script>
    let x = 10;
    if (x == 10) {
        // This is a block
    };
    // This is outside the block
</script>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

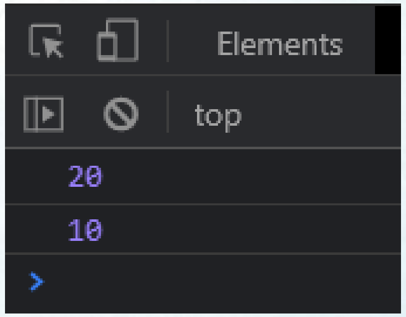
let block-scoping

- In this example on right
- The x inside the block is a new variable, different from the x declared outside of the block

- The x inside the code will be 20, while the x outside the code will be 10



```
<script>
    let x = 10;
    if (x == 10) {
        x = 20;
        console.log(x);
    };
    console.log(x);
</script>
```



```
20
10
>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- In the past, var is used instead for declaring variables

```
var x = 50;
console.log(window.x) // 50
```

- Every variables that are declared using var is added to the property list of the global object – window

```
let x = 50;
console.log(window.x) // undefined
```

- Variables declared using let is not attached to the global object

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

- The example on the right shows a for loop that intends to…
- Output numbers from 0 to 4 every second
- Try it out yourself!

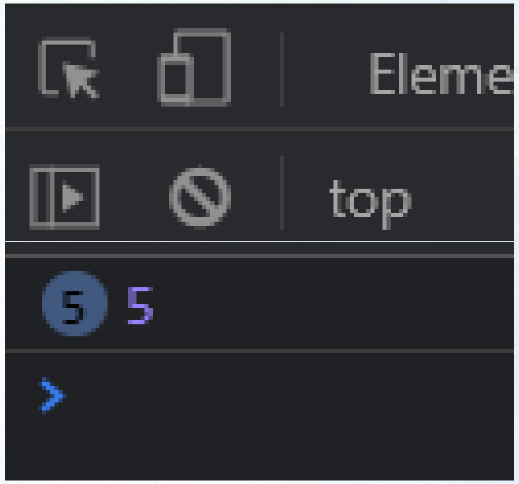
```
for (var i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, 1000);
};
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

- Executing the code will result in…
- Outputting 5 five times
- Why is that?

- To know the answer we need to dive into the asynchronous nature of JS



```
5 5
>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Asynchronous JS

- The JavaScript Engine starts executing your code from the first statement to the last

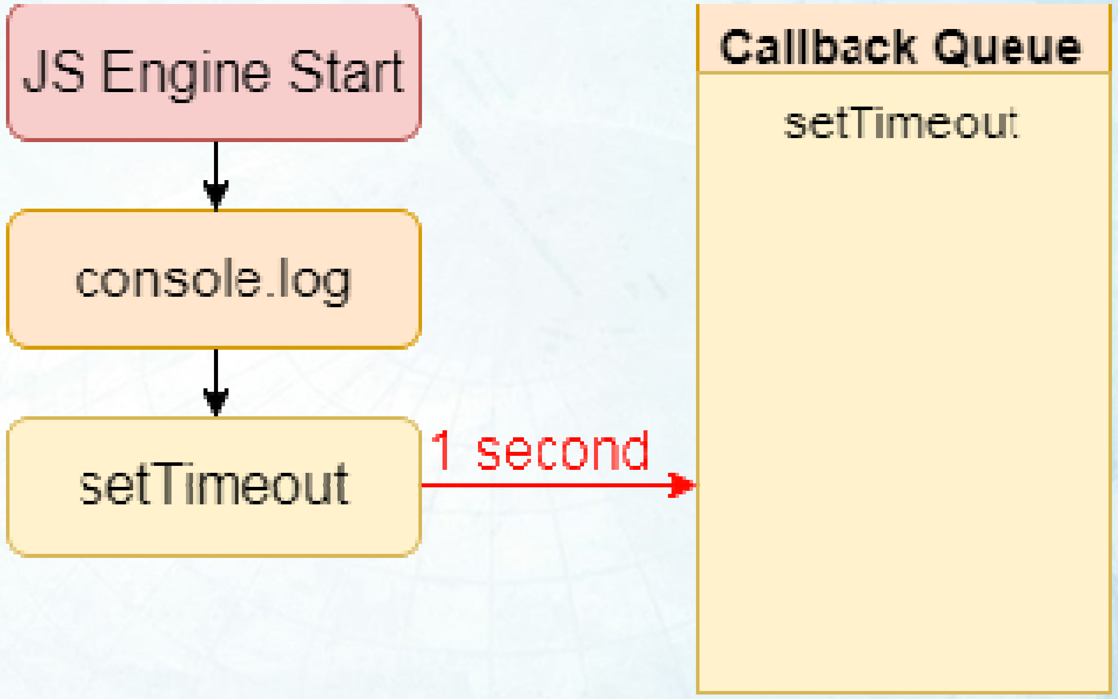
- setTimeout is not a normal function, in HTML, it is used to call a block after a set amount of time (1 second in this case)

```
<script>
    console.log("first statement");

    setTimeout( function sayHi() {
        console.log('Hi')
    }, 1000);

    console.log("last statement");
</script>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Asynchronous JS

- When the JS engine gets to the setTimeout function, it creates a timer that executes immediately, waiting for 1 second
- After that time has passed, the function will be passed to a callback queue

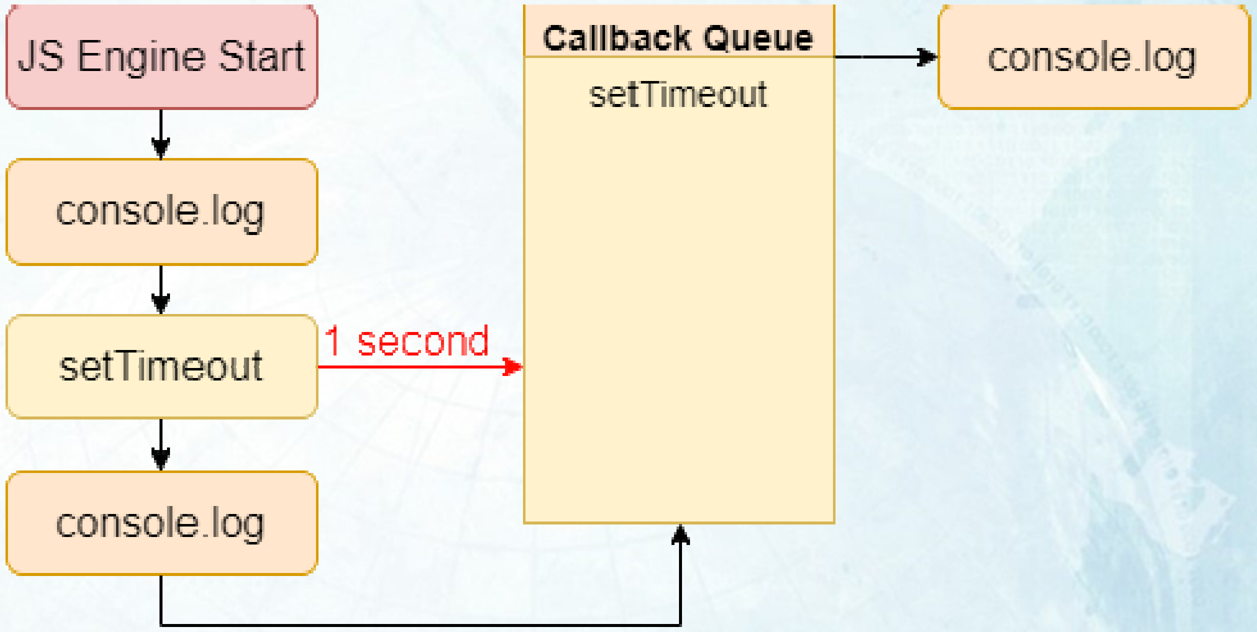


Copyright © 2022 Krystal Institute Limited. All rights reserved.

Asynchronous JS

- After all code in the call stack (lets assume it as codes not inside the setTimeout function for now) was executed

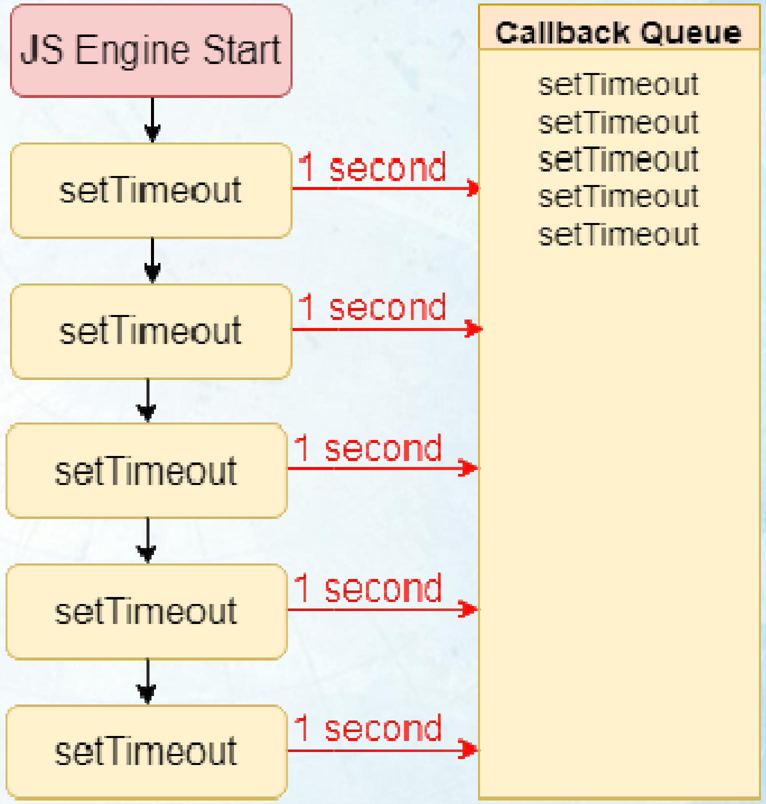
- It starts executing code inside the callback queue, in order



Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

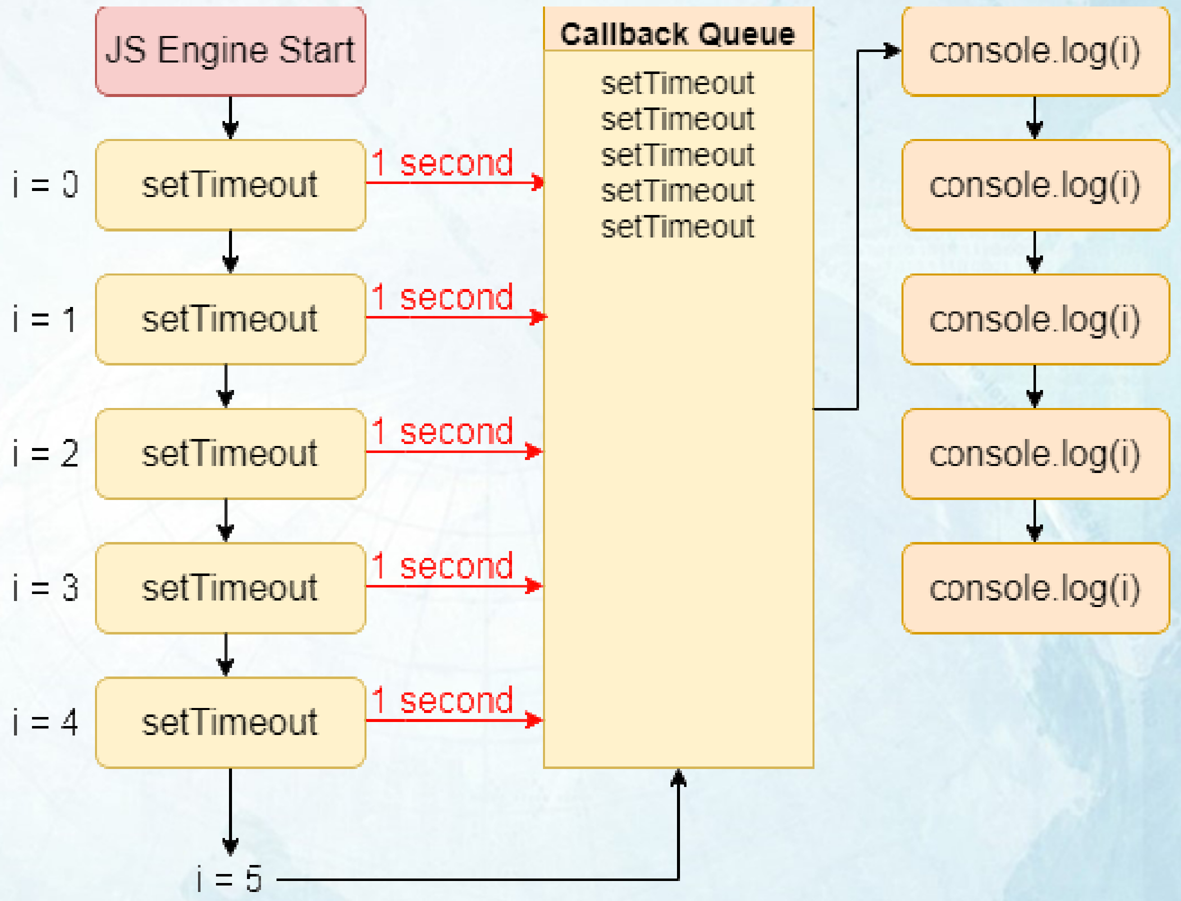
- Back to the for loop example
- In the first loop, setTImeout is called and a timer was created, waiting for 1 second
- Without executing any console.log, i gets added by 1 and the loop continues for 5 times
- Note that all timers happen at the same time as it is asynchronous



Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

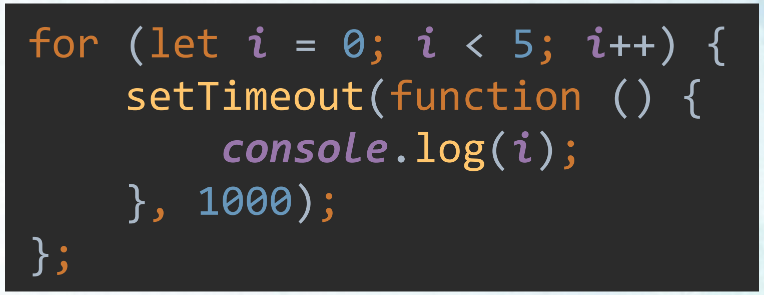
- After all 5 setTimeout finished waiting, the callback queue will be executed in order
- All of them uses i, which is already 5 by that point (I's value is added globally at the end of every loop), hence all of them displaying 5



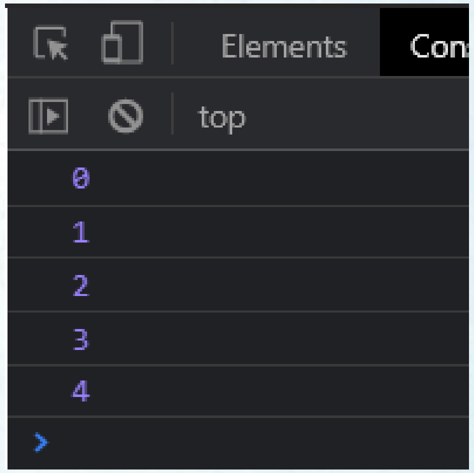
Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

- To fix this issue, simply change the var to let
- let differs from var that let is block scoped, so the same value won't get used everywhere, only inside the block it is declared



```
for (let i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, 1000);
};
```



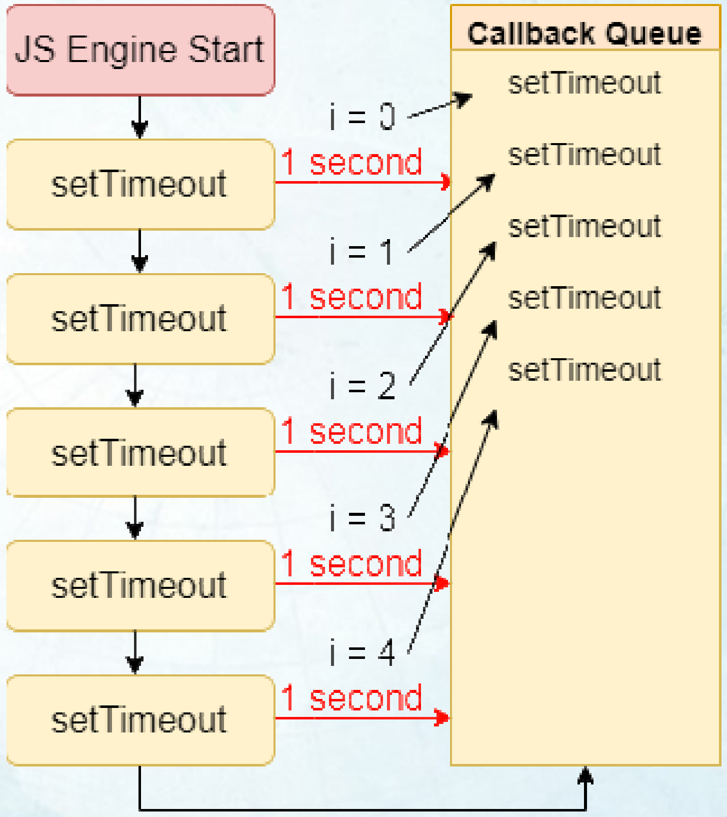
```
0
1
2
3
4
>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

- Using the same flow logic
- In each loop, let declares a new i in each loop, and will only be used in that single loop
- There are 5 loops, so there will be 5 i, each with values from 0 to 4
- When each setTimeout executes, it will use the i variable declared in its loop, creating the desired effect

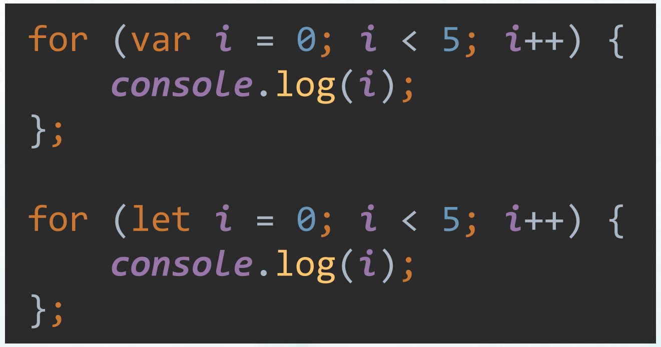
Each i is a different variable



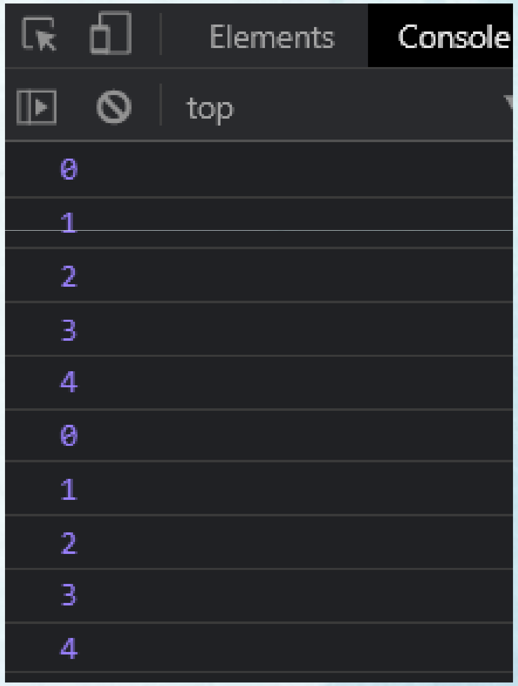
Copyright © 2022 Krystal Institute Limited. All rights reserved.

let in for loop

- Note that this is only an issue with asynchronous functions like setTimeout
- For other functions like console.log , this will not be different using var or let as no functions are put in the callback queue



```
for (var i = 0; i < 5; i++) {
    console.log(i);
};

for (let i = 0; i < 5; i++) {
    console.log(i);
};
```

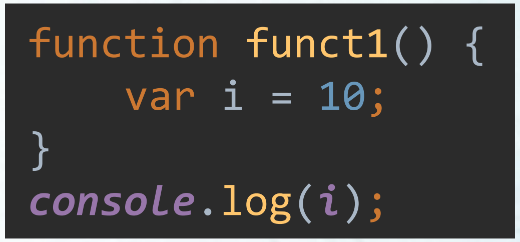


```
0
1
2
3
4
0
1
2
3
4
```

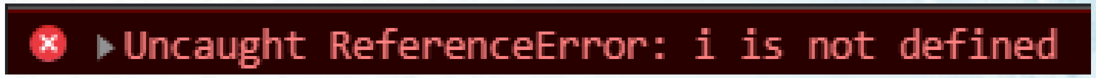
Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- One difference between let and var is the scope
- As mentioned before, variables declared using let is block-scoped
- As for var, they are typically global, except when inside a function
- That means they cannot be referenced outside if they are declared inside a function



```
function funct1() {
    var i = 10;
}
console.log(i);
```

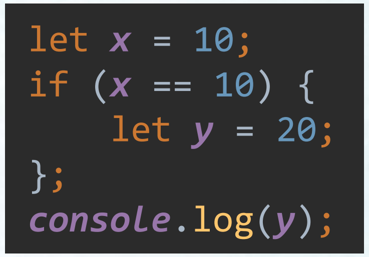


```
▶Uncaught ReferenceError: i is not defined
```

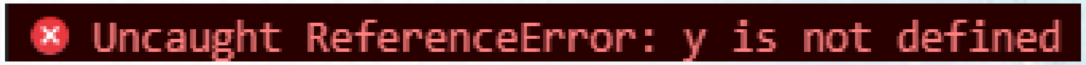
Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- Note that var is only a local variable inside functions, var used in for loops, if else statements and other statements are global as long as its not inside a function



```
let x = 10;
if (x == 10) {
    let y = 20;
};
console.log(y);
```



```
Uncaught ReferenceError: y is not defined
```

- The let keyword is only accessible inside blocks, which is true even for if else statements and for loops as they uses blocks

Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- The var keyword allows you to redeclare a variable
- Redeclaring a variable with let will throw an error
- This might sound like a good thing but…
- Redeclaring does nothing
- It only adds confusion to the code

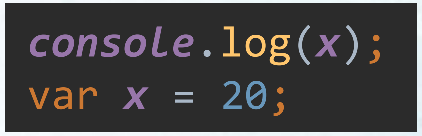
```
var x = 20;
var x;
console.log(x); // 20
```

```
let x = 20;
let x; // error
console.log(x);
```

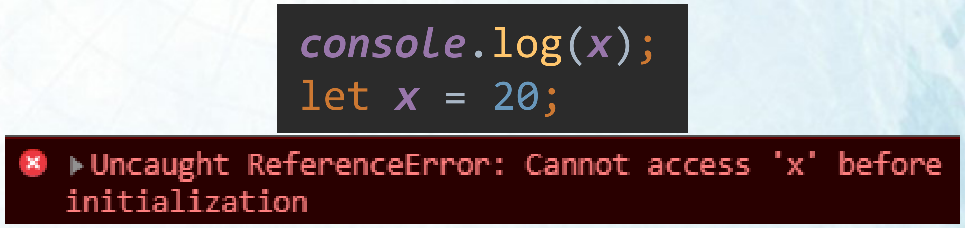
Copyright © 2022 Krystal Institute Limited. All rights reserved.

let vs. var

- Variables declared using var can hoist
- Variables declared using let cannot hoist
- Hoisting is a complex concept, so lets make it simple



```
console.log(x);
var x = 20;
```



```
console.log(x);
let x = 20;
```

```
Uncaught ReferenceError: Cannot access 'x' before
initialization
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Hoisting

- On the example on right, we have a code that console logs a variable before it is declared
- It is possible using var
- Hoisting is a behavior that appears to move declarations
- It doesn't actually happen

```
console.log(x);
var x = 20;
```

Is the same as

```
var x;
console.log(x);
x = 20;
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

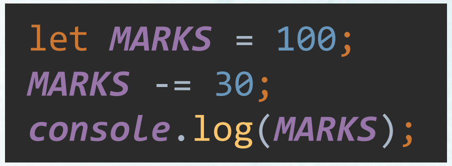
Hoisting

- As you can guess, this is another feature that is seldom useful and often cause more harm than good
- Using let will throw a ReferenceError
- var or not, it is still best to declare the variable first before using it to avoid confusion and potential errors

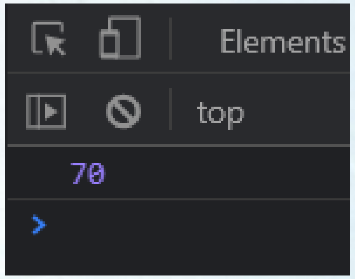
Copyright © 2022 Krystal Institute Limited. All rights reserved.

Constants

- The const keyword works like the let keyword
- It declares a block-scoped variable
- The variable is read-only, meaning you cannot reassign it
- In variables declared by the let keyword, you can change and reassign values to it



```
let MARKS = 100;
MARKS -= 30;
console.log(MARKS);
```

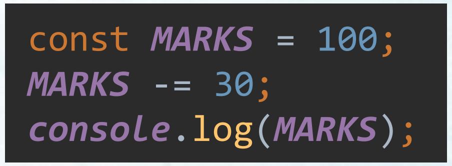


```
70
>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

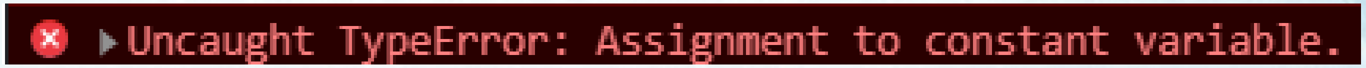
Constants

- In variables declared by the const keyword, they are immutable



```
const MARKS = 100;
MARKS -= 30;
console.log(MARKS);
```

- Trying to change or reassign a const variable will result in a TypeError

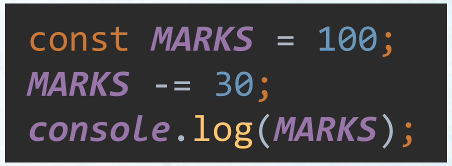


```
Uncaught TypeError: Assignment to constant variable.
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

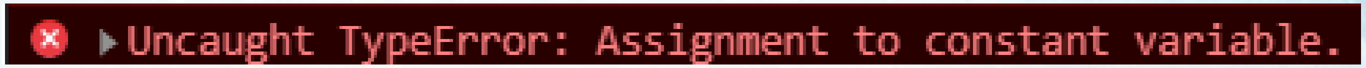
Constants

- The const keyword must also be used and assigned a value



```
const MARKS = 100;
MARKS -= 30;
console.log(MARKS);
```

- Declaring it without a value will result in SyntaxError

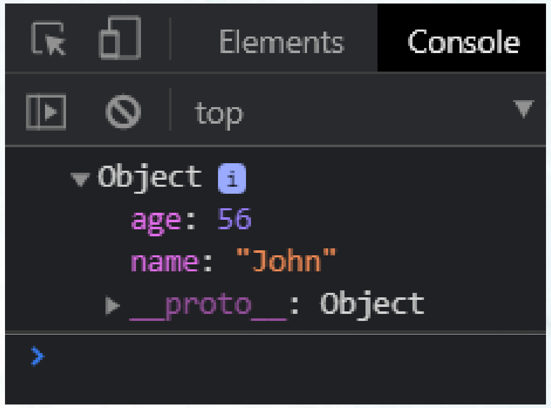


```
Uncaught TypeError: Assignment to constant variable.
```

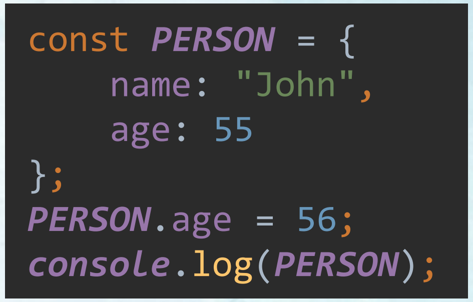
Copyright © 2022 Krystal Institute Limited. All rights reserved.

Constants & Objects

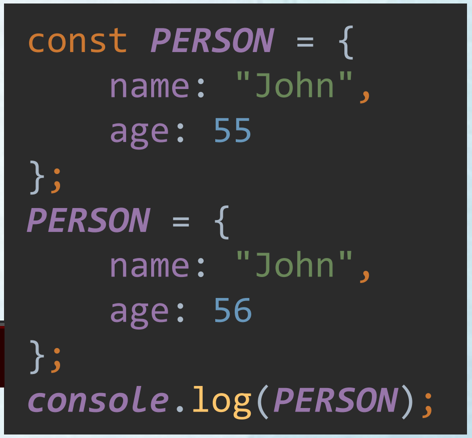
- An object declared as a variable is special
- Changing an object's property values is allowed, as it does not change the variable itself
- However, reassigning the object with another is still not allowed



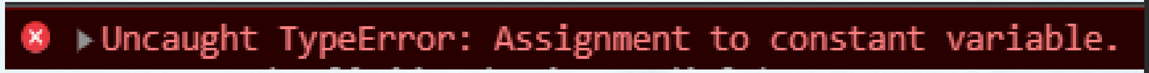
```
▼Object i
    age: 56
    name: "John"
  ▶__proto__: Object
>
```



```
const PERSON = {
    name: "John",
    age: 55
};
PERSON.age = 56;
console.log(PERSON);
```



```
const PERSON = {
    name: "John",
    age: 55
};
PERSON = {
    name: "John",
    age: 56
};
console.log(PERSON);
```



```
▶Uncaught TypeError: Assignment to constant variable.
```

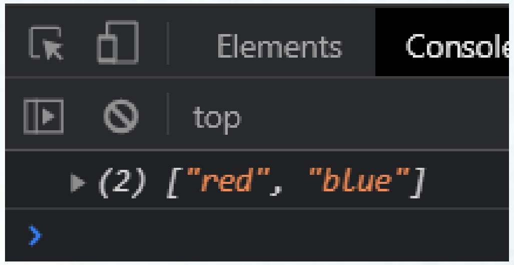
Copyright © 2022 Krystal Institute Limited. All rights reserved.

Constants & Arrays

- If an array was assigned as a constant

- Array methods like push and pop still functions

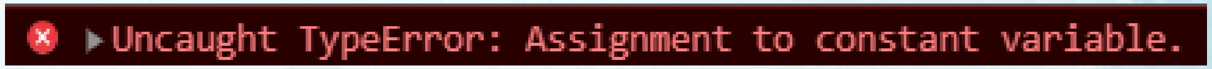
- Reassigning it will again result in TypeError



```
▶ (2) ["red", "blue"]
```

```
const COLORS = ["red"];
COLORS.push("blue");
console.log(COLORS);
```

```
COLORS = []
console.log(COLORS);
```



```
▶Uncaught TypeError: Assignment to constant variable.
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Constant widespread practice

- In previous examples, you see constant variables with all upper cases
- Although it is not necessary, it is a good practise to use uppercase and underscores to declare variables
- It makes it much easier to read and differ from variables made using the let keyword

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Constant widespread practice

- In this scenario, you can assign color hexcodes to constants (since they will not change)
- It will make it easier to type and read when you need it

```
const COLOR_RED = "#E72019";
const COLOR_GREEN = "#05D731";
const COLOR_BLUE = "#0E31E2";
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

8.3 Default Parameters

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Parameters & Arguments

- We often use parameters and argument interchangeably

- On the example on right,
- The add function has 2 parameters – x and y
- Calling an add function requires 2 arguments – 10 and 5

```
function add(x, y) {
    console.log(x + y)
};

add(10, 5);
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Parameters & Arguments

- The default value of a parameter is undefined
- If you don't provide the function with arguments, it will cause it to be undefined, rending the whole function useless

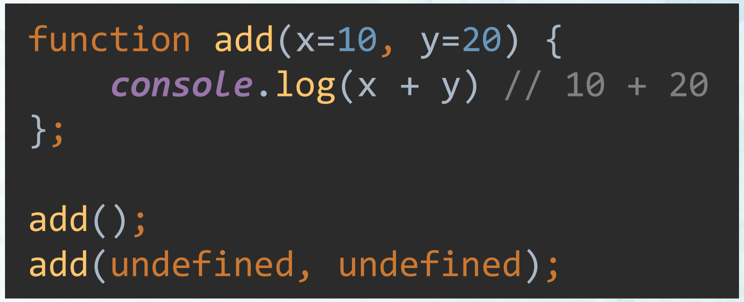
```
function add(x, y) {
    console.log(x + y) // undefined + undefined
};

add();
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

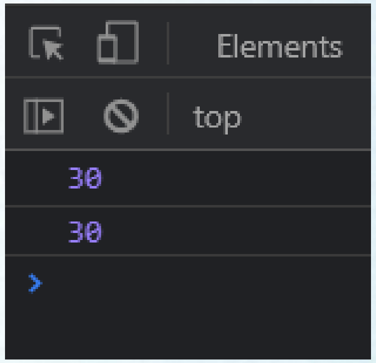
Default Parameter

- We can change the default value of a parameter so even without any arguments, the function will still function!

- To do that, assign a value to the parameter inside the function



```
function add(x=10, y=20) {
    console.log(x + y) // 10 + 20
};

add();
add(undefined, undefined);
```



```
30
30
>
```

Copyright © 2022 Krystal Institute Limited. All rights reserved.

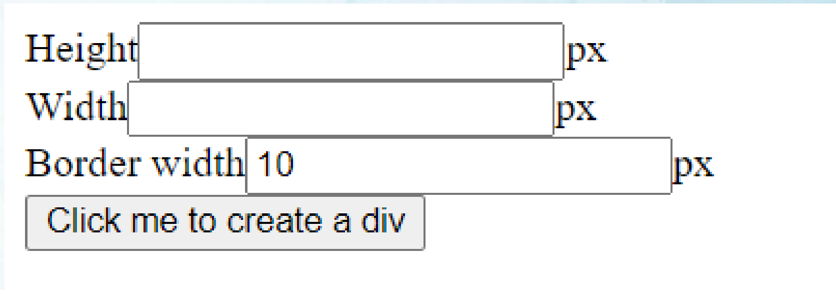
Default Parameter Exercise

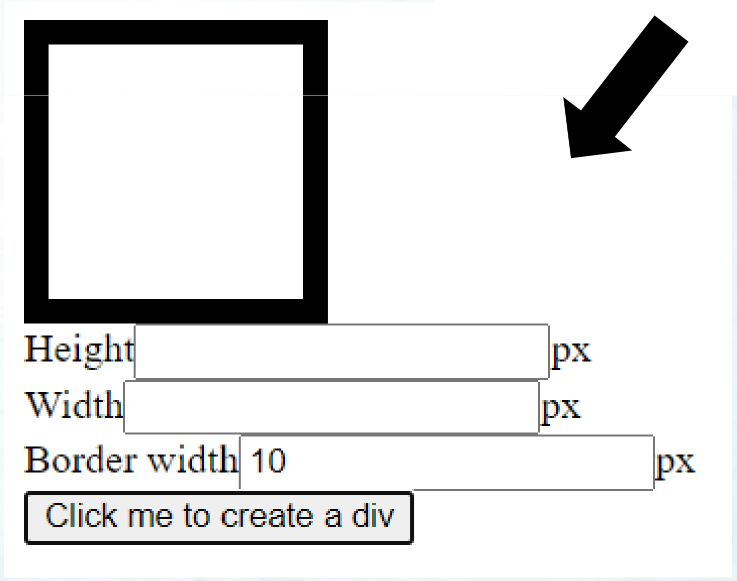
- Create a website that…
- Has a button that calls a function on click, and 3 inputs for height, width, and border width
- If there is no inputs in one input element, it should be set to undefined (value is always a string, assign it to a variable)
- The function will create a new div at the very top of <body>, based on the inputs, and has 3 parameters:
  - height: height of the div, defaults to 100px
  - width: width of the div, defaults to 100px
  - Border-width: width of the border, defaults to 3px

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Default Parameter Exercise Example

- Given 3 input boxes and a button, if any of the inputs are typed with valid numbers (e.g. typing 10 in border width) and you click on the button
- It will show the created div with border width as 10px

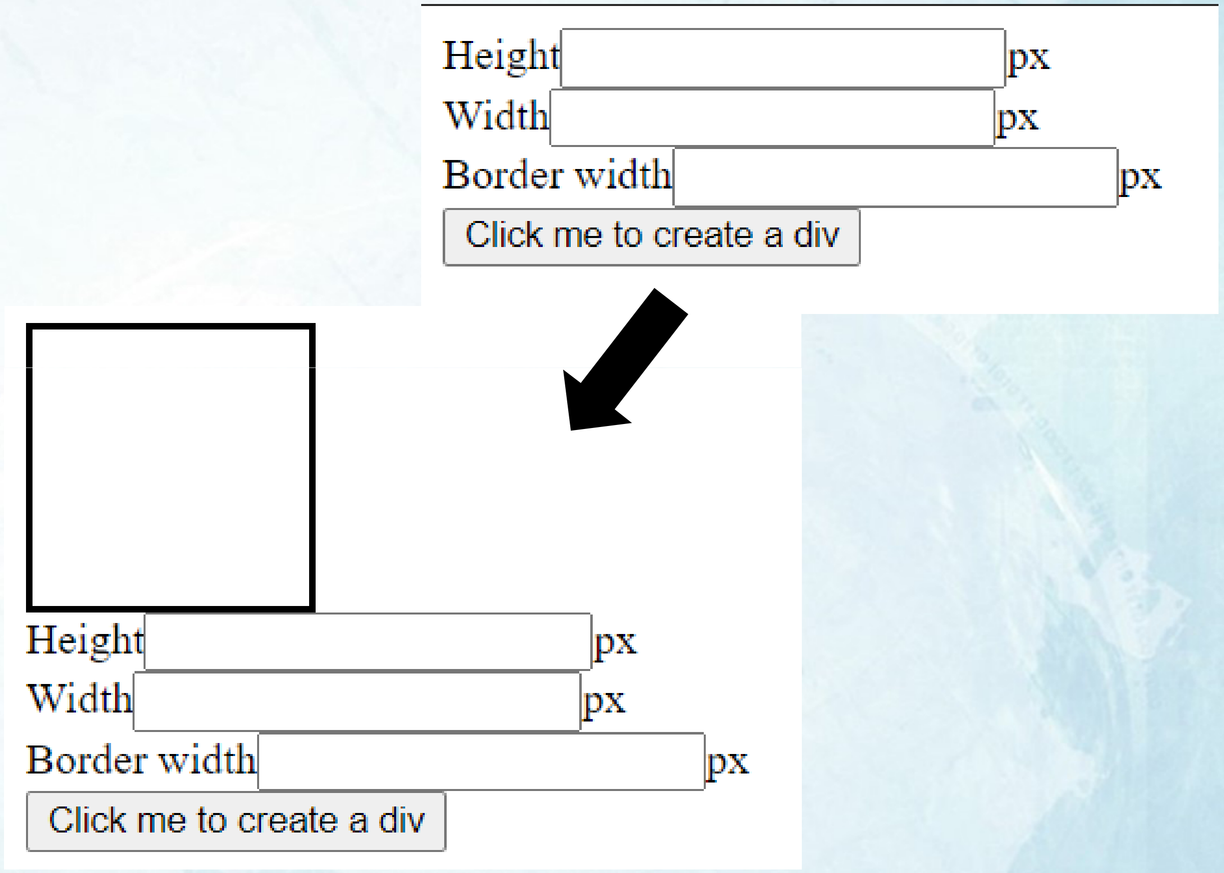

Height px
Width px
Border width 10 px


Height px
Width px
Border width 10 px

Copyright © 2022 Krystal Institute Limited. All rights reserved.

Default Parameter Exercise Example

- If nothing is inputted and the button was clicked, the div should have 100px height, 100px width and 3px border width



Copyright © 2022 Krystal Institute Limited. All rights reserved.

References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/