
SOFT

CICSOFT

WHAT'S CICSOFT?

CICSoft is an interest-oriented student organization fostering a collaborative environment for members to enhance their technical skills while developing applications from scratch.

Our aim is to empower students by delivering the means to build incredible apps and realize their ideas through code.

ENVIRONMENT
Installations

Install VSCode

VSCode Extension : Live Server

VSCode Extension : HTML CSS

CICSOFT

Do you remember?

CICSOFT

JAVASCRIPT TRILOGY

JavaScript : The Rise

The history behind JS, and how it came of being the language we all love today.

JavaScript : The Reign

Learning and Developing with JS, an in-depth approach to the programming language.

JavaScript : Forever

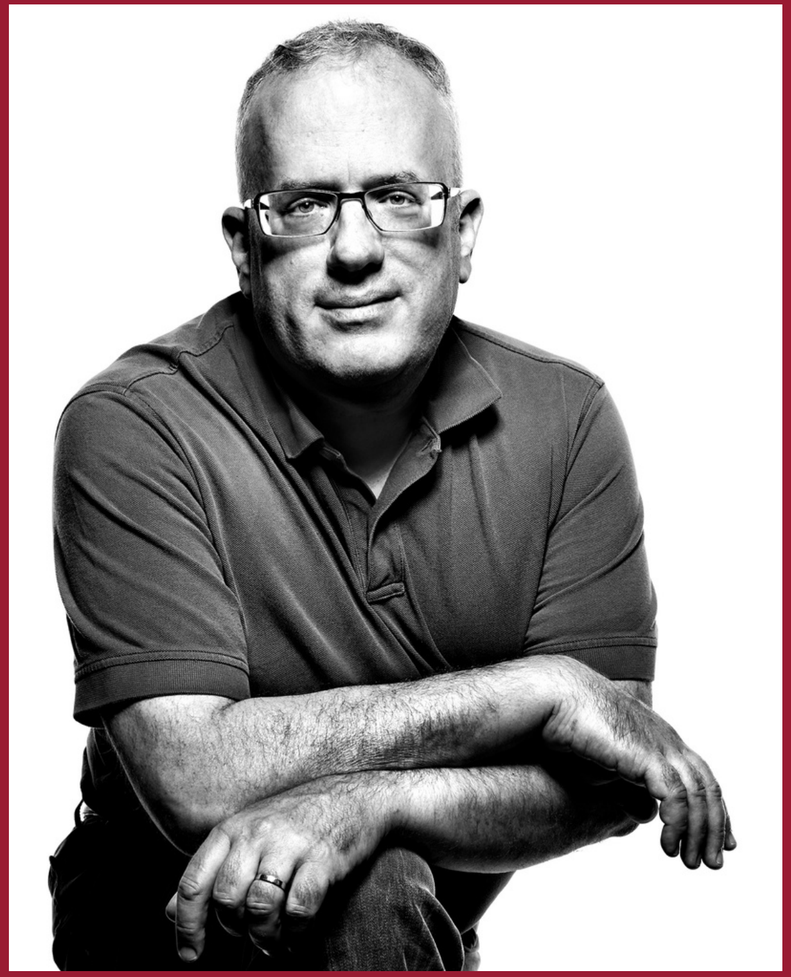
A take home activity to practice JS concepts.


JS
THE RISE

BRIEF HISTORY

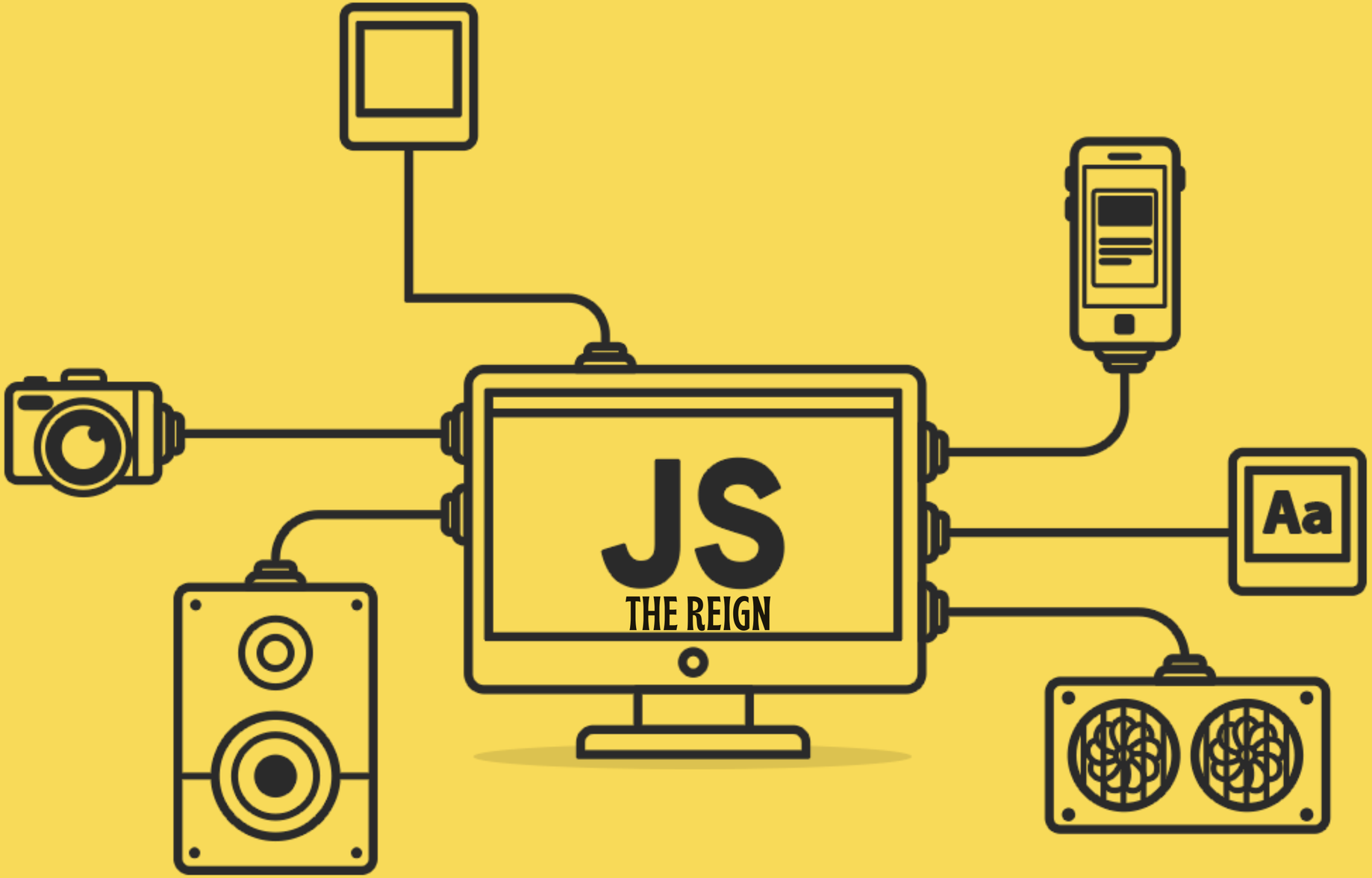
CICSOFT

HISTORY

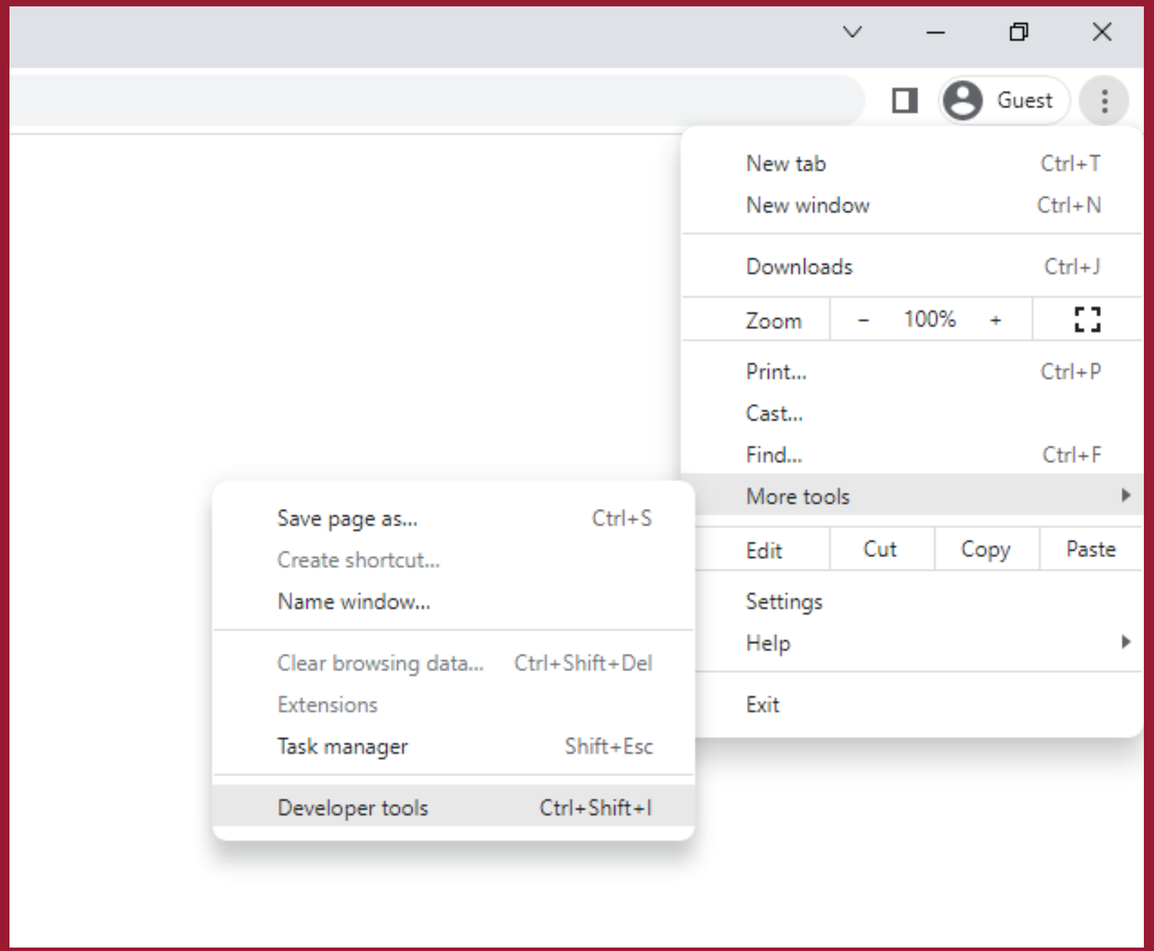


- Created by Brendan Eich.
- Created in 1 week.
- (Still has broken Logic)

- Belongs to C Family of Languages


JS
THE REIGN
Aa

ENVIRONMENT



VSCODE

index.html > ...

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <script src="index.js"></script>
5       </head>
6       <body>
7
8       </body>
9   </html>
```


PRIMITIVES
Animals?

CICSOFT

PRIMITIVES

What are the types?

- numbers - 420, 6.9
- strings - 'Hello'
- boolean - true/false
- undefined
- null


NUMBERS
Look Ma! I can count

NUMBERS

- 37 // Evals to 37
- 42.0 // Evals to 42
- 13.5 // Evals to 13.5
- 3.0 === 3 // Evals to True

CICSOFT

NUMBERS

Operations

- 37 + 5 // Evaluates to 42
- 20 - 3 // Evaluates to 17
- 21*20 // Evaluates to 420
- 14/2 // Evaluates to 7
- 15/2 // Evaluates to 7.5

NUMBERS

Not A Number (NaN)

What does 10 / 0 evaluate to?

NUMBERS

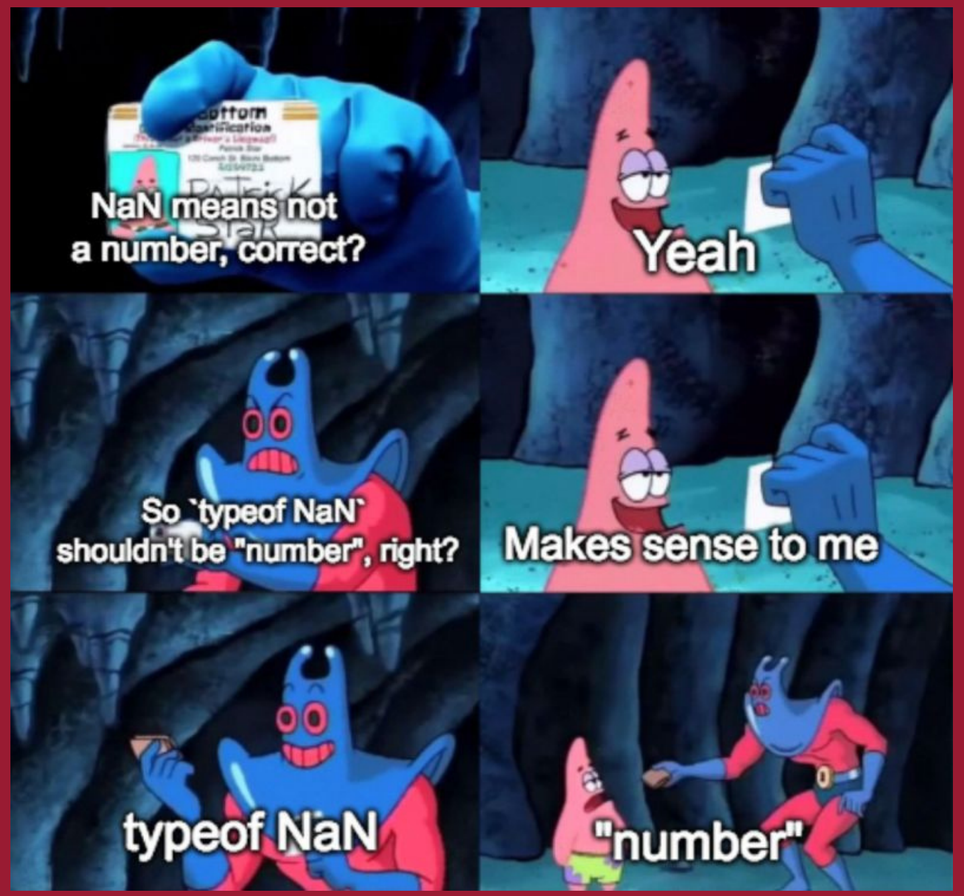
Not A Number (NaN)

What does 10 / 0 evaluate to?

What does Infinity * 0?

NUMBERS

Not A Number (NaN)




STRINGS
Mozart making an appearance?

STRINGS

- "This is a string"
- 'This is also a string'
- `Strings can have numbers too`
- '23' // is not the same as 23

STRINGS

Operations

- 'concat' + 'enation' // 'concatenation'
- 'string' + '1' // 'string1'
- 'string'.toUpperCase()// 'STRING'
- 'string'[0] // 's'
- String(2+2) // '4'

STRING

Concatenation

What do the following evaluate to?

- 'string' + '1'
- 'string' + 1
- 'string' + 1 + 2
- 1 + 2 + 'string'

STRING

Concatenation

- 'string' + '1' // 'string1'

STRING

Concatenation

- 'string' + '1' // 'string1'
- 'string' + 1 // 'string1'

STRING

Concatenation

- 'string' + '1' // 'string1'
- 'string' + 1 // 'string1'
- 'string' + 1 + 2 // 'string12'

STRING

Concatenation

- 'string' + '1' // 'string1'
- 'string' + 1 // 'string1'
- 'string' + 1 + 2 // 'string12'
- 1 + 2 + 'string' // '3string'


BOOLEANS
Like Scary Ghosts?

BOOLEANS

Literally of two types

BOOLEANS

Literally of two types

1. True

BOOLEANS

Literally of two types

1. True
2. False

BOOLEANS

Literally of two types

1. True
2. False

Literally.

BOOLEANS

Operations

1. Or ||
   - (hungry || thirsty) then, take a break
2. And &&
   - (tired && late) then, sleep
3. Not !
   - !sleepy then, watch youtube

BOOLEANS

Comparisions - PROBLEM TIME!

- 3>2
- true && (1>5)
- true || (1>5)
- (true || 1) > 5

T: True | F: False | N:Niether

CICSOFT

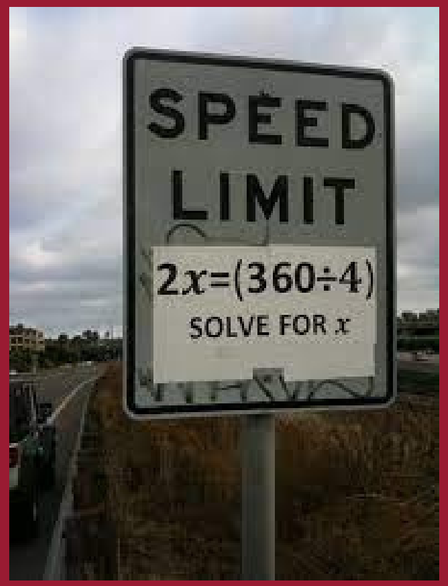
NULL

- and undefined

UNDEFINED AND NULL

VARIABLES

Who is Mr X?



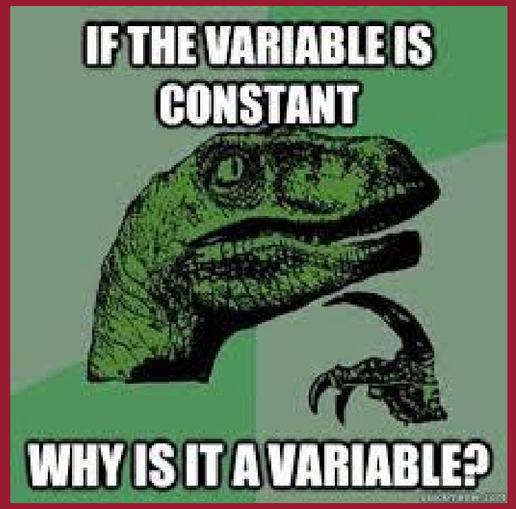
VARIABLES

const	(If the variable doesn't change)
let	(If the variable changes)
var	(Deprecated)
~~Undefined Global~~	(Just dont!)

CONST

const a = 10;

When you use const, you can't reassign the variable.



CICSOFT

LET

let a = 10;

When you use let, you can reassign
the variable.

Let is also block-scoped
(We will talk about what this means when we discuss functions)

VAR

var a = 10;

When you use var, you can also reassign the variable.

var is not block-scoped
(again, functions section)

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const	?	?	?
let	?	?	?
var	?	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const		?	?
let	?	?	?
var	?	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const			?
let	?	?	?
var	?	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const			?
let		?	?
var	?	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const			?
let			?
var	?	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const	✗		?
let	✓		?
var	✓	?	?

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const	✗		?
let	✓		?
var	✓		?

CICSOFT

WILL THEY WORK?

```
1. const a = 10;
2.     let b = 12;
3.  var c = a + b;
4.       b = b+2;
5.       c=b*3;
6.         a=a;
7.         let x;
8. console.log(x);
```


CONSOLE
Gaming!

CONSOLE

console.log()

- Most handy tool

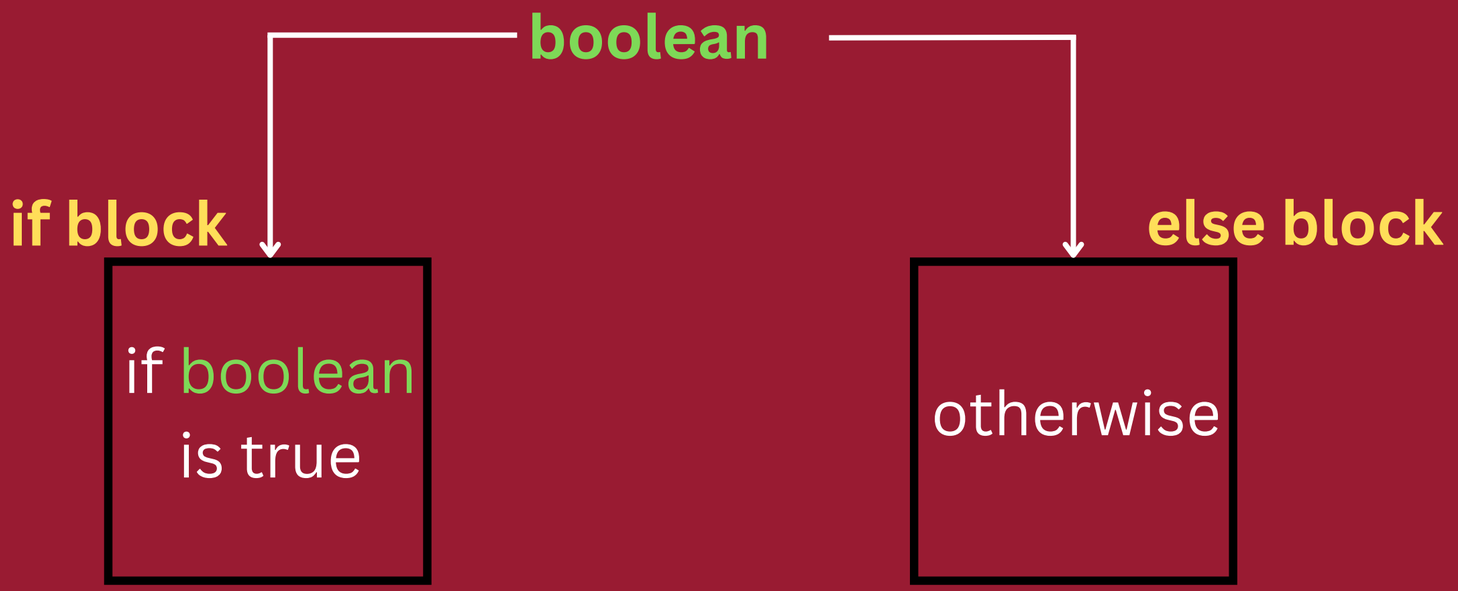
console.assert()

- Used for testing


This or That?

CONTROL FLOW

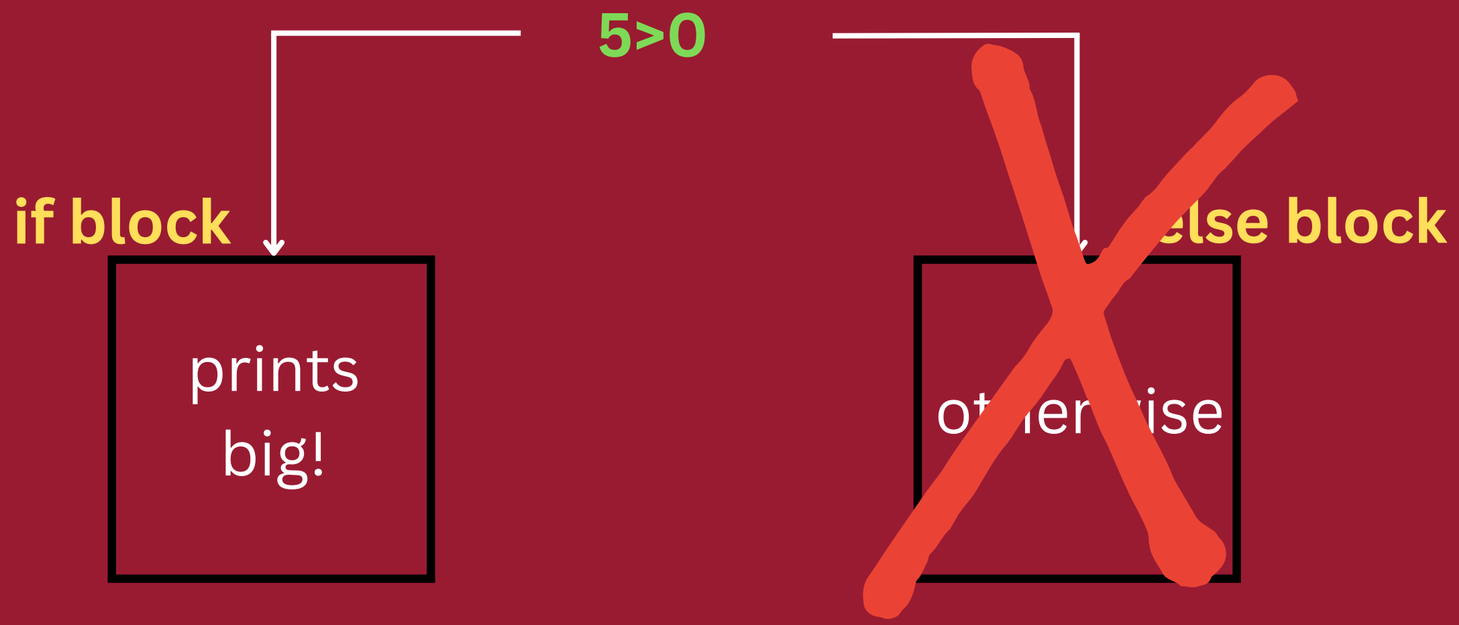
if and else statents



CONTROL FLOW

if and else statents

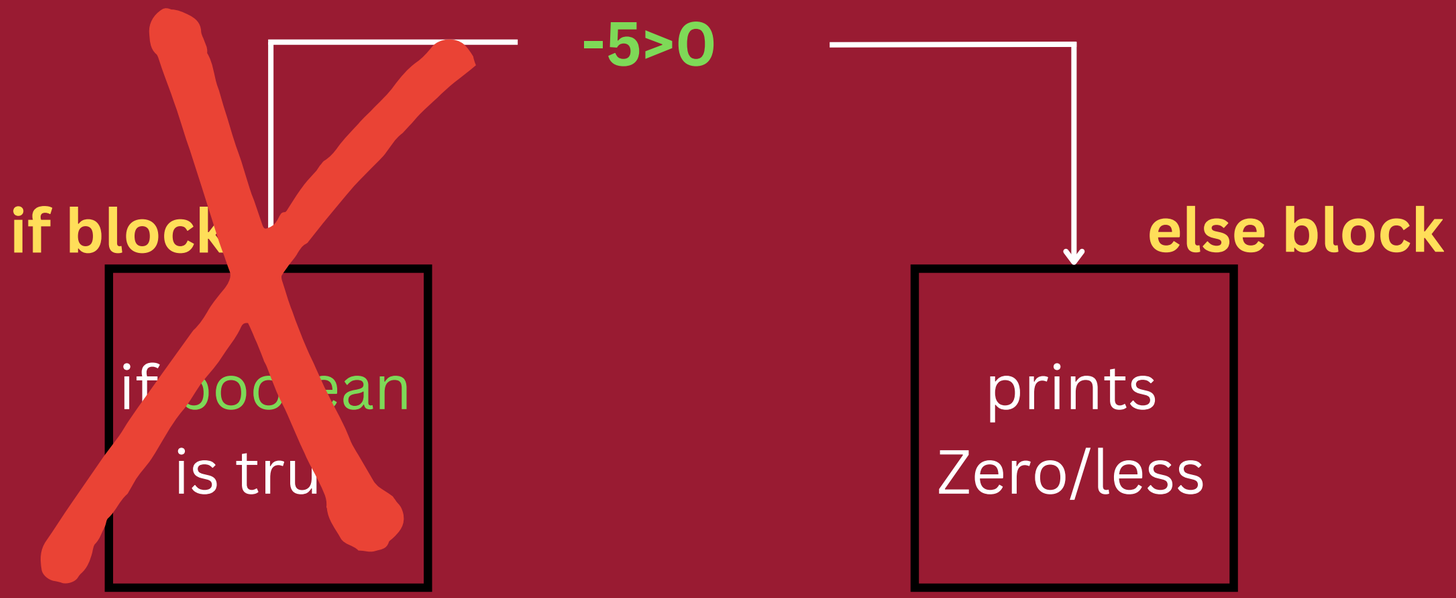
```
if(5>0){
        console.log('big!');
}
else{
        console.log('Zero/Less');
}
```



CONTROL FLOW

if and else statents

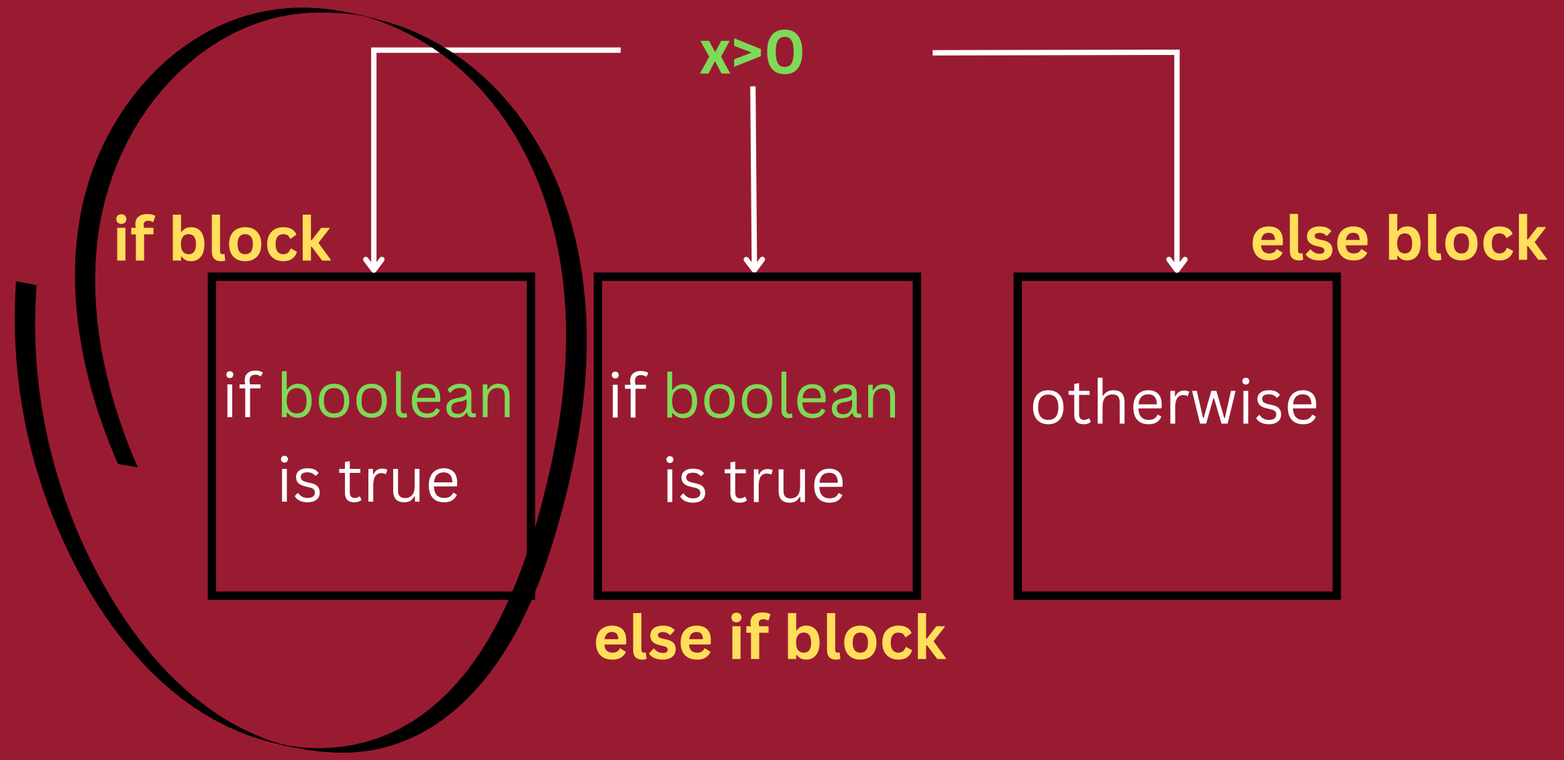
```
if(-5>0){
        console.log('big!');
}
else{
        console.log('Zero/Less');
}
```



CONTROL FLOW

else if statments

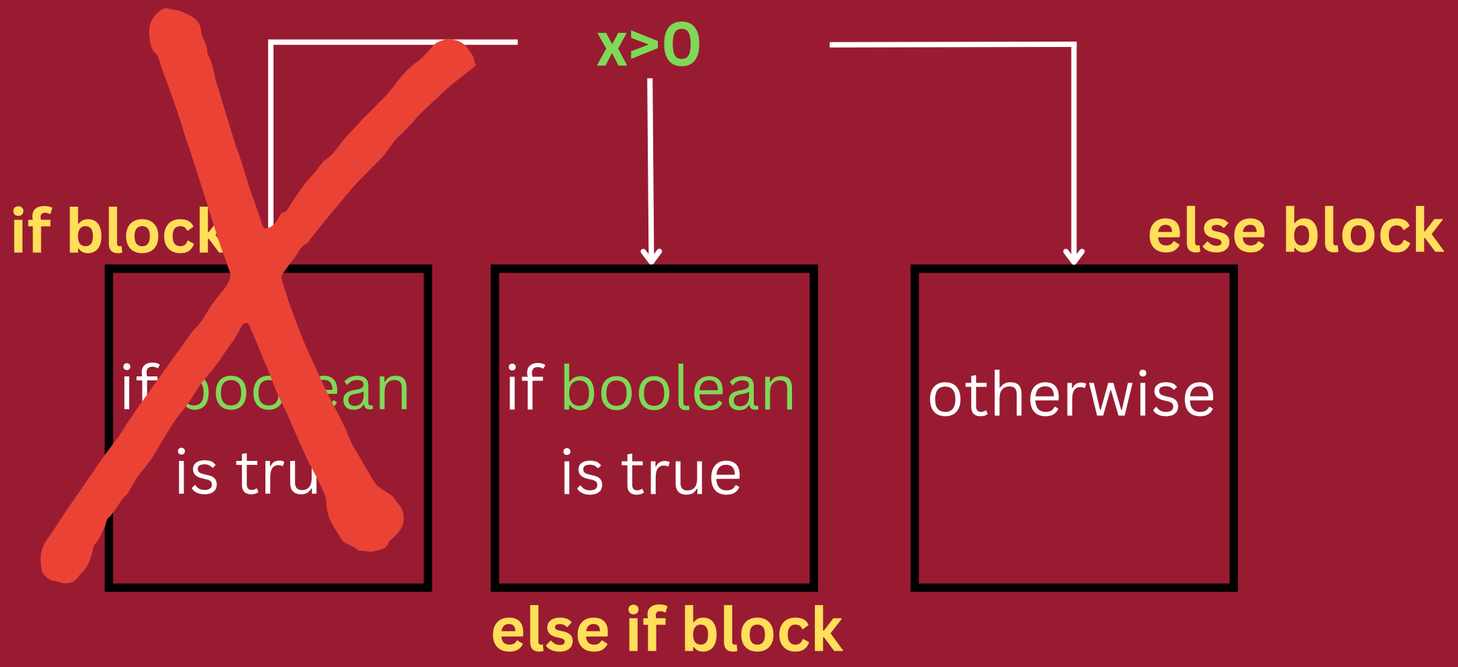
```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```



CONTROL FLOW

else if statments

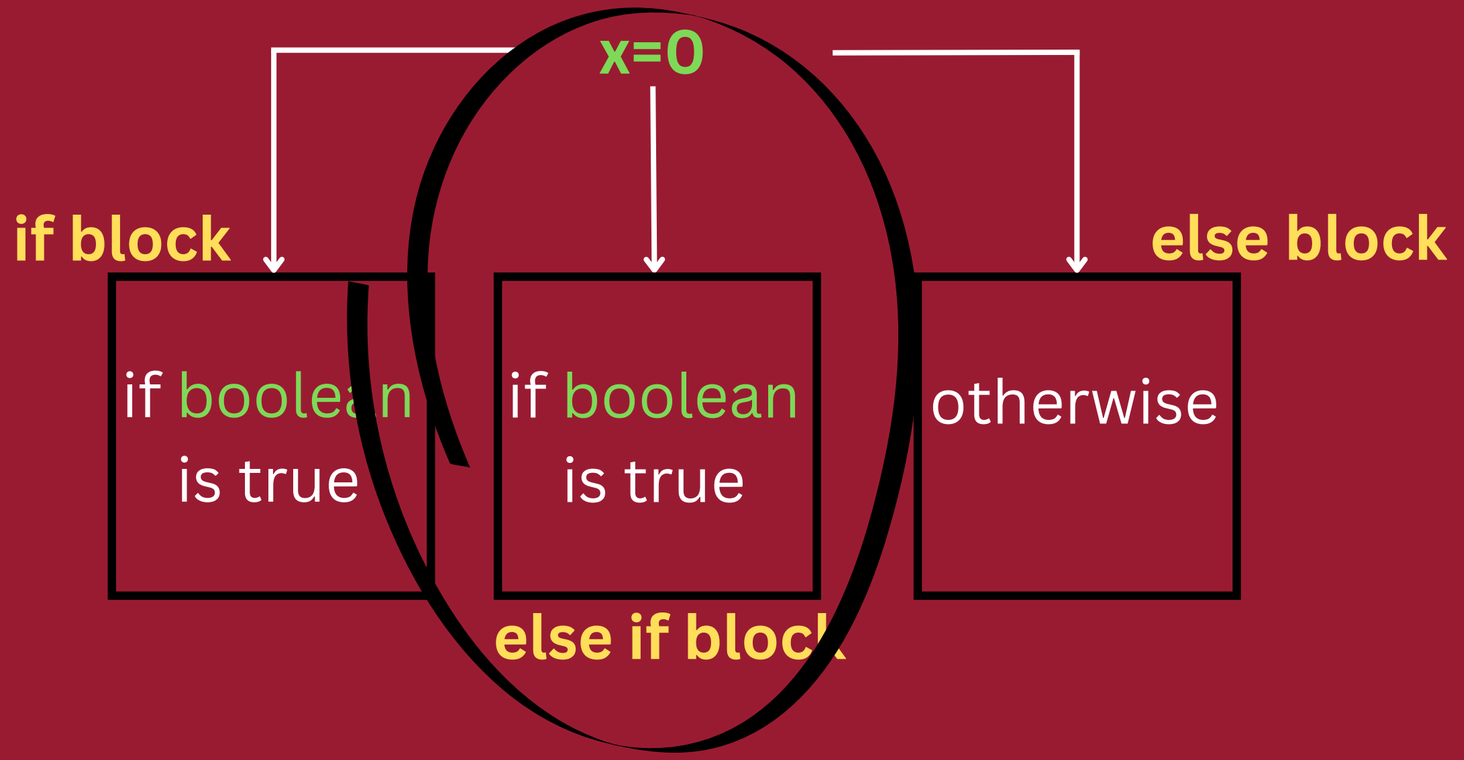
```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```



CONTROL FLOW

else if statments

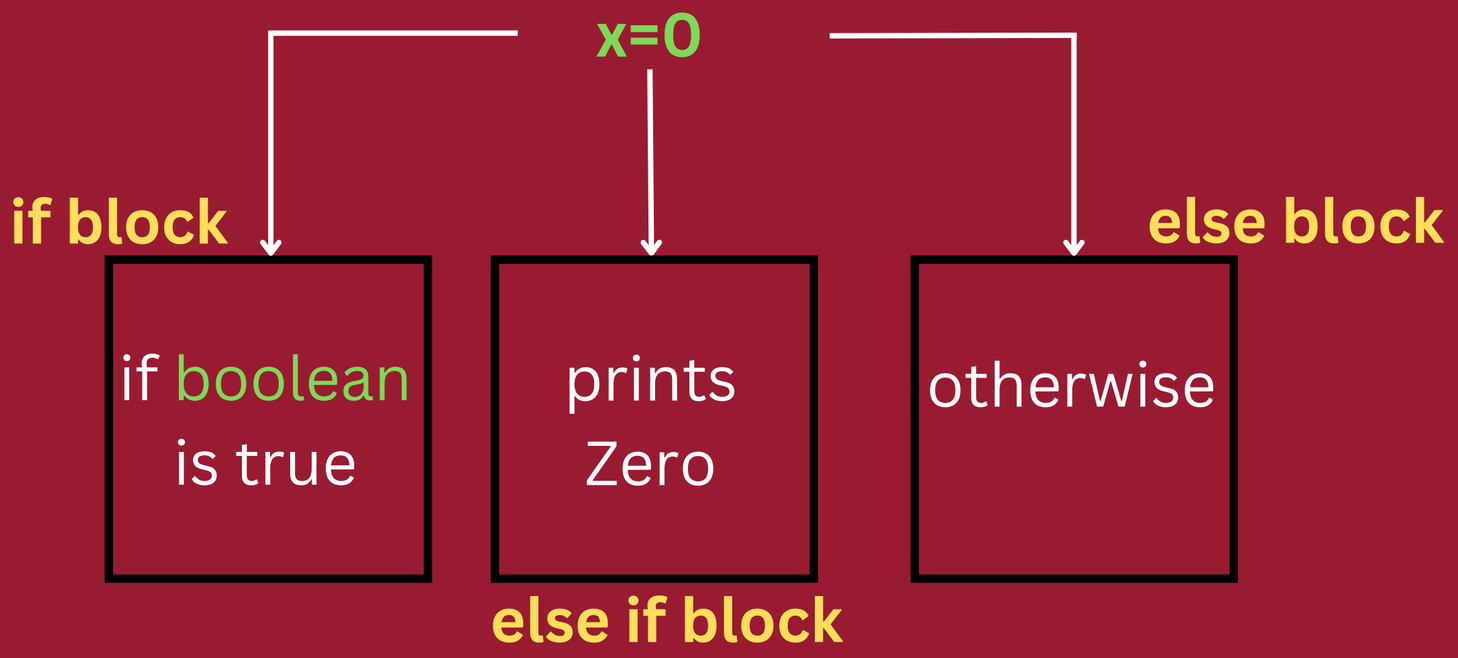
```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```



CONTROL FLOW

else if statments

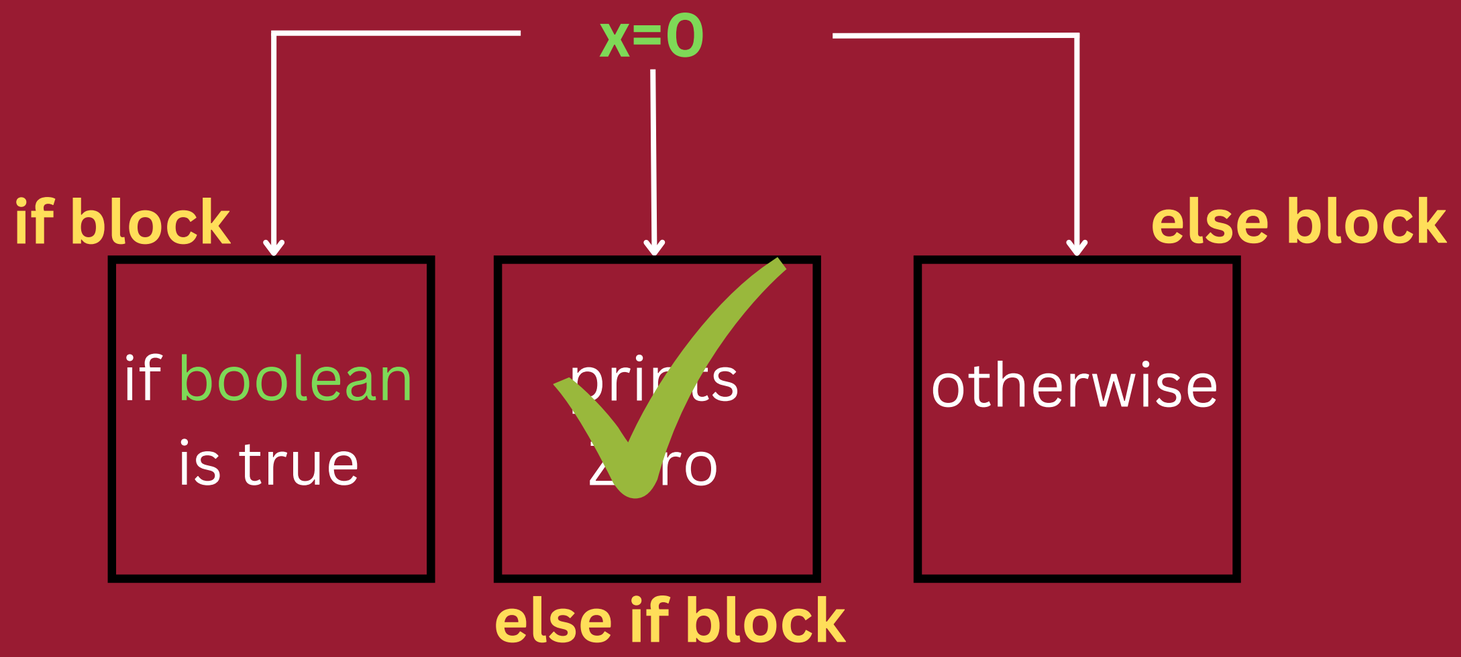
```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```



CONTROL FLOW

else if statments

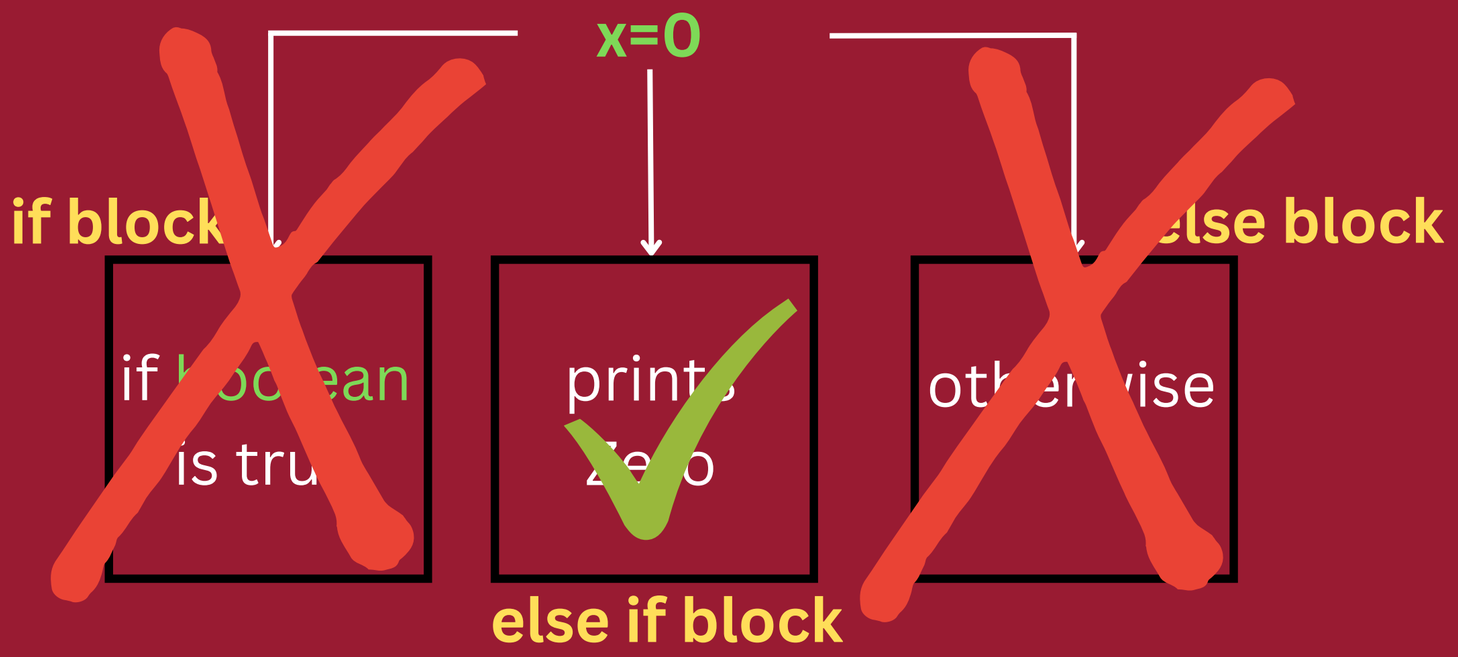
```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```



CONTROL FLOW

else if statments

```
let x = 0
if(x>0){
        console.log('big!');
}
else if (x==0){
        print('Zero');
}
else{
        console.log('Less');
}
```




ARRAYS
The light from the sun?

ARRAYS

Container for a series of things

```
const numArr = [10,20,30,40]

const strArr = ['hello', 'from', 'CICSoft']

const weirdArray = [1, 'hello', '3.0']
```

CICSOFT

ARRAYS

Operations

const Array = [10,20,30]

```
Array.length()     // returns the length : 3
Array.push(40)     // Adds 40 at the end
Array.pop()        // deletes and returns the last element
Array.shift()      // deletes and returns the first element

Array.splice()     // Read up. It can do most operations

Array[i]           // returns the ith element of the array (starting 0)
```


LOOPS
Edge of Tomorrow S***?

CICSOFT

LOOPS

Why Need loops

```
const Array = [10,20,30,40]
```

Find the sum of the Array.

```
const names = ['Anshul', 'Kirat', 'AVS']
```

Find the longest name.

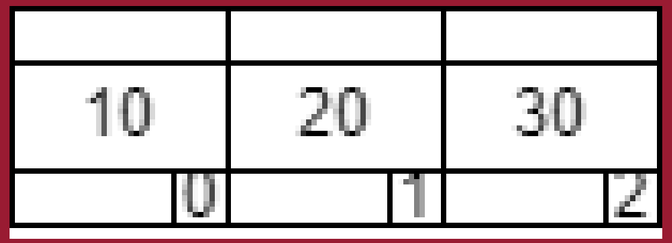
LOOPS

Why Need loops

Loops happens for repeated tasks

const array = [10,20,30]

```
let sum = 0;
for (let i=0; i<array.length(); i++) {
  sum = (sum + array[i]);
}
```



LOOPS

Activity time!

Print 'JS is fun!' 10 times onto the console

OBJECTS

The best part about JS

OBJECTS

They're a structure

```
const obj = {
  name: 'Xeze's hp laptop',
  age: 9,
  model: 'HP',
  color: 'silver'
}
```

OBJECTS

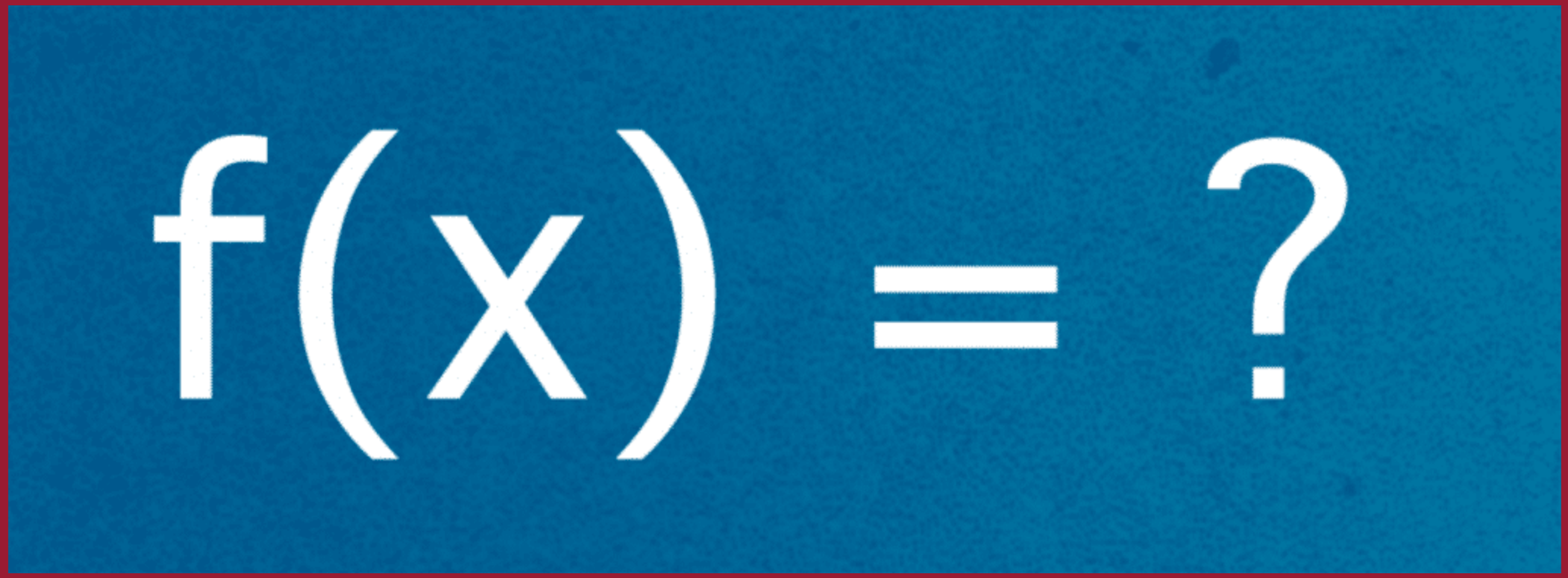
Operations

```
const obj = {
  name: 'Xeze's hp laptop',
  age: 9,
  model: 'HP',
  color: 'silver'
}

obj.name                    // returns the name
obj["name"]                 // returns the name

const field = "name"
obj[field]                  // evaluates to obj["name"]
```

OBJECTS

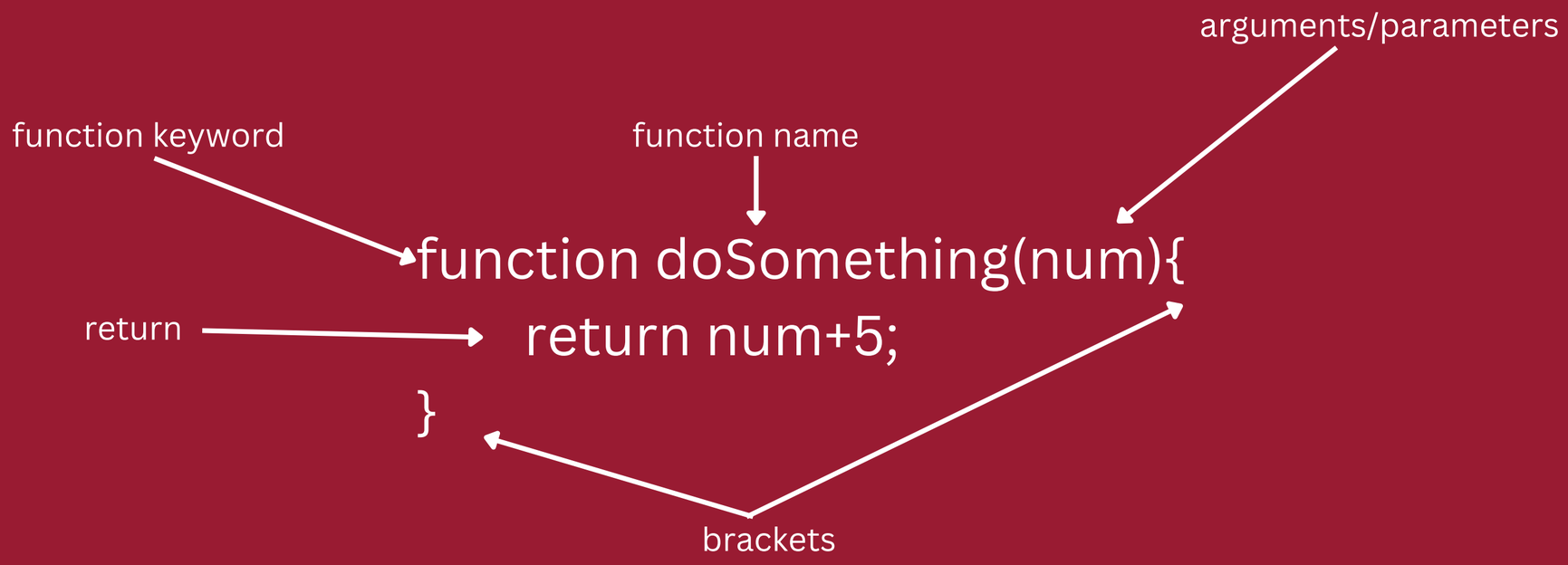
Operations

```
const obj = {
  name: 'Xeze's hp laptop',
  age: 9,
  model: 'HP',
  color: 'silver'
}

obj.weight = 2.6          // adds a new field
obj['weight'] = 2.6       // adds a new field

delete obj.weight         // deletes the field
delete obj['weight']      // deletes the field
```


f(x) = ?

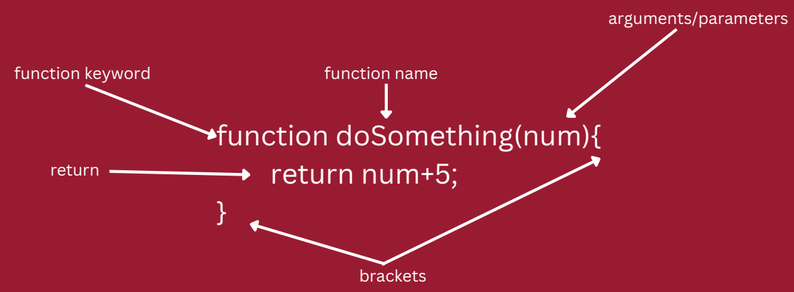
FUNCTIONS

Block of code that does something, when it is called.



DEFINITIONS

Function Keyword: tells JS that we are defining a function



Function Name: What do we call the function?

Arguments: local variables accesible within the function. A function could have zero to infinite parameters.

Return Statment: The function could return after completing the something it was designed to do

Brackets: Define the start and end of the function

COMING BACK

```
function doSomething (num){
        return num+5;
}
```

How do you think I can use this function?

COMING BACK

```
function doSomething (num){
        return num+5;
}
```

Call it!

Block of code that does something, when it is called.

COMING BACK

```
function doSomething (num){
        return num+5;
}
```

Also, let's name this function. Let's name it addFive

COMING BACK

```
function addFive(num){
        return num+5;
};
addFive(100);
```

That it?

COMING BACK

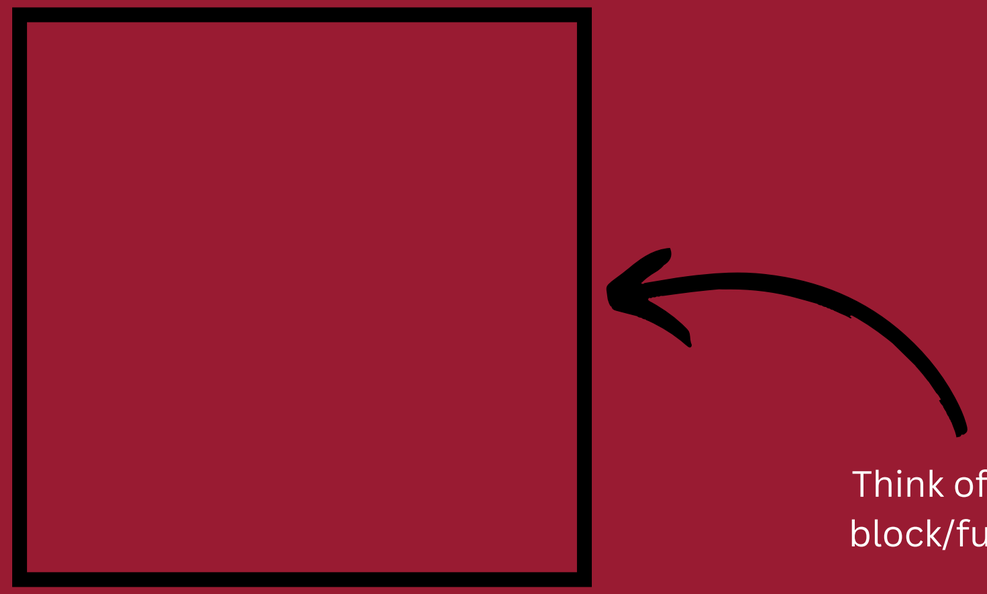
```
function addFive(num){
        return num+5;
};
let value = addFive(100);
```

SCOPING

We talked about a block of code.
What does it mean to be a block of code?

SCOPING

Let's think of this being a box



Think of this as
block/function.

SCOPING

Let's think of this being a box

```
function addFive(num){
    return num+5;
}
```

SCOPING

Let's think of this being a box

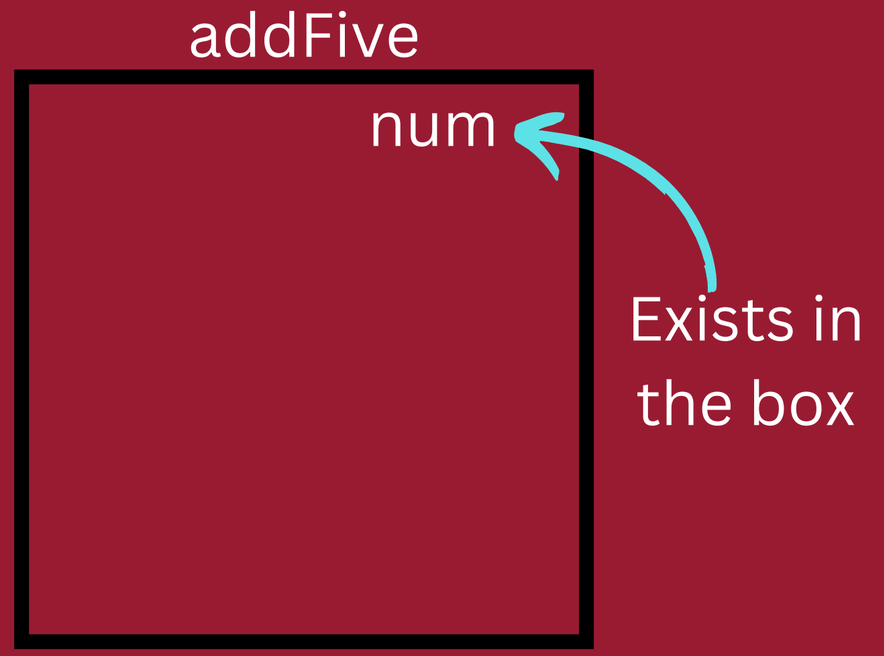
```
function addFive(num){
    return num+5;
}
```

Where do you think num will be?
in the box?
out of the box?

SCOPING

Let's think of this being a box

```
function addFive(num){
    return num+5;
}
```

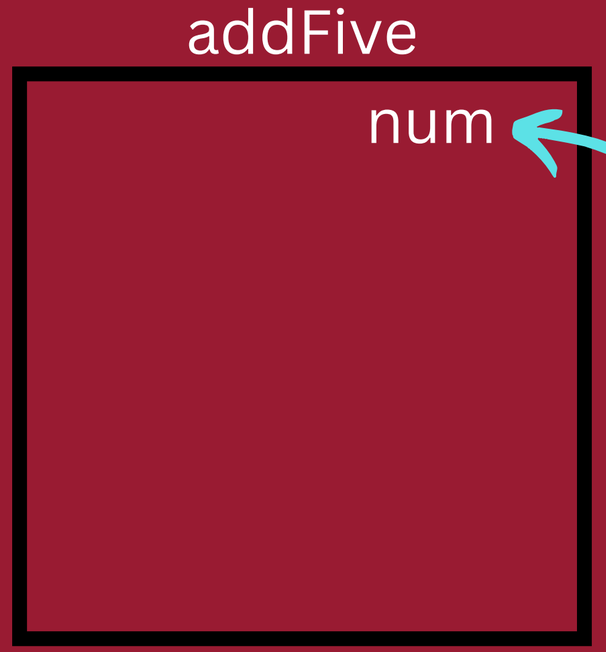


Arguments: **local** variables accesible within the function.

SCOPING

Let's think of this being a box

```
function addFive(num){
    return num+5;
}
```



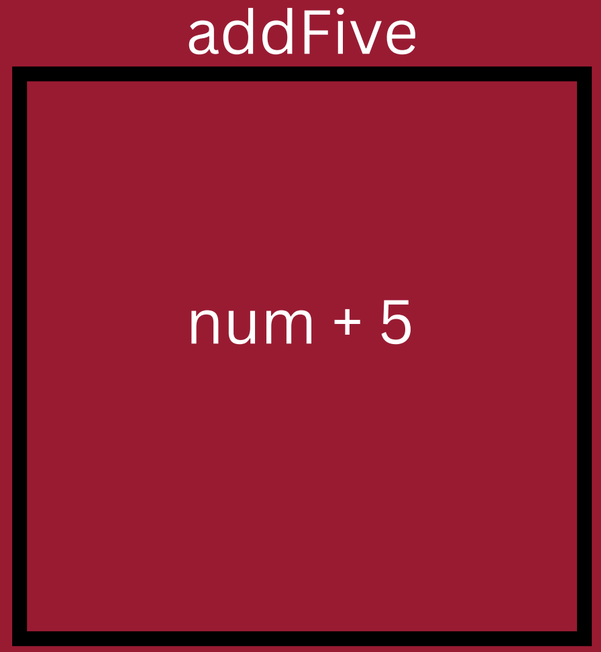
exists and accessible only in the box

Arguments: local variables accesible within the function.

SCOPING

Let's think of this being a box

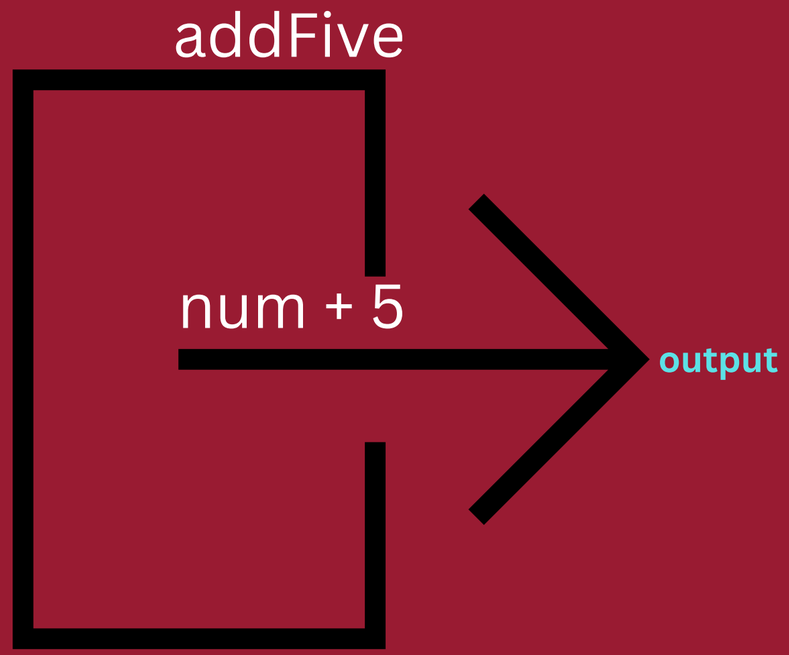
```
function addFive(num){
    return num+5;
}
```



SCOPING

Let's think of this being a box

```
function addFive(num){
    return num+5;
}
```



LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
const arg=100;
arg=addFive(arg);
```

Let's find out the value of arg at the end....

LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
const arg=100;
arg=addFive(arg);
```

Notice the Error?

Let's find out the value of arg at the end....

LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
const arg=100;
arg=addFive(arg);
```

Notice the Error?

We can't reassign const!

Let's find out the value of arg at the end....

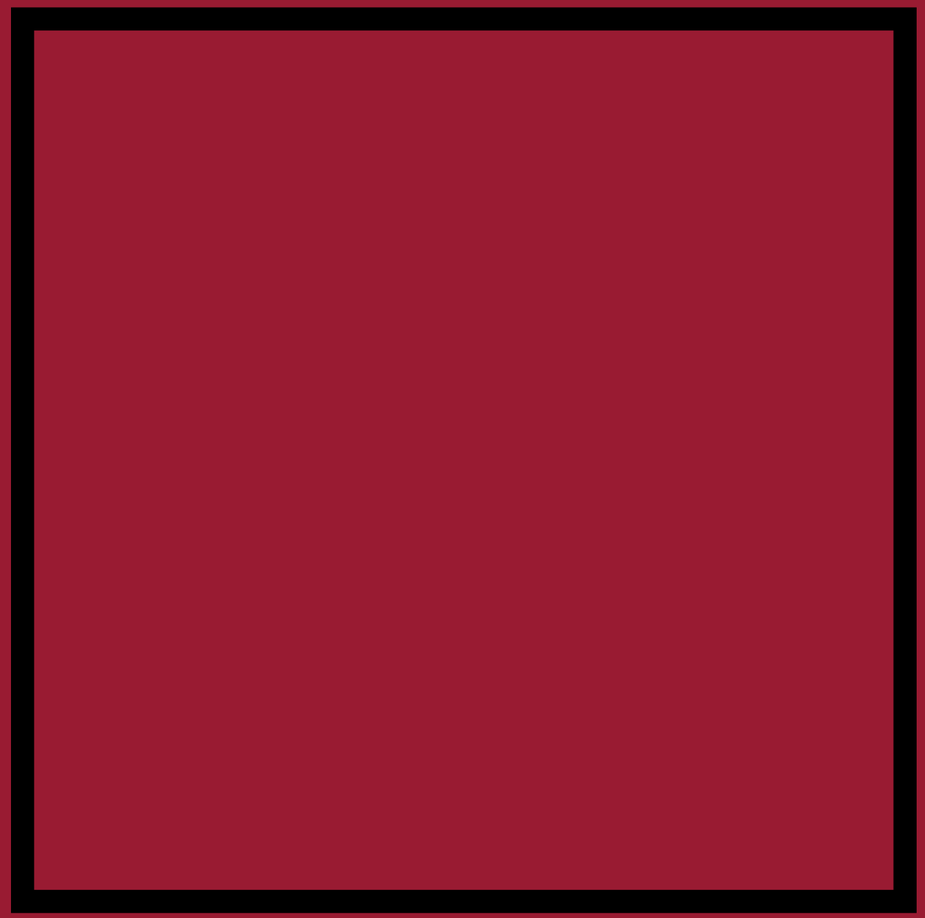
LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
let arg=100;
arg=addFive(arg);
```

Let's find out the value of arg at the end....

LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

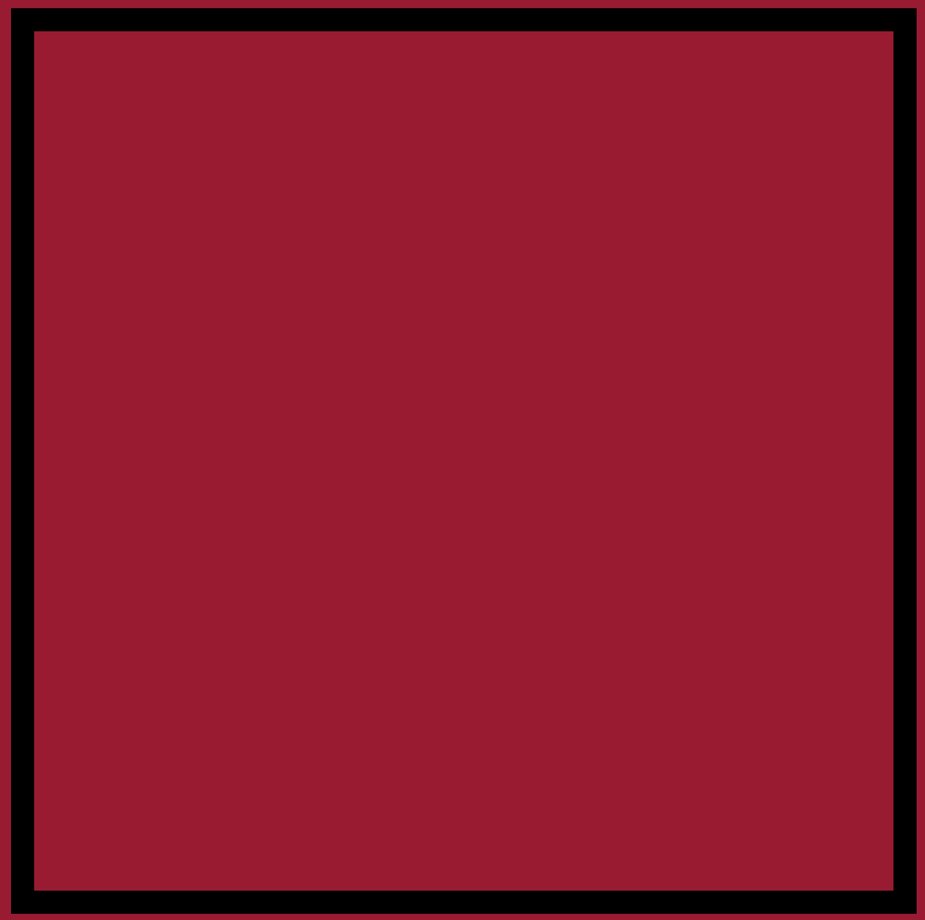


which box is this?
function?

LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

JS file

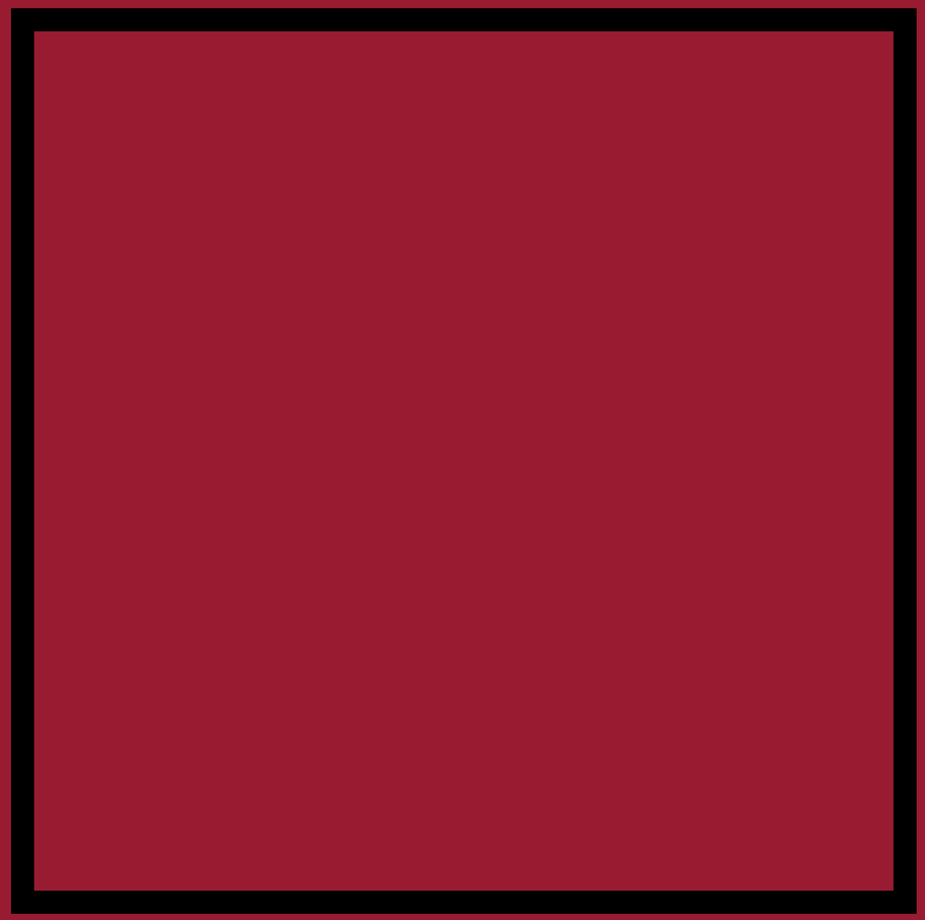


No!

LET'S ADD SOME MORE CODE

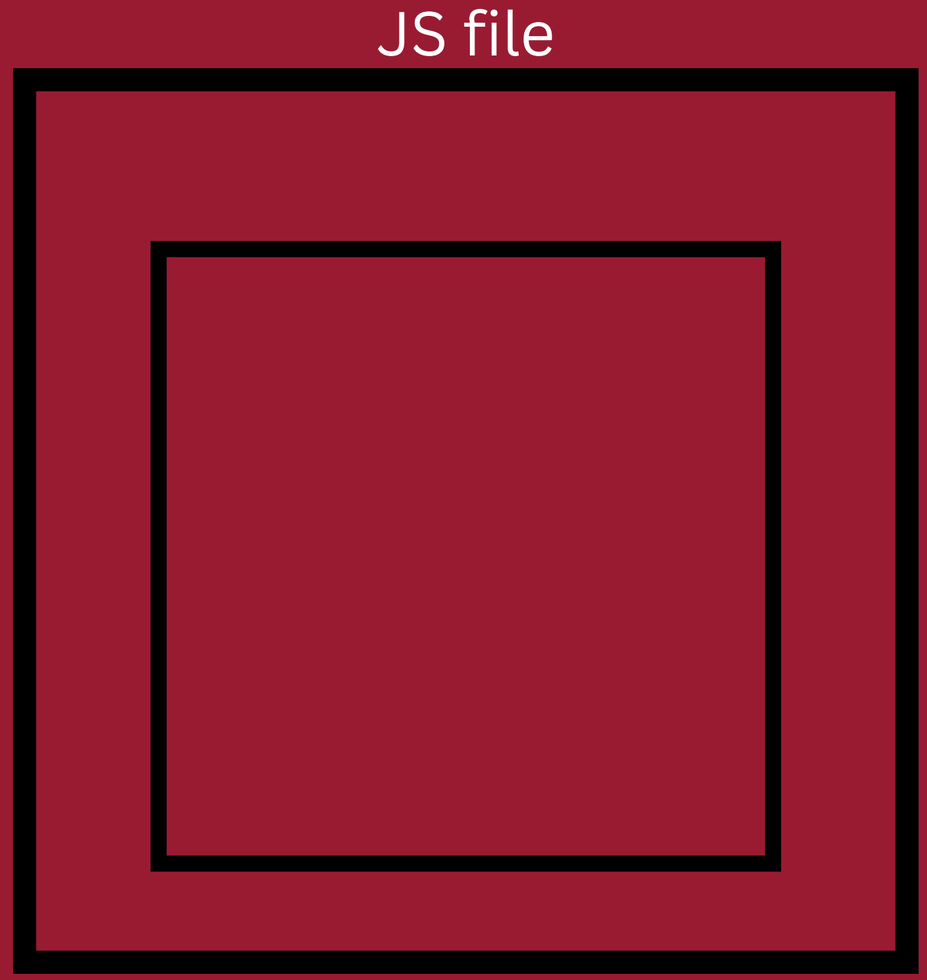
```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

JS file



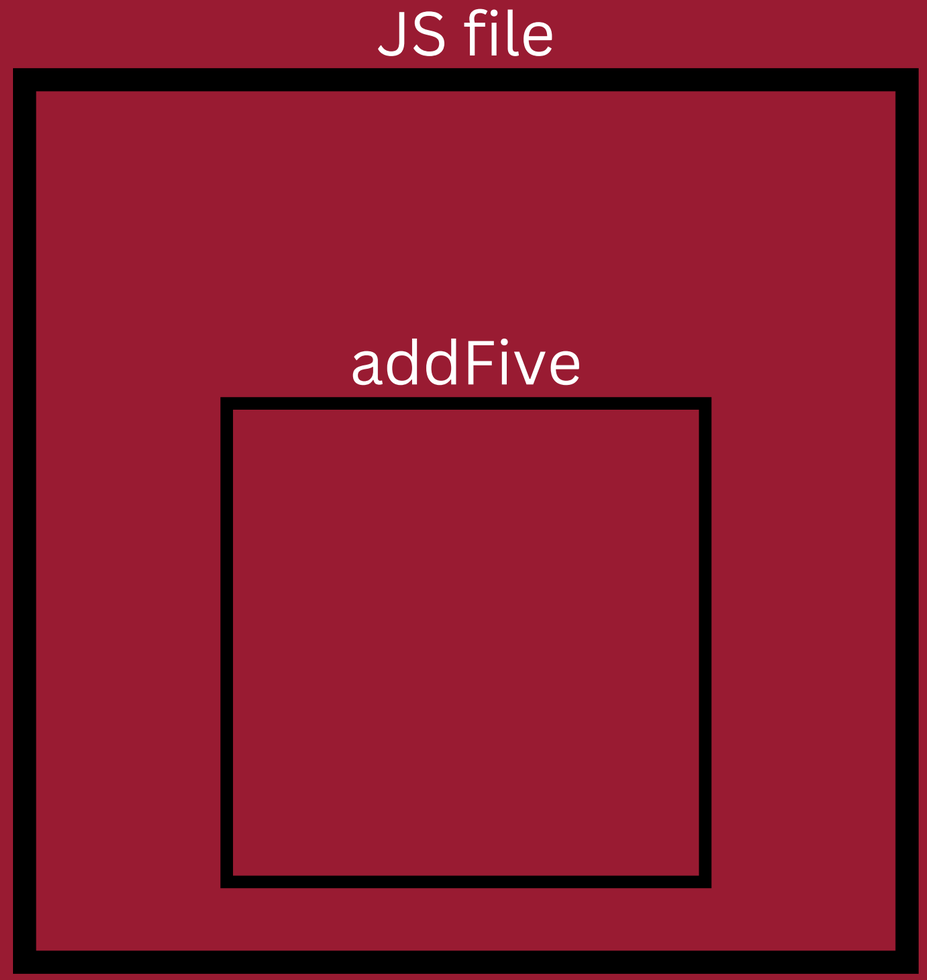
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

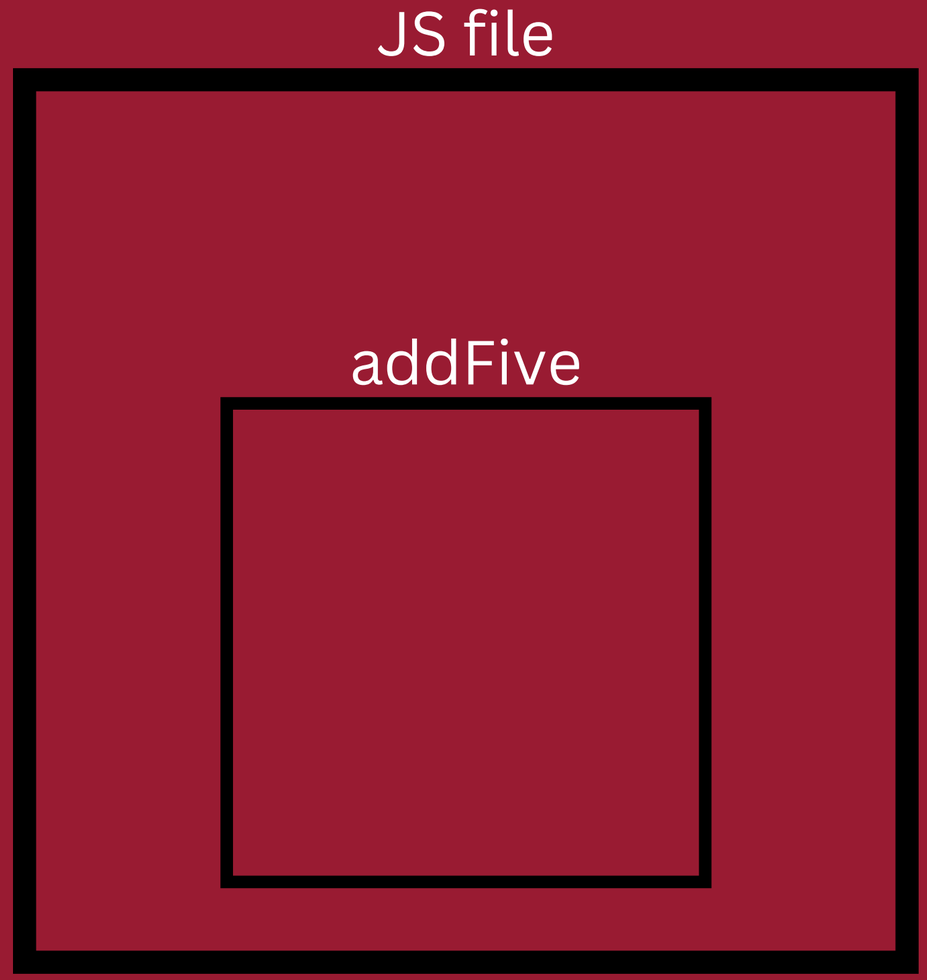


LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
let arg=100;
arg=addFive(arg);
```

We don't evaluate the function yet!

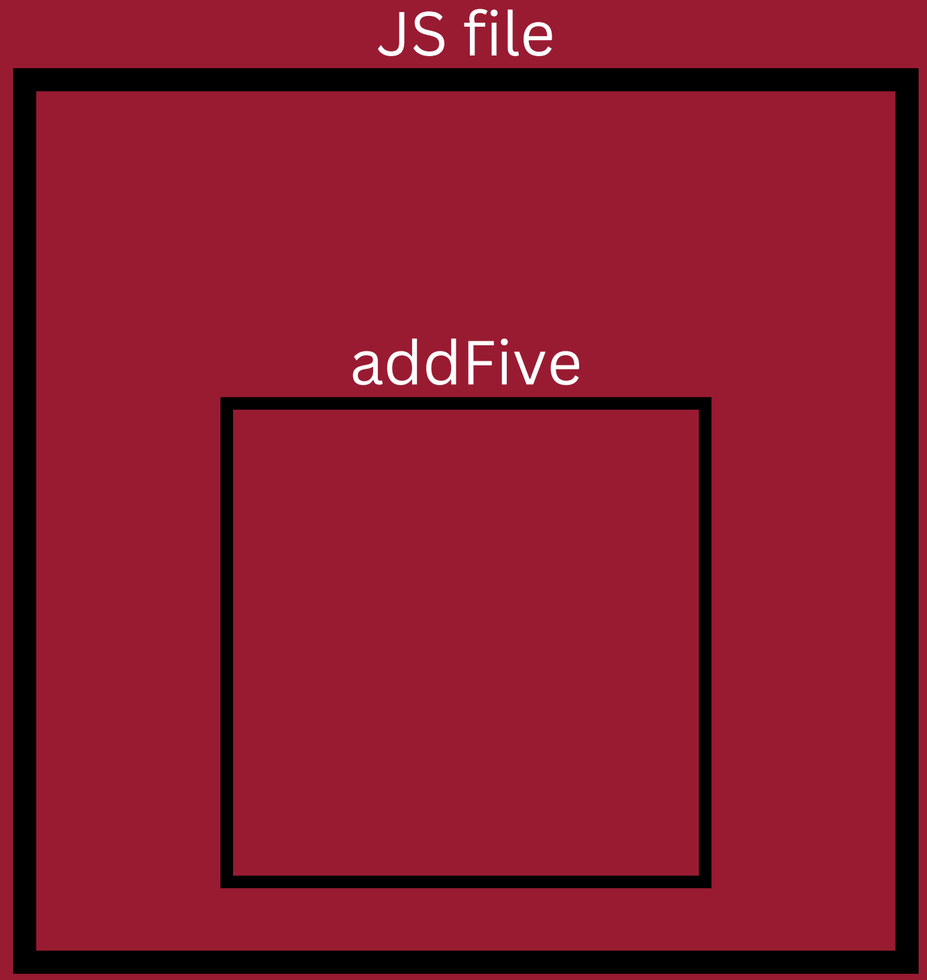
Becase it's not called yet



LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
let arg=100;
arg=addFive(arg);
```

Where will arg go?



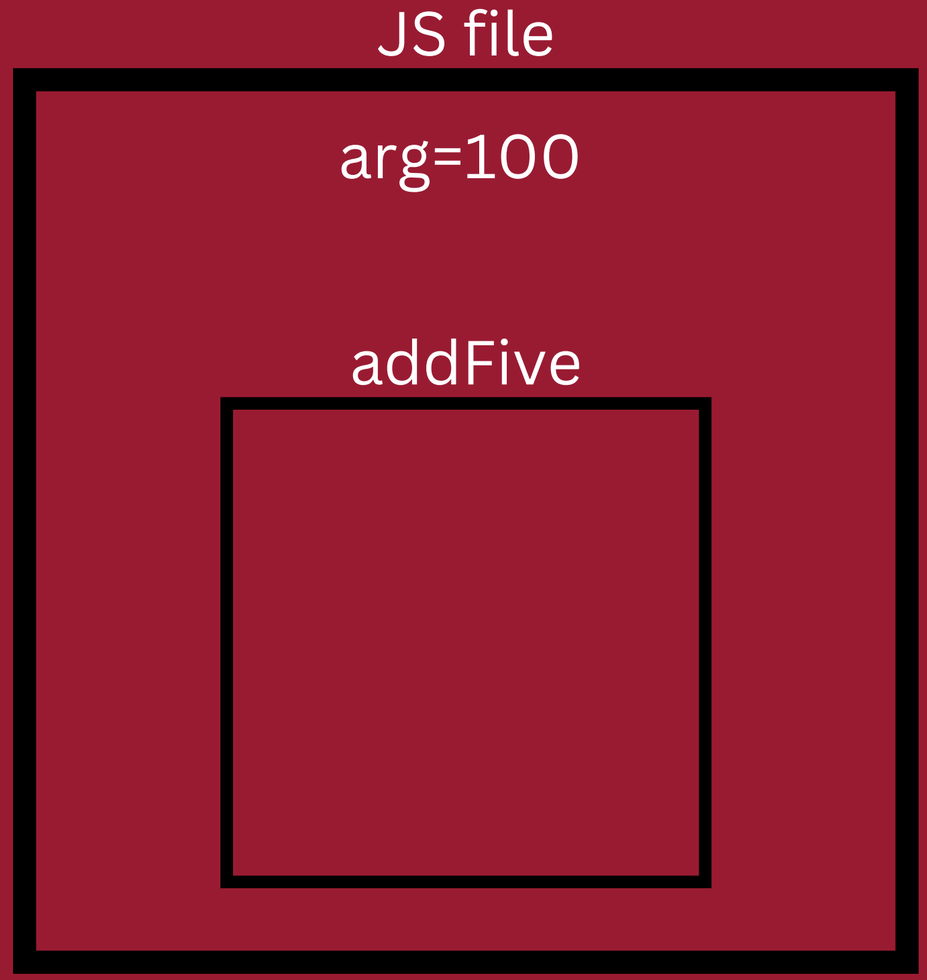
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

It's going to go in the **outer** box(**JS** file)

LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

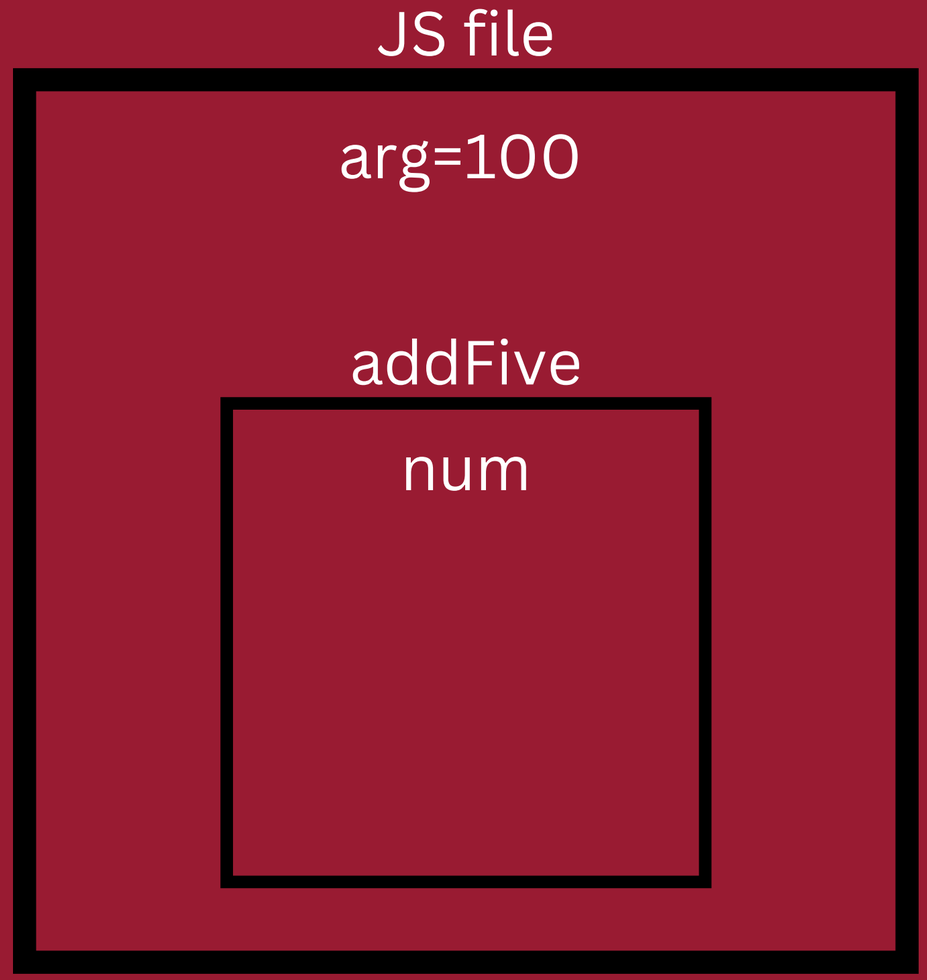


LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



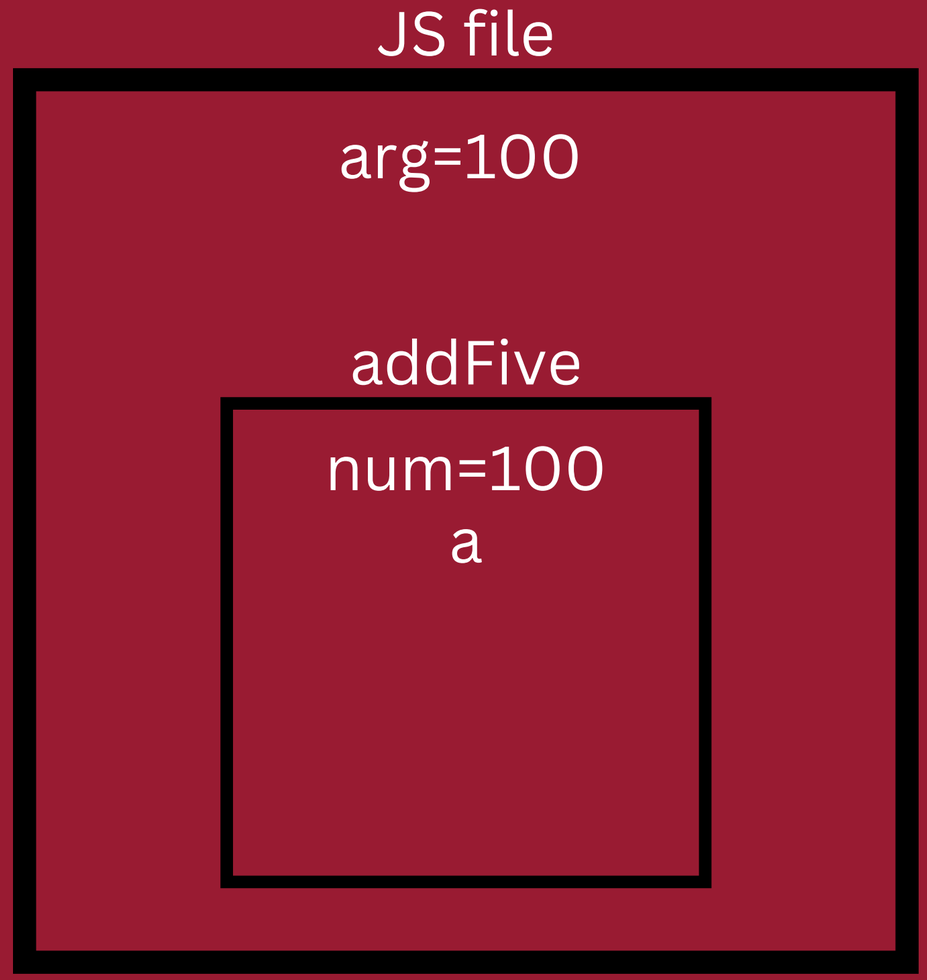
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```

We passed in the argument
which is arg = 100. Therefore num=100

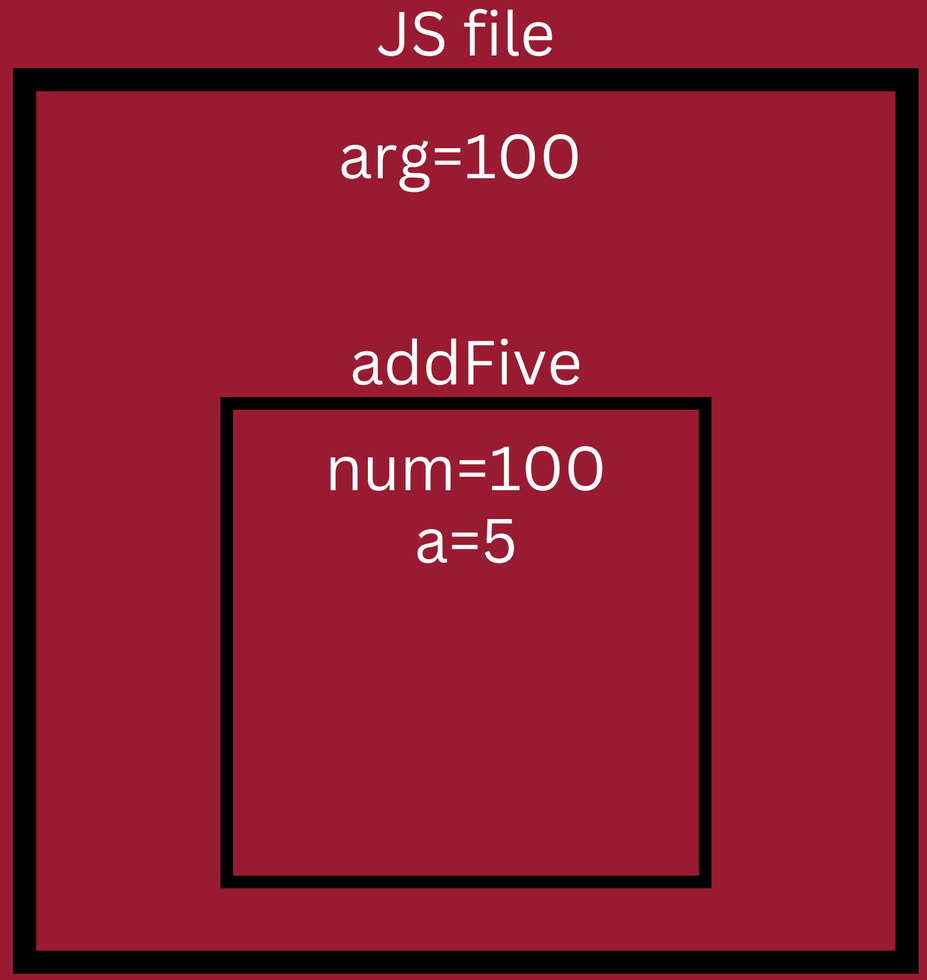
LET'S ADD SOME MORE CODE

```
function addFive(num){
    const a = 5;
    return num+a;
}
let arg=100;
arg=addFive(arg);
```



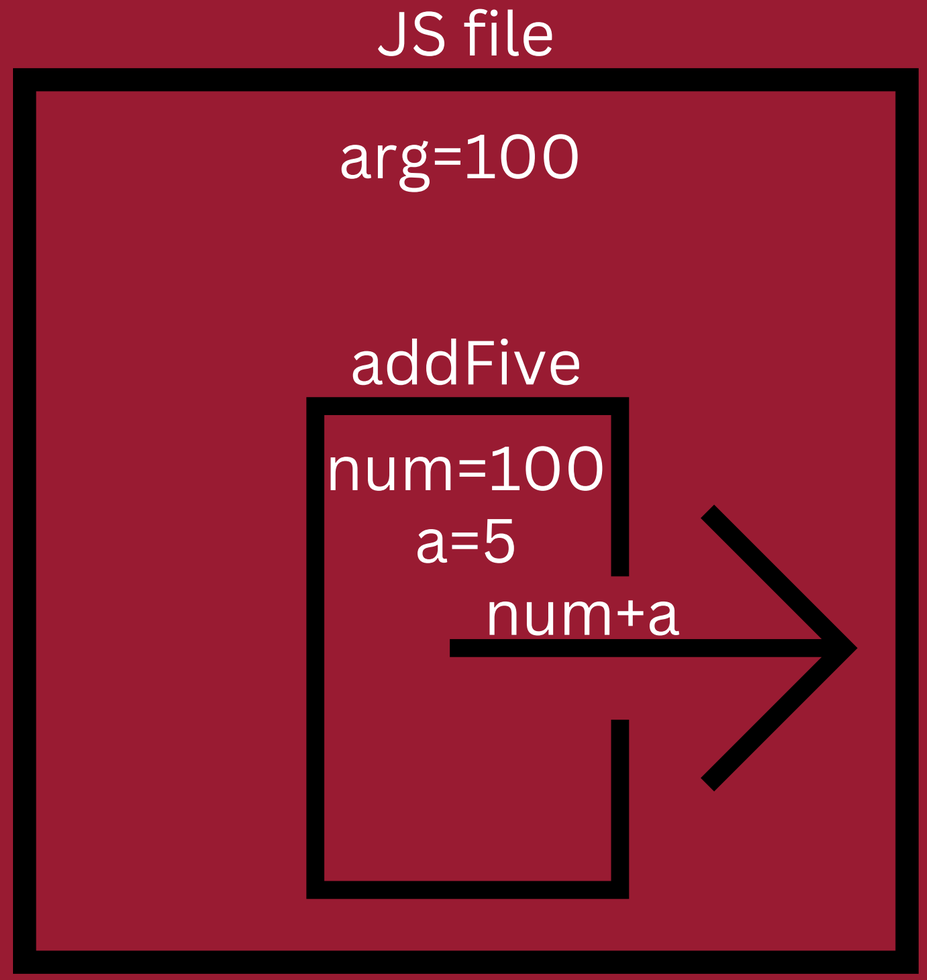
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



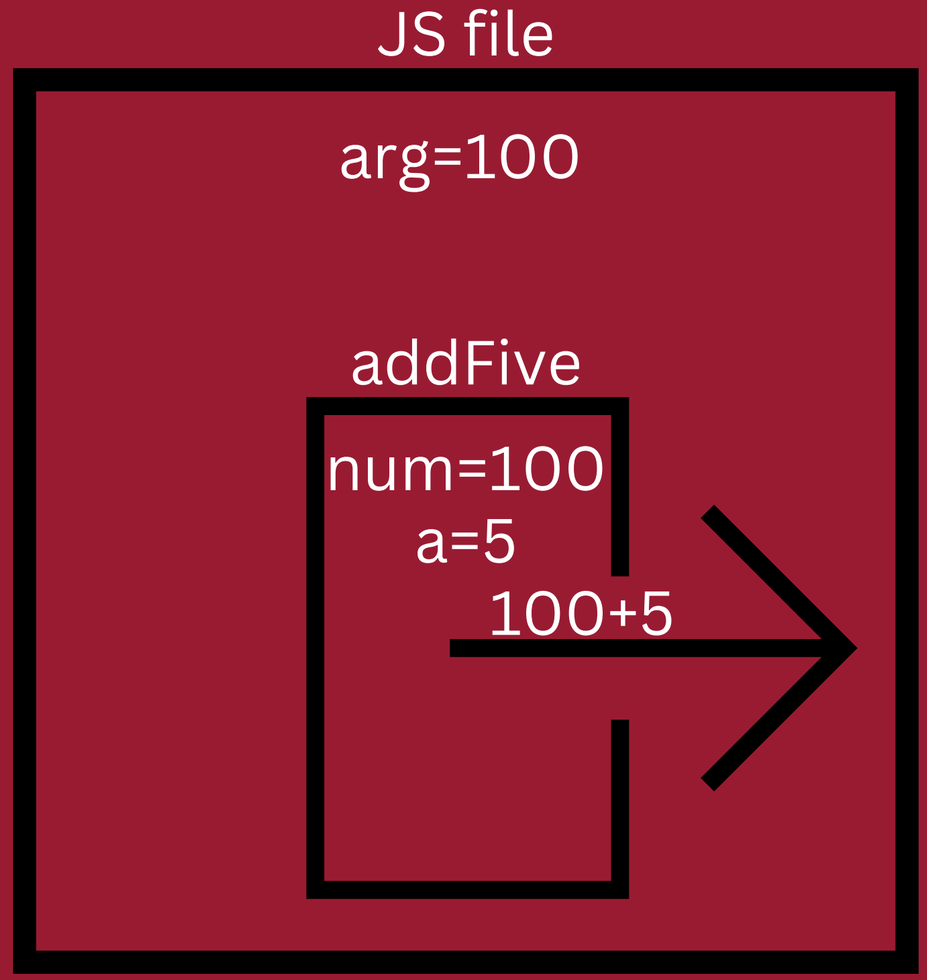
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



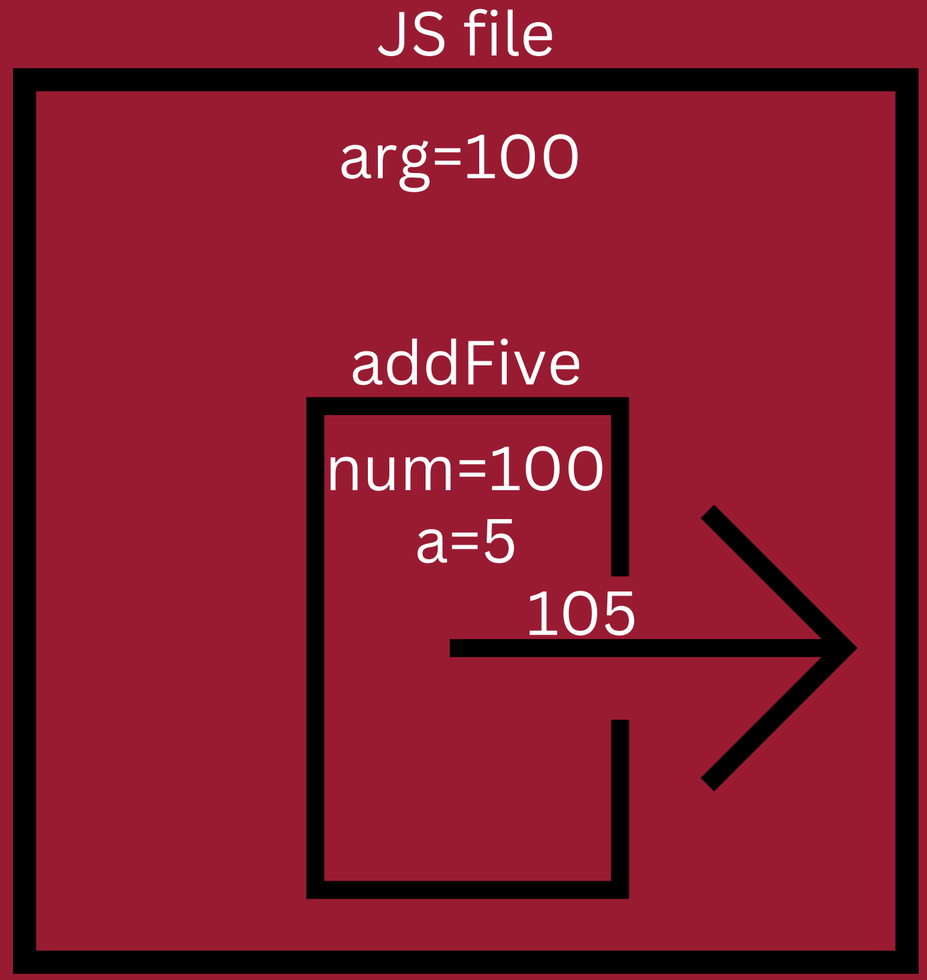
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



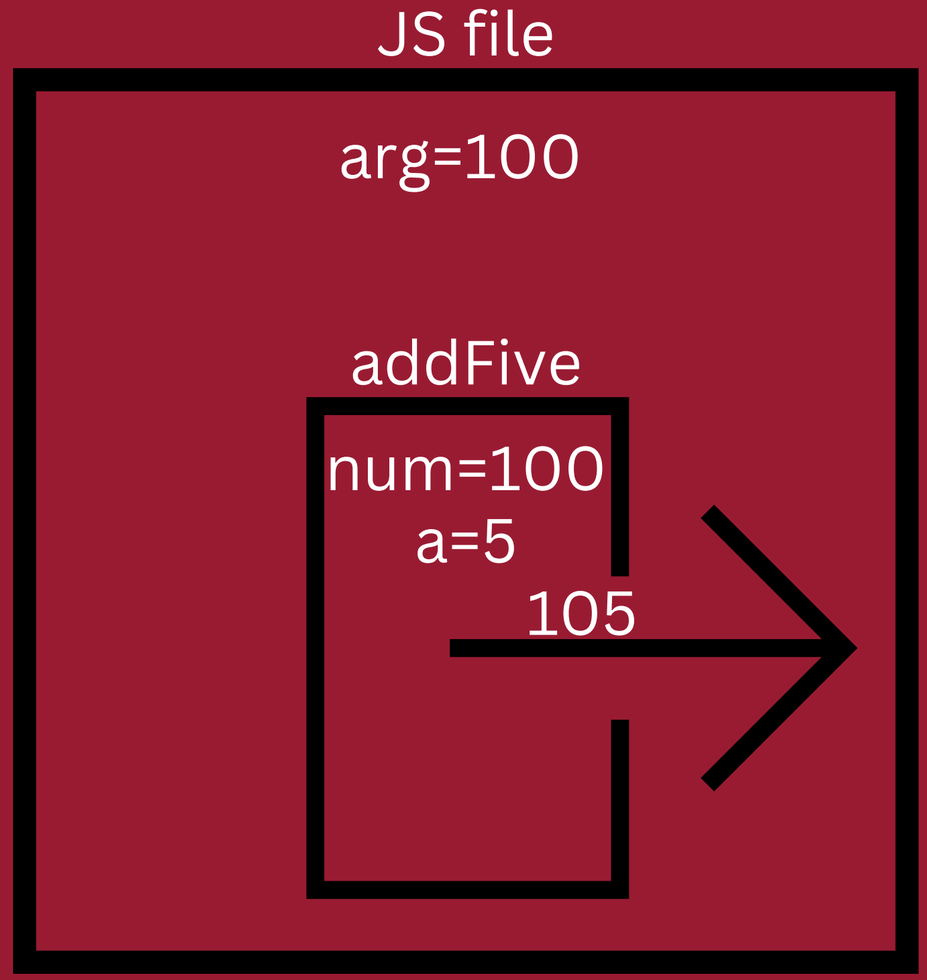
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



LET'S ADD SOME MORE CODE

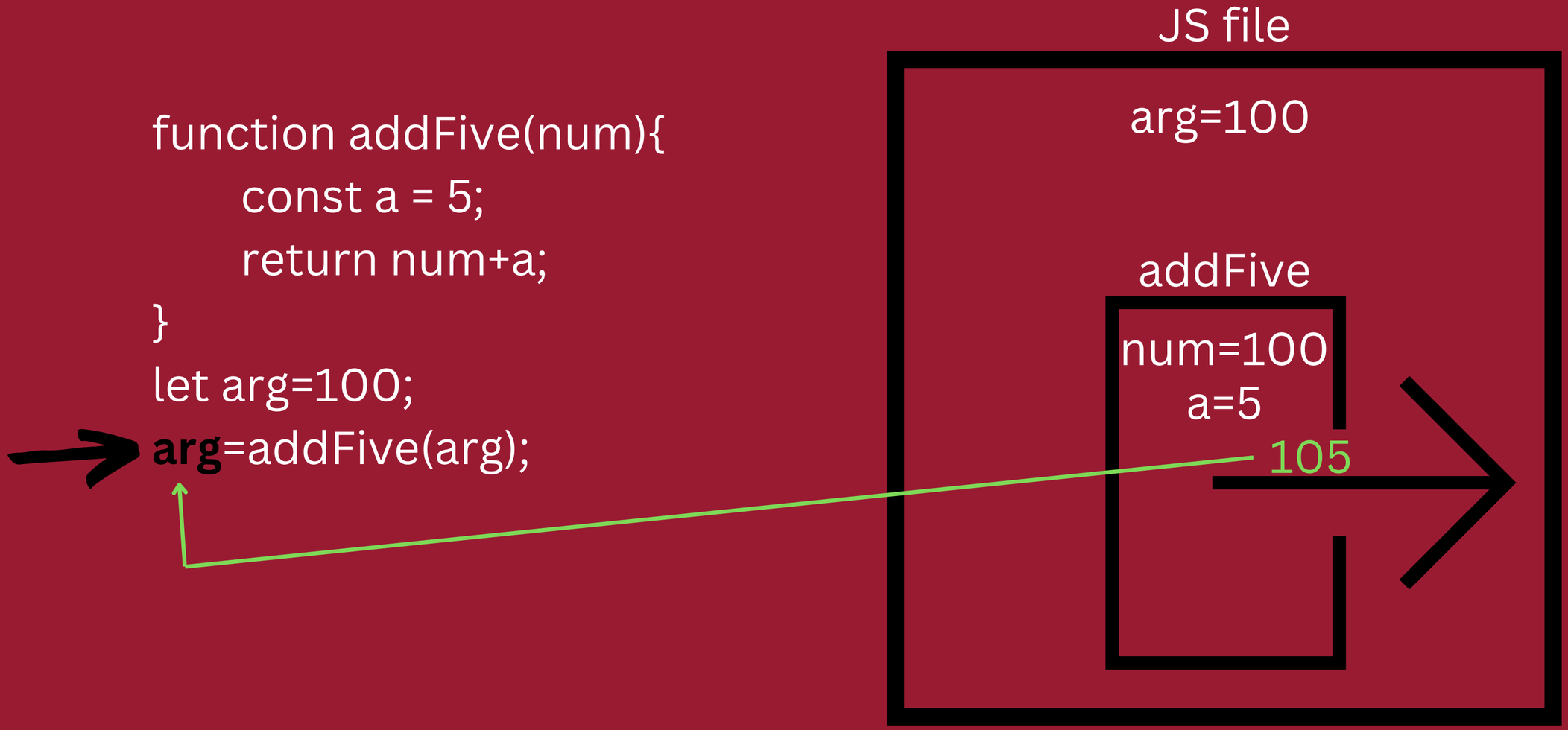
```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



where is the returned value stored?

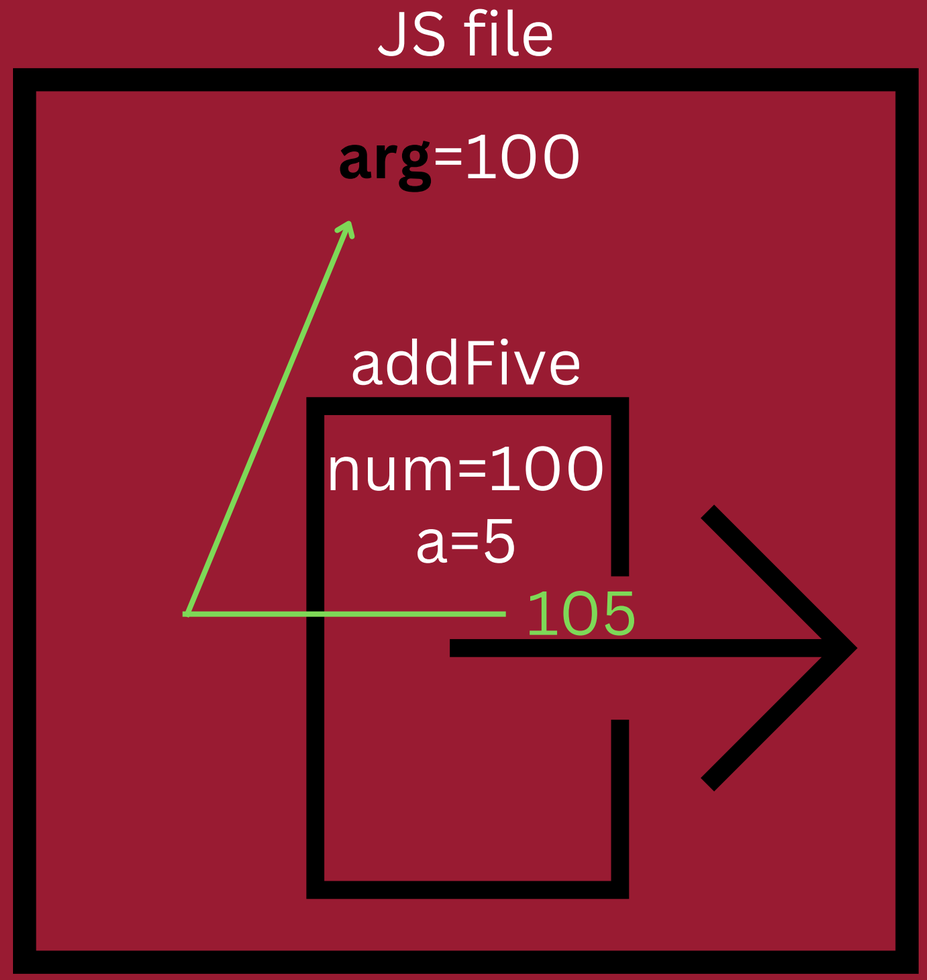
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



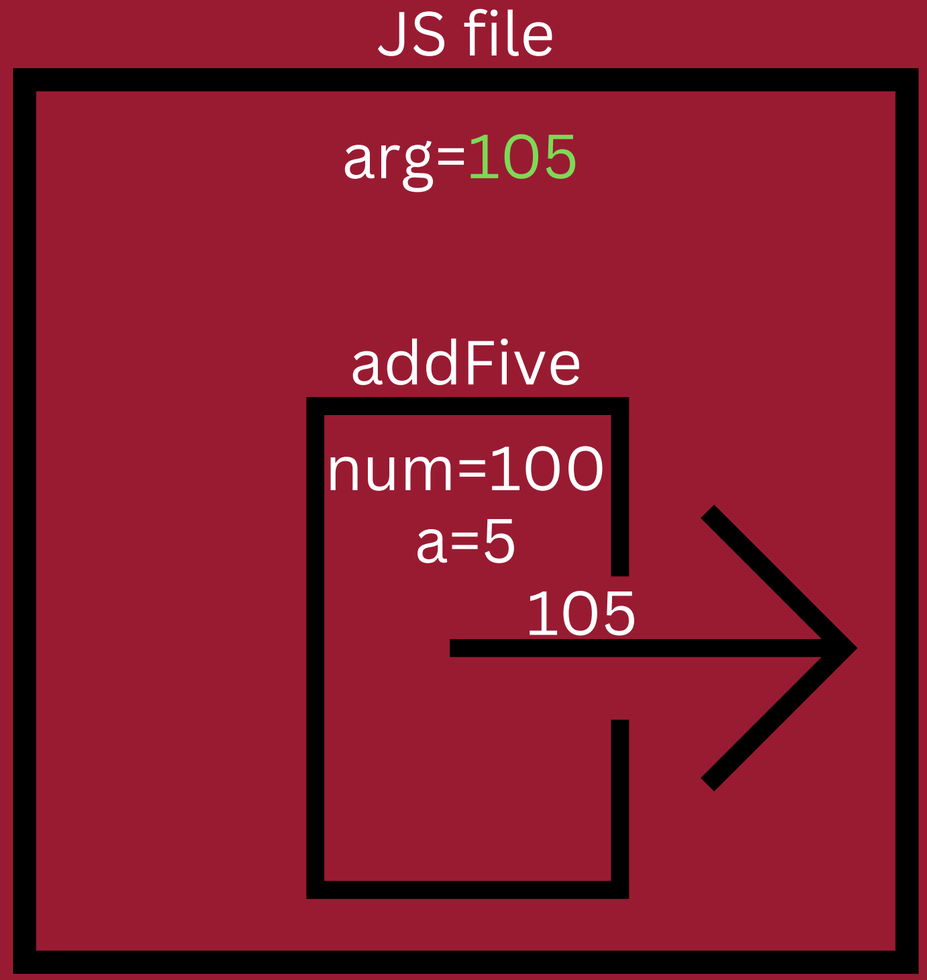
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



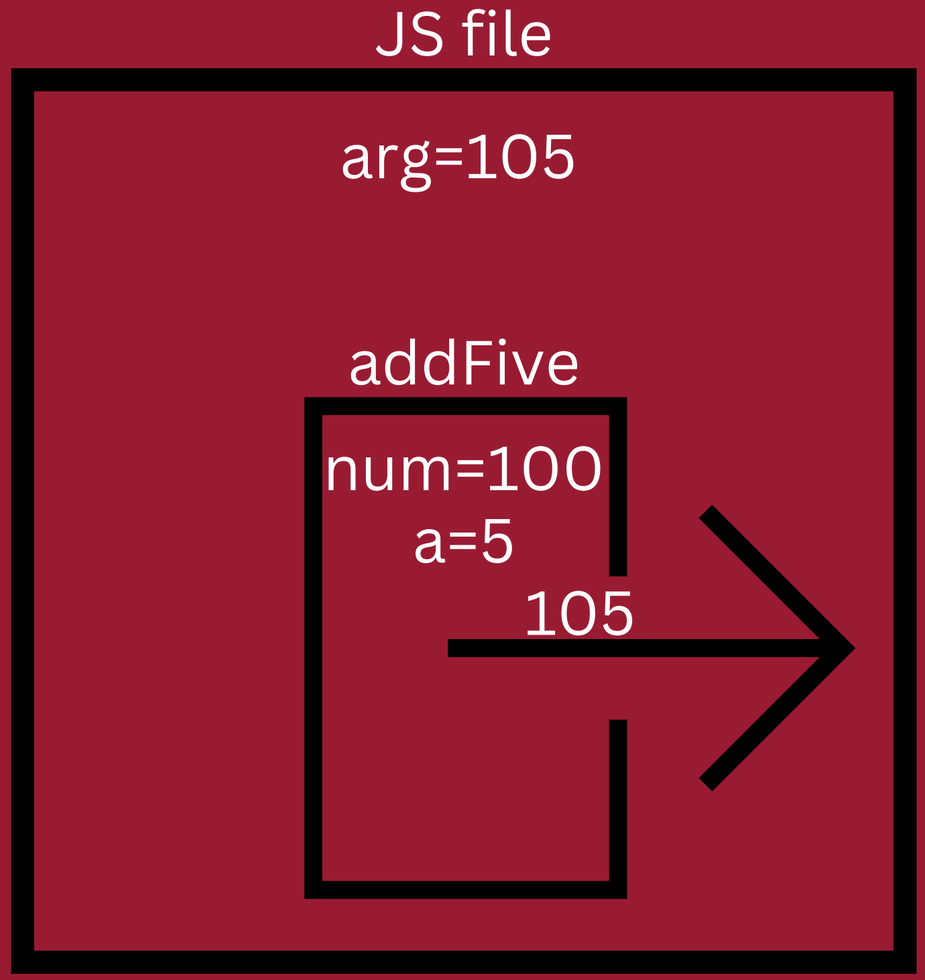
LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



LET'S ADD SOME MORE CODE

```
function addFive(num){
        const a = 5;
        return num+a;
}
let arg=100;
arg=addFive(arg);
```



Therefore **arg** is **105**

CICSOFT

VS?

Variable	Re-assign?	Scope	Good to use
Const			?
let			?
var			?

VS?

Variable	Re-assign?	Scope	Good to use
Const			
let			?
var			?

VS?

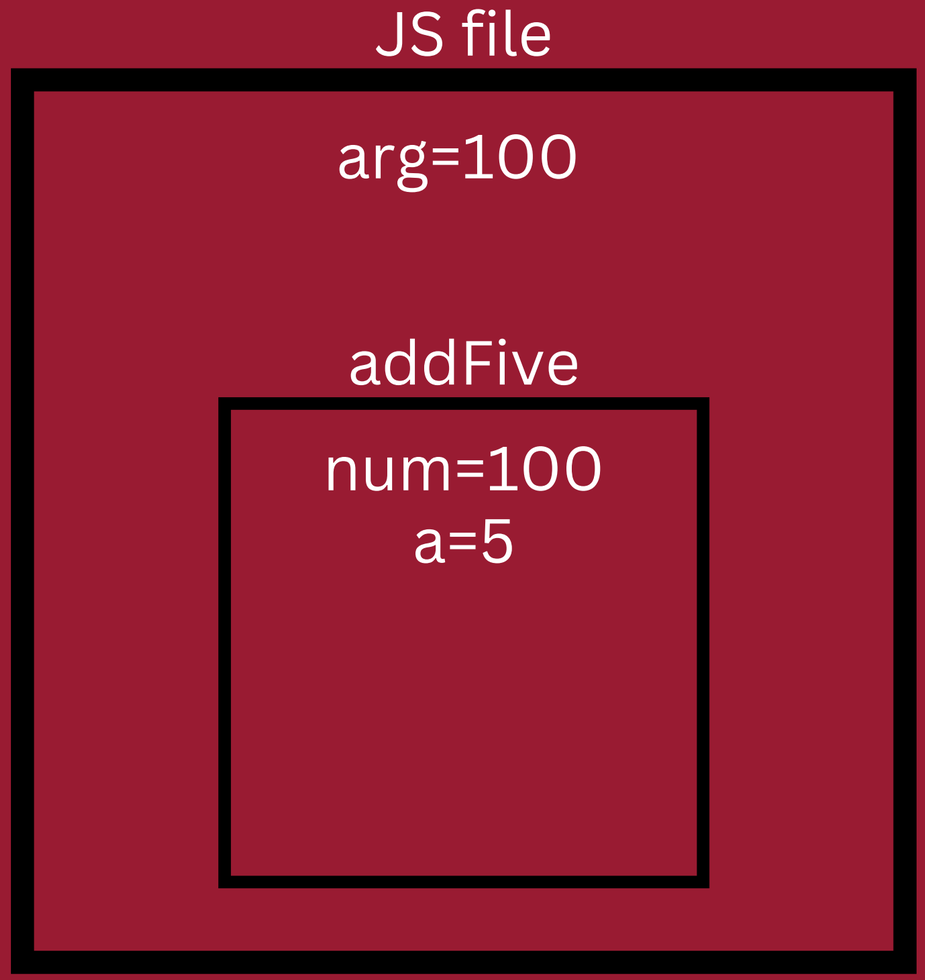
Variable	Re-assign?	Scope	Good to use
Const			
let			
var			

VS?

Variable	Re-assign?	Scope	Good to use
Const			
let			
var			

WHY?

The inner box is like a one way mirror.

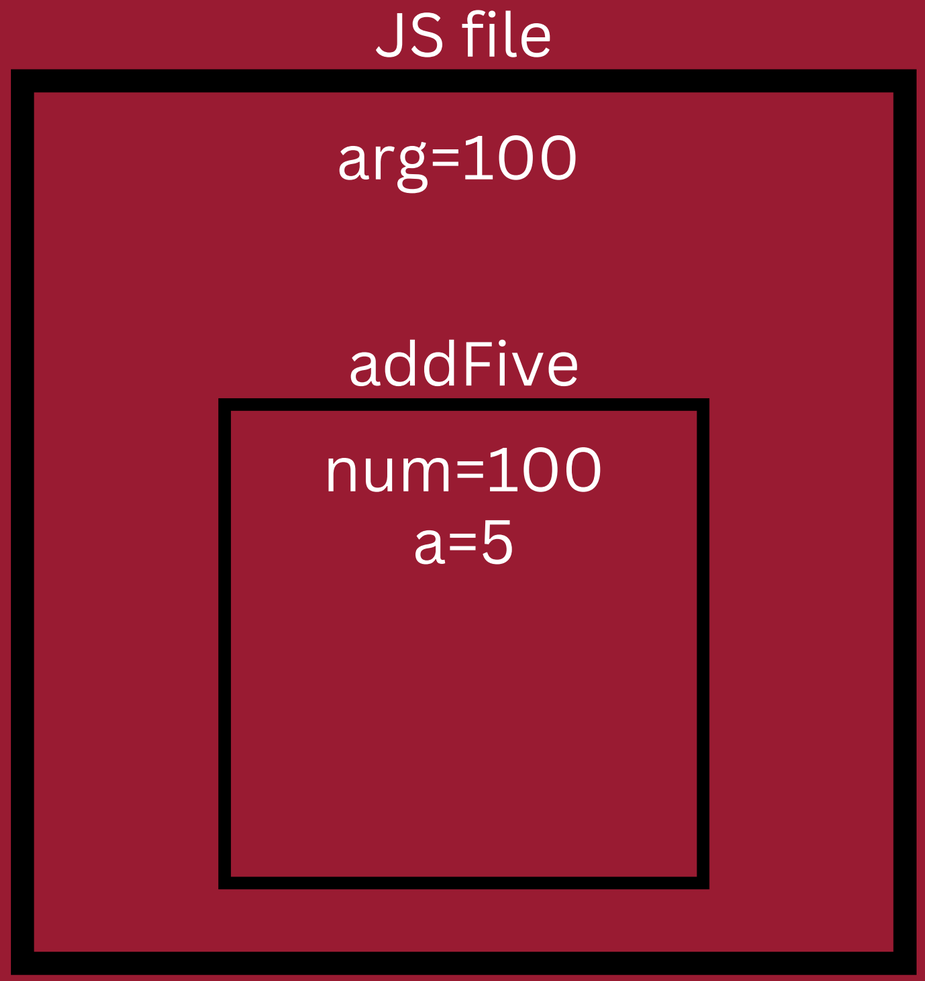


You can use variables defined outside the box inside, but not the other way around!

WHY?

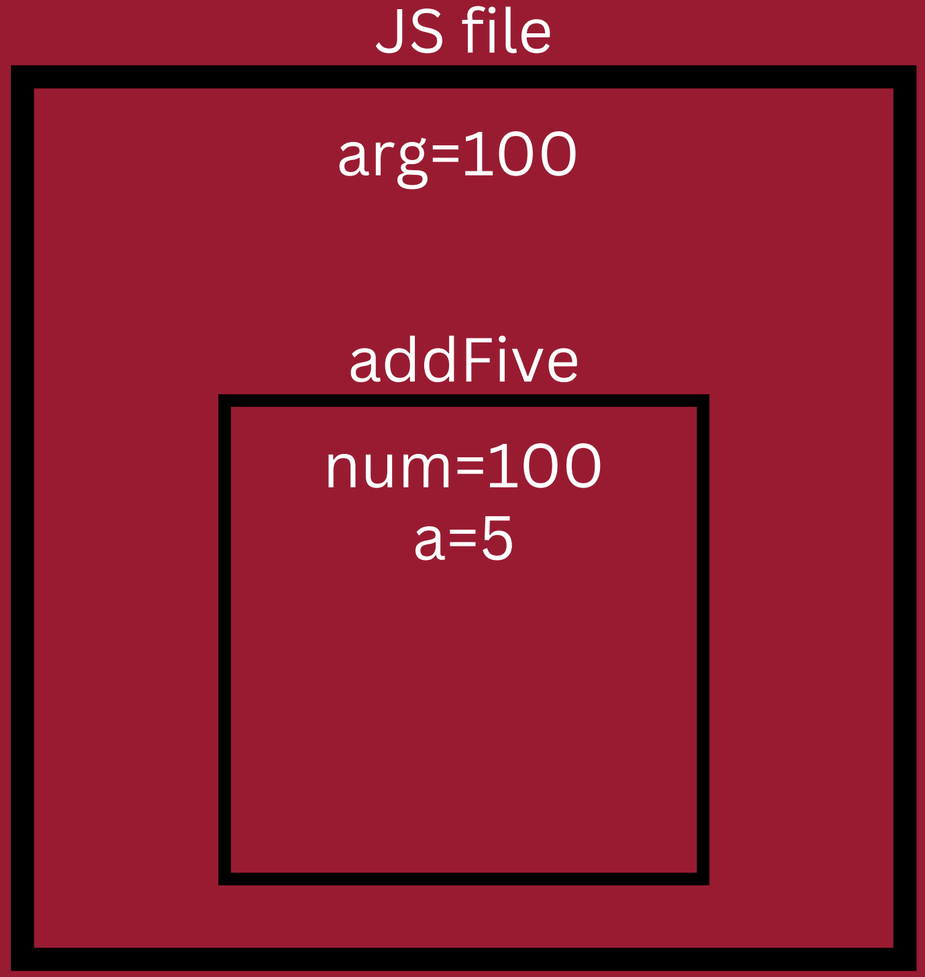
Which means I can use
arg in the function without any problems.

But I can't use variable a globally.



WHY?

```
> function addFive(num){
        const a = 5;
        return num+a;
  }
  let arg=100;
  arg=addFive(arg);
< 105
> function addFive(num){
        const a = 5;
        return num+arg;
  }
  let arg=100;
  arg=addFive(arg);
< 200
> function addFive(num){
        const a = 5;
        return num+a;
  }
  let arg=100;
  arg=addFive(arg)+a;
✕ ▶Uncaught ReferenceError: a is not defined
      at <anonymous>:6:18
```




```
var x = 10;
```

```
let x = 10;
```

With var, since it defines all variables to be global, they can be accessed anywhere, and could be changed by accident leading to errors!

CICSOFT

ARROW FUNCTIONS

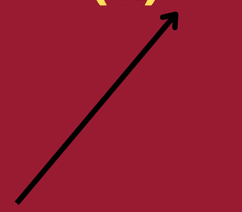
```
> let test=(a)=>{return a+2;}
< undefined
> test(5)
< 7
```

instead of saying

```
function test(a){
    return a+2;
}
```

you can also say

```
const test = (a) => {return a+2;}
```



This is called an arrow,
and functions that use arrows
are called arrow functions

*We don't use function keyword

ANONYMOUS FUNCTIONS

```
> let test=(a)=>{return a+2;}
< undefined
> test(5)
< 7
```

we can also represent this
as an anonymous function
(a) => {return a+2;}

function has no name, how do we access it?
High Order Functions use this a lot.

We will talk about them later.

*We don't name them or store them in a variable

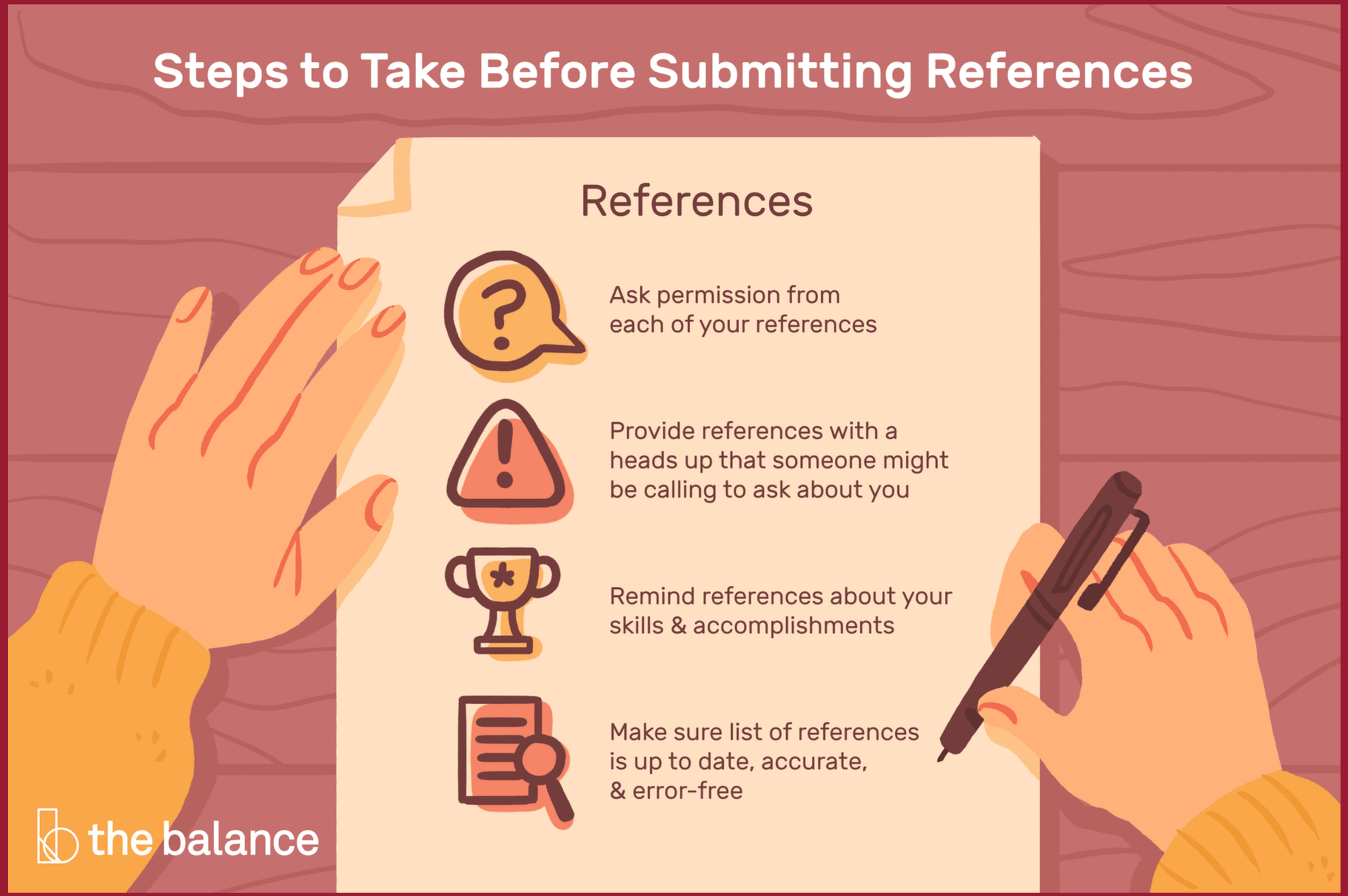
ACTIVITY

Write a function that calculates
the factorial of the given argument.

ACTIVITY

Now with using the same function, calculate the factorial of a global variable called factNum instead of using the argument

Steps to Take Before Submitting References



CICSOFT

REFERENCES

The primitive types are passed by **value**.

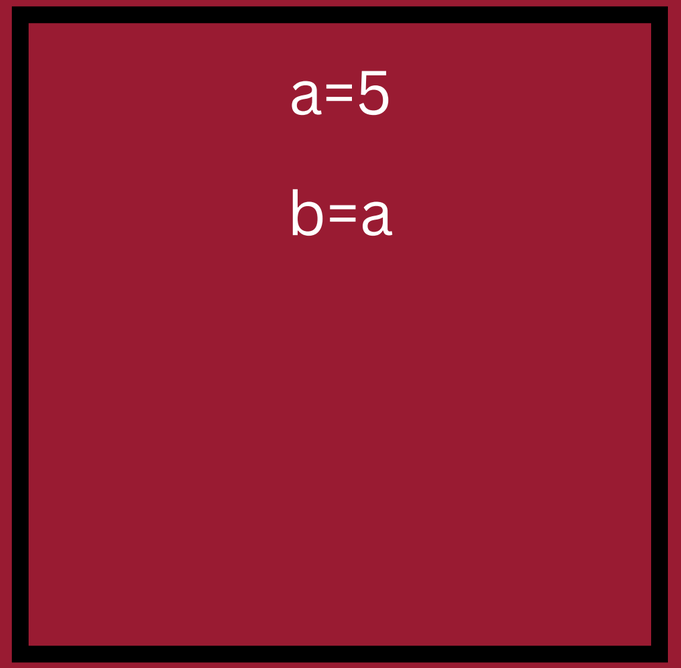
Arrays and Objects are passed by **refrences**.

REFERENCES

This means when I say

let a = 5;

let b = a;

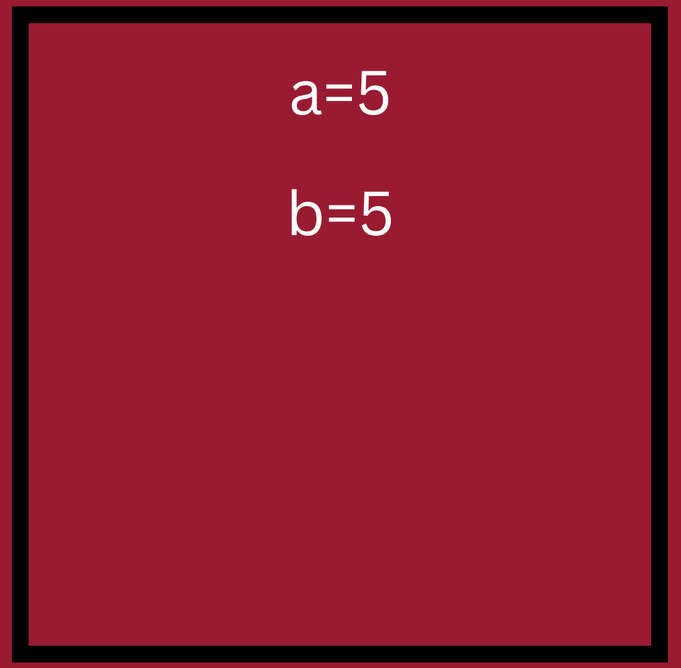


REFERENCES

This means when I say

let a = 5;

let b = a;



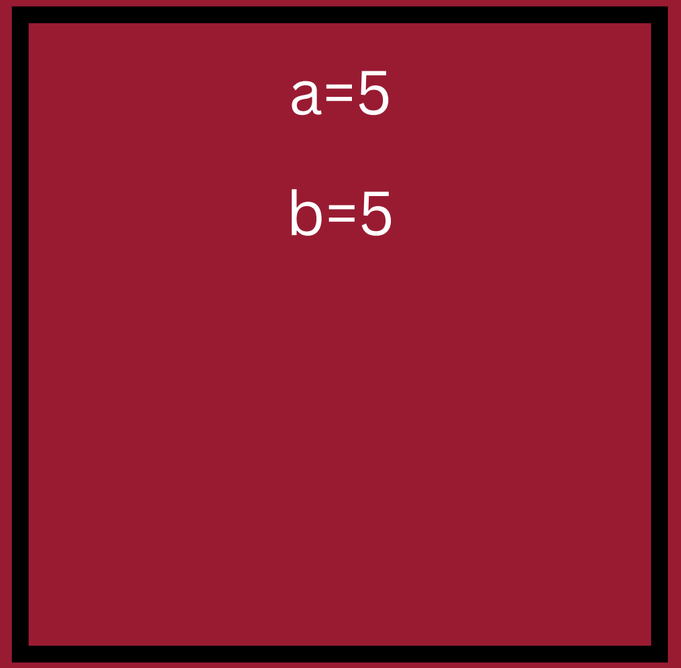
REFERENCES

This means when I say

let a = 5;

let b = a;

a=10;



will both a and b change?

REFERENCES

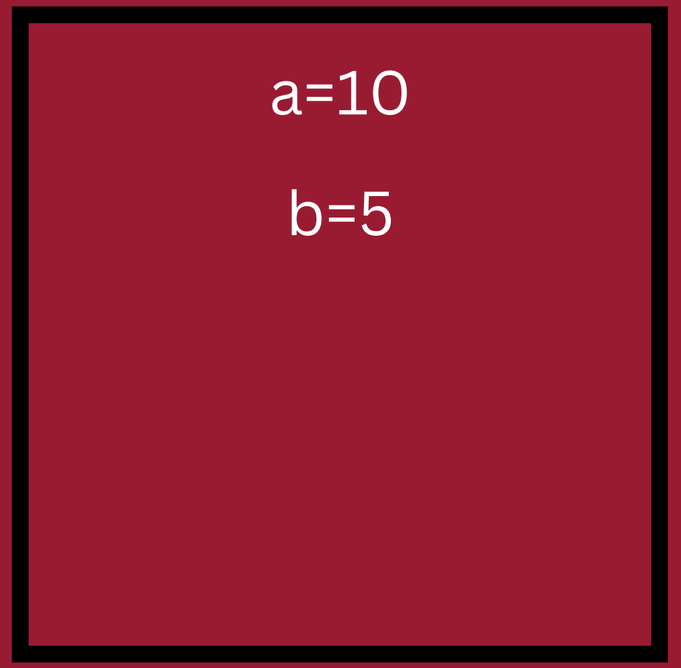
This means when I say

let a = 5;

let b = a;

a=10;

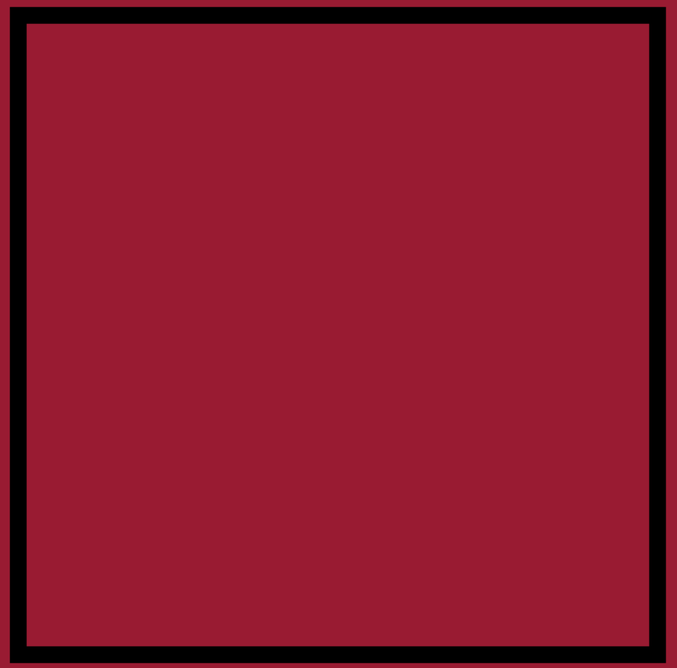
only a changes



REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```

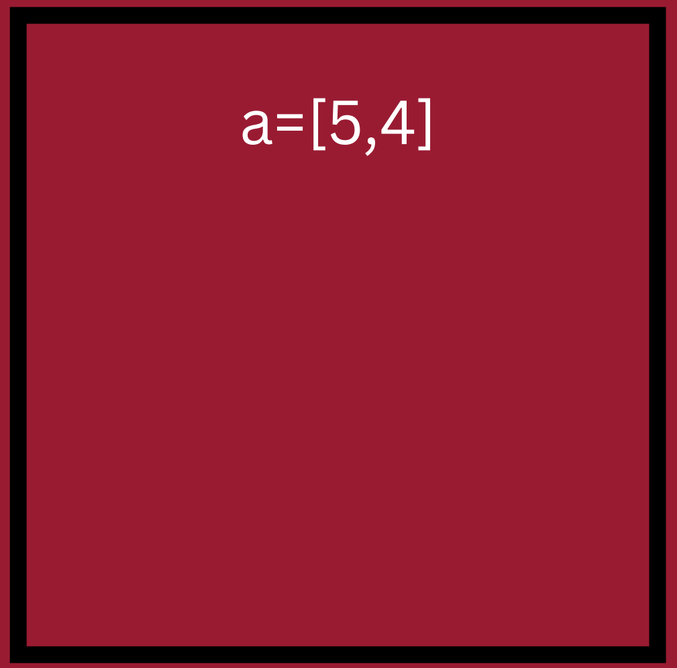


what will this look like?

REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```

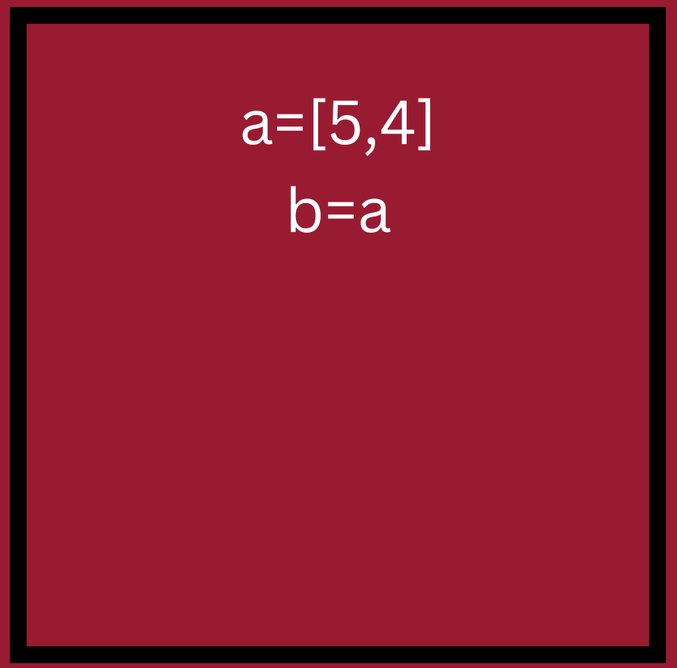


what will this look like?

REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```

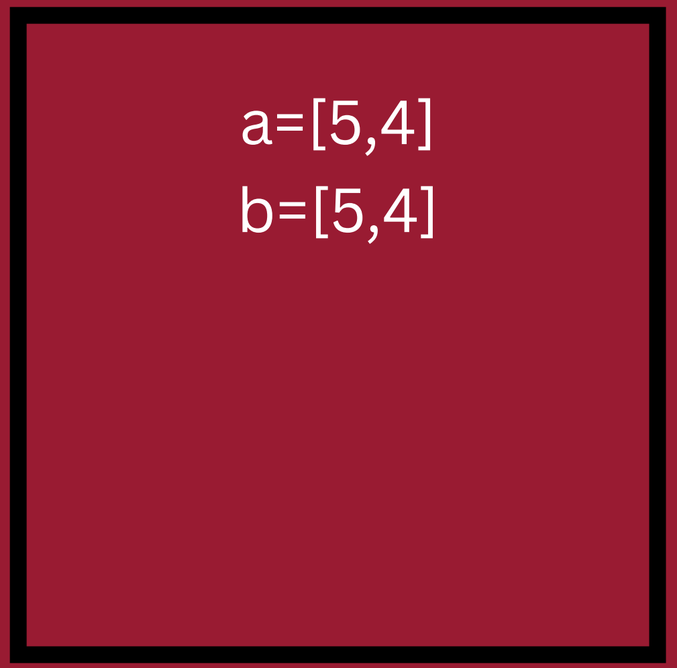


what will this look like?

REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```

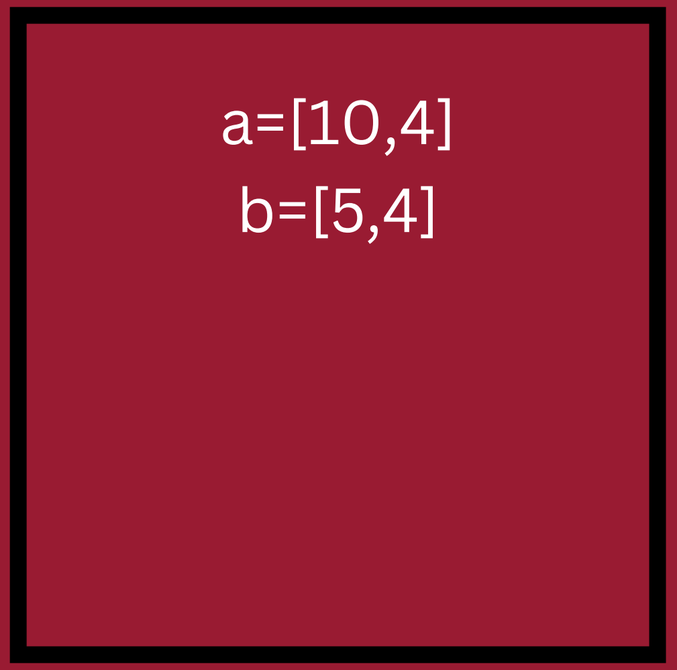


what will this look like?

REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```



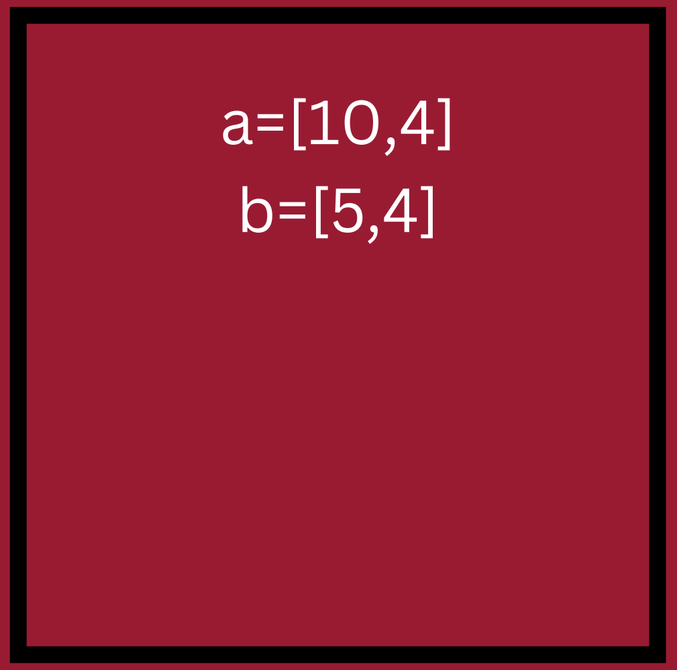
what will this look like?

Correct?

REFERENCES

Now if we had

```
let a = [5,4];
let b = a;
a[0]=10;
```



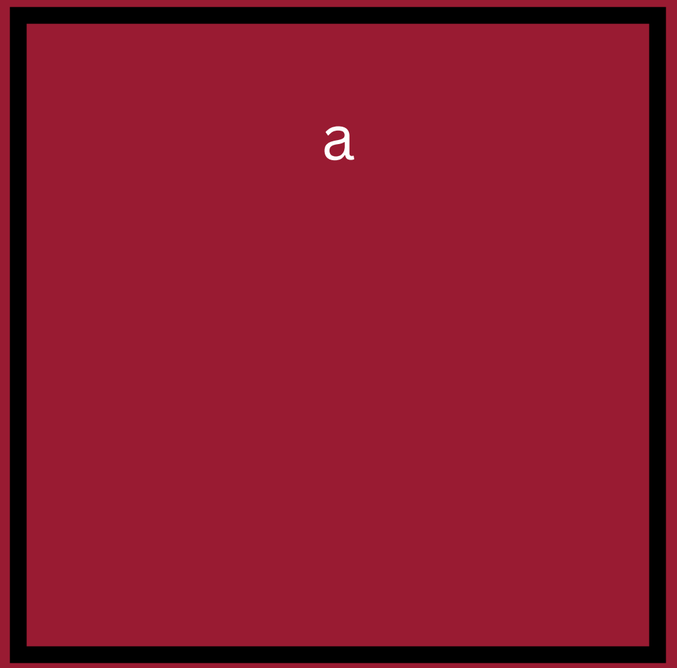
what will this look like?

Wrong!

REFERENCES

Now if we had

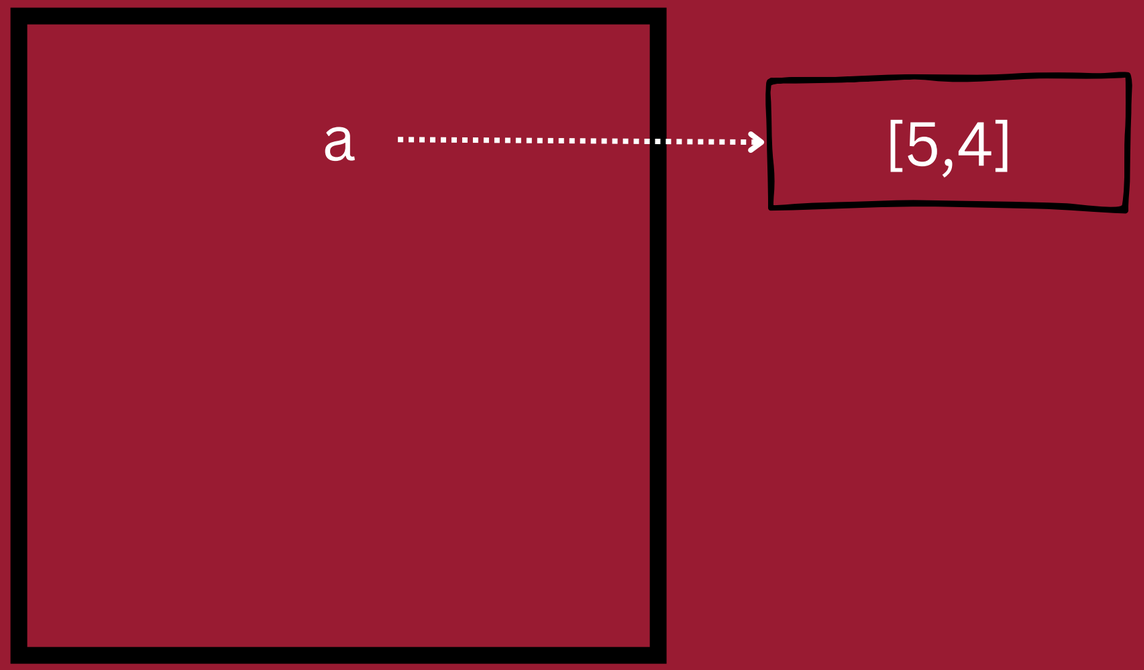
```
let a = [5,4];
let b = a;
a[0]=10;
```



what will this look like?

REFERENCES

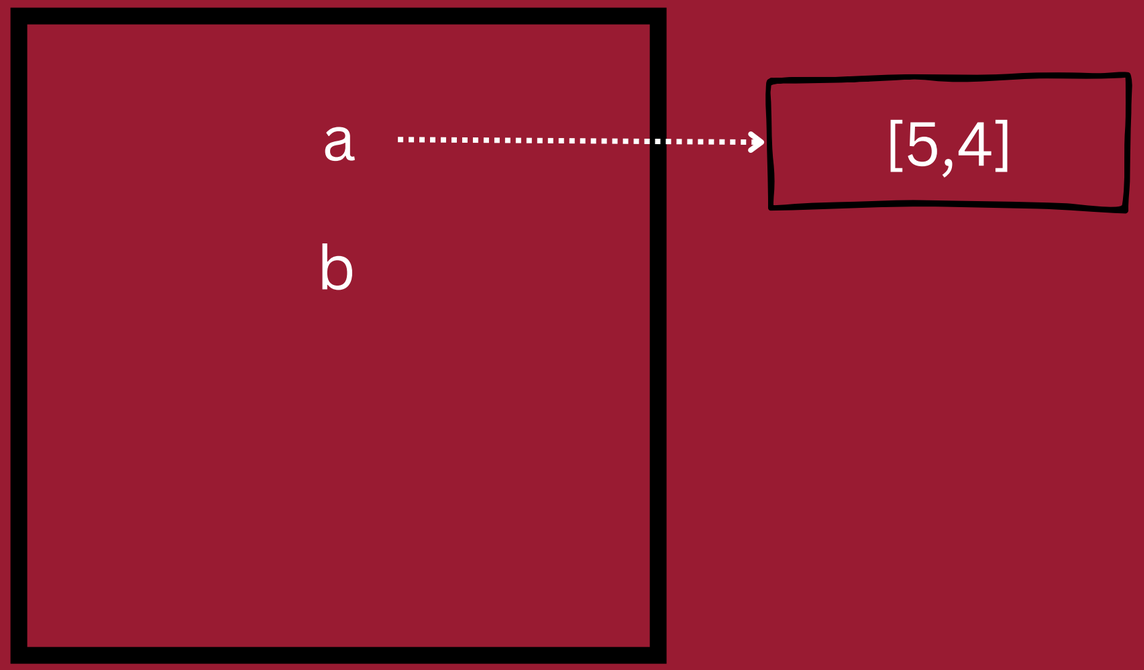
Now if we had
let a = [5,4];
let b = a;
a[0]=10;



what will this look like?

REFERENCES

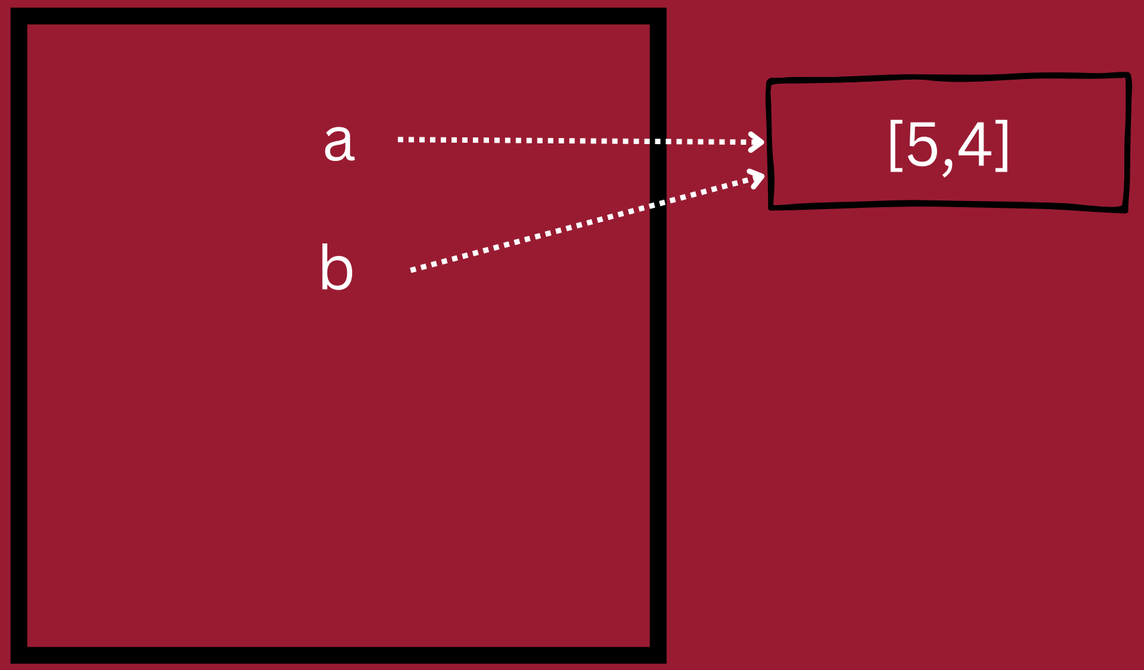
Now if we had
let a = [5,4];
let b = a;
a[0]=10;



What is b?

REFERENCES

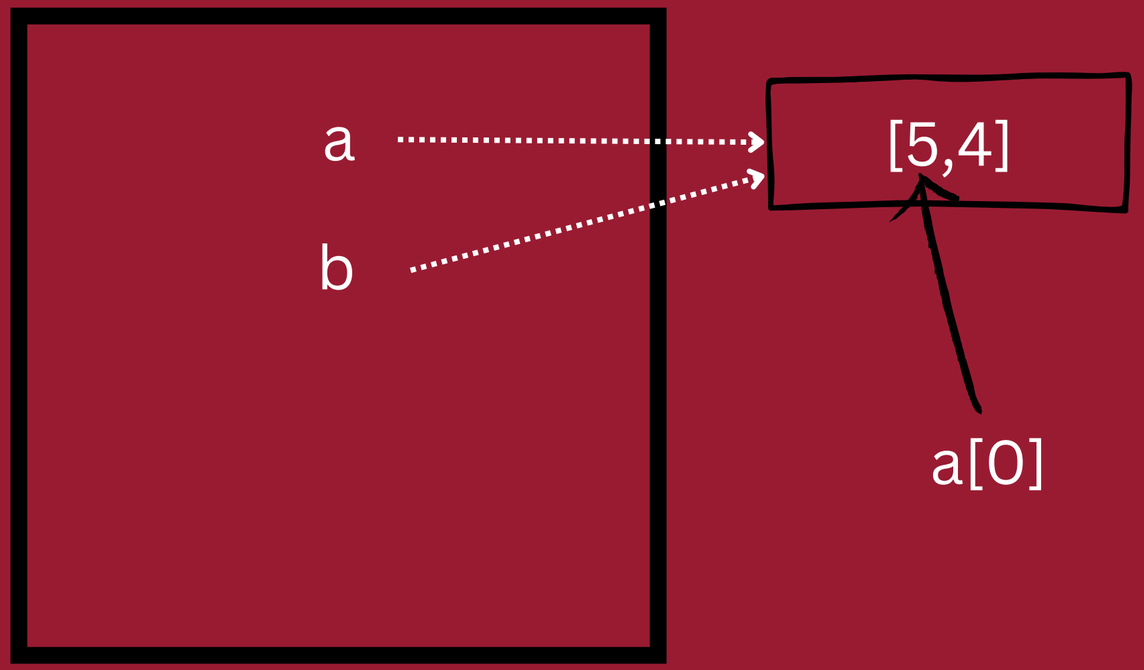
Now if we had
let a = [5,4];
let b = a;
a[0]=10;



What is b?

REFERENCES

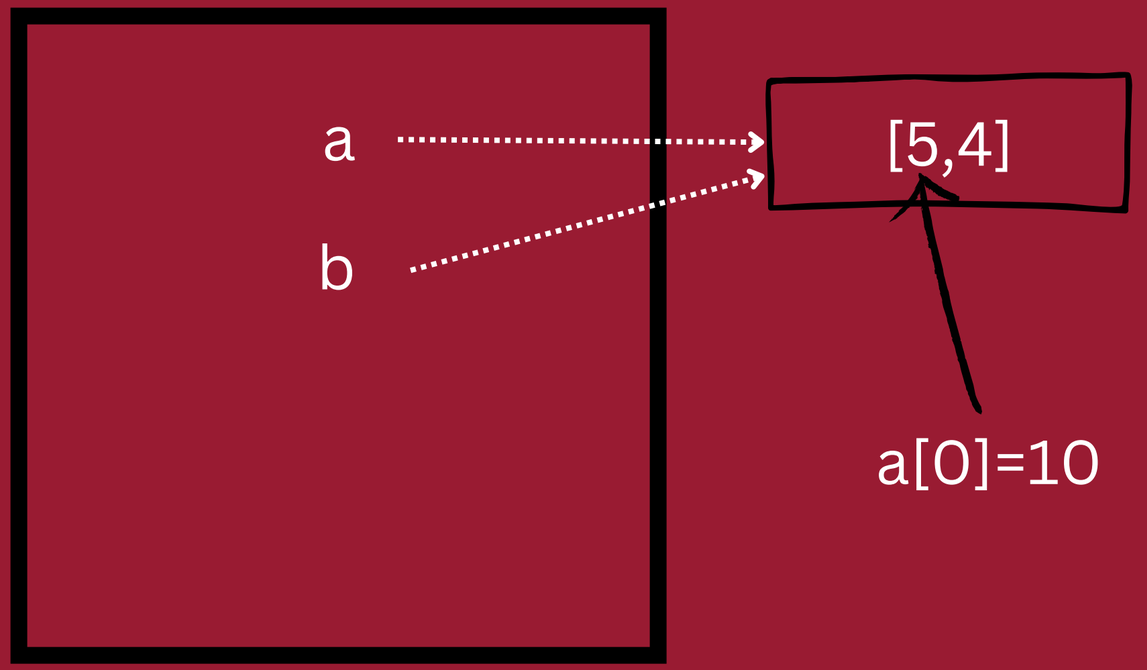
Now if we had
let a = [5,4];
let b = a;
a[0]=10;



What is b?

REFERENCES

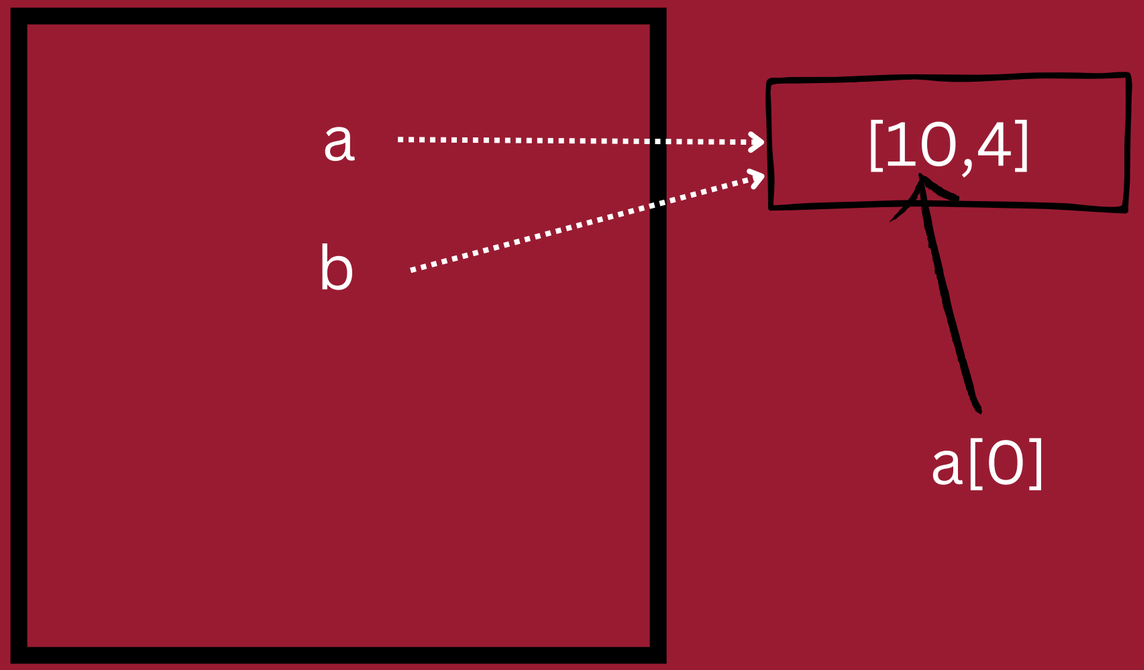
Now if we had
let a = [5,4];
let b = a;
a[0]=10;



What is b?

REFERENCES

Now if we had
let a = [5,4];
let b = a;
a[0]=10;

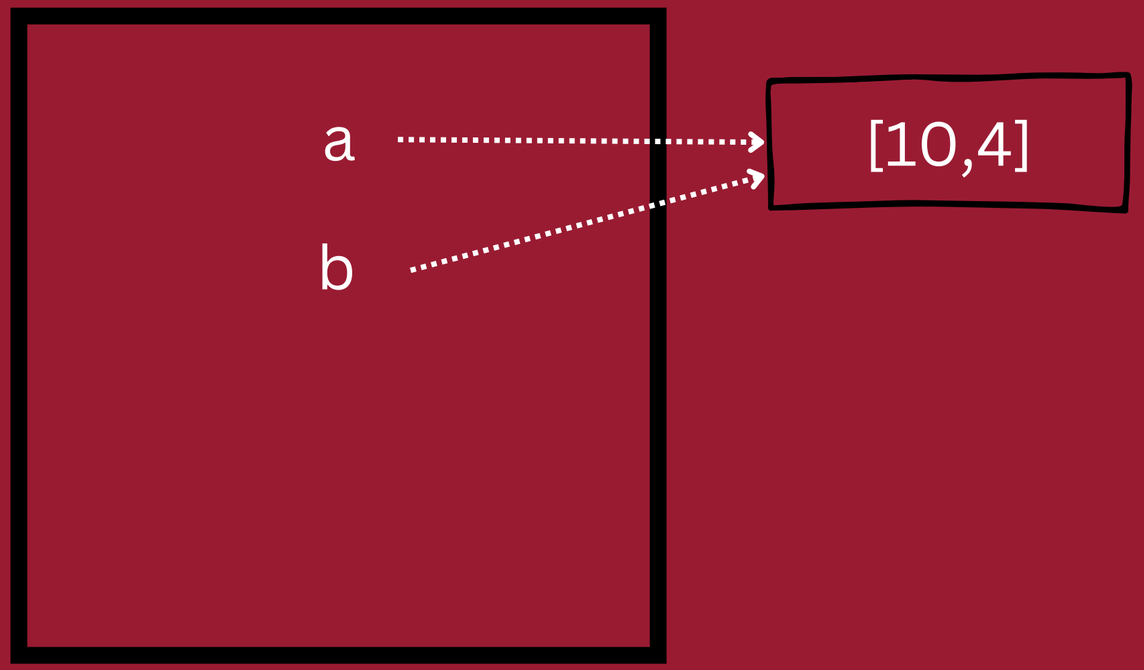


What is b?

REFERENCES

Now if we had

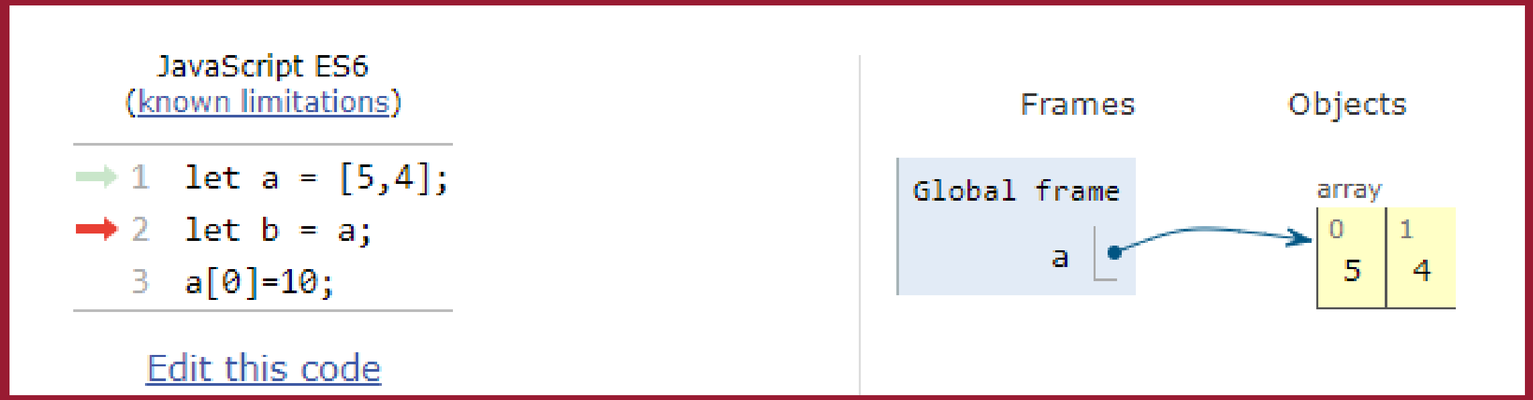
```
let a = [5,4];
let b = a;
a[0]=10;
```



b changes as well!

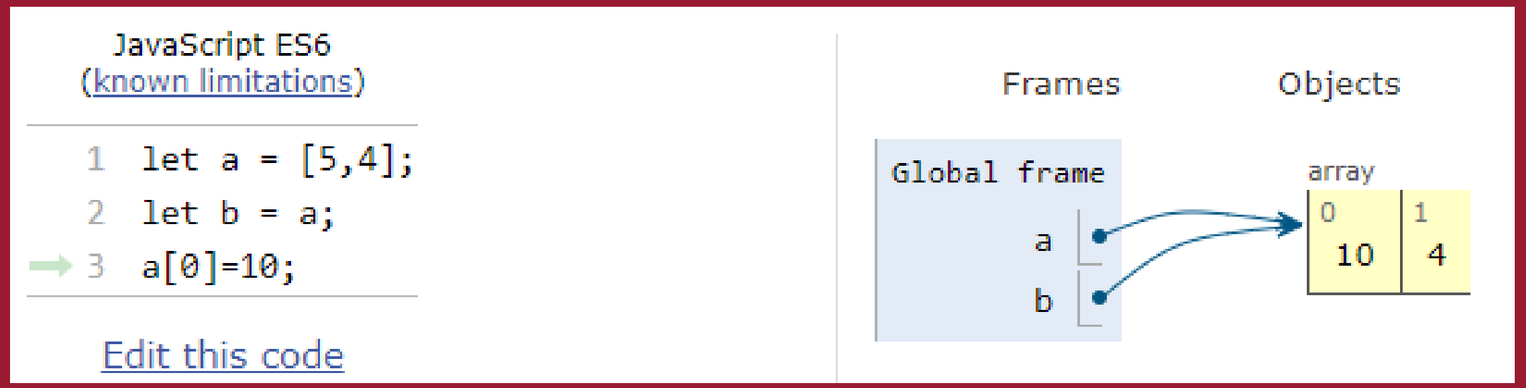
LET'S VISUALIZE

```
let a = [5,4];
let b = a;
a[0]=10;
```



LET'S VISUALIZE

```
let a = [5,4];
let b = a;
a[0]=10;
```



WHAT ABOUT THIS?

```
const a = {person:'Kirat',year:'junior'};
let b = a;
a.year='2024';
```

What happens here?

**if you are reading the slides try this yourself at pytutor.com*
don't forget to change language to jsES6

ODDITIES

Now, for some fun

ODDITIES

Find more on:

https://github.com/denysdovhan/wtfjs


JS
FOREVER