JAVASCRIPT 2023

@coderdost

https://www.youtube.com/@coderdost

1. JavaScript Basics

Weakly Typed Language

```
let name = “abhishek” ;

let object=   {name: “abhishek”} ;
```

Strongly Typed Language

```
Integer num = 1 ;

Cat cat = Cat() ;
```

Strongly Typed Language C C++ Java

Attaching JS

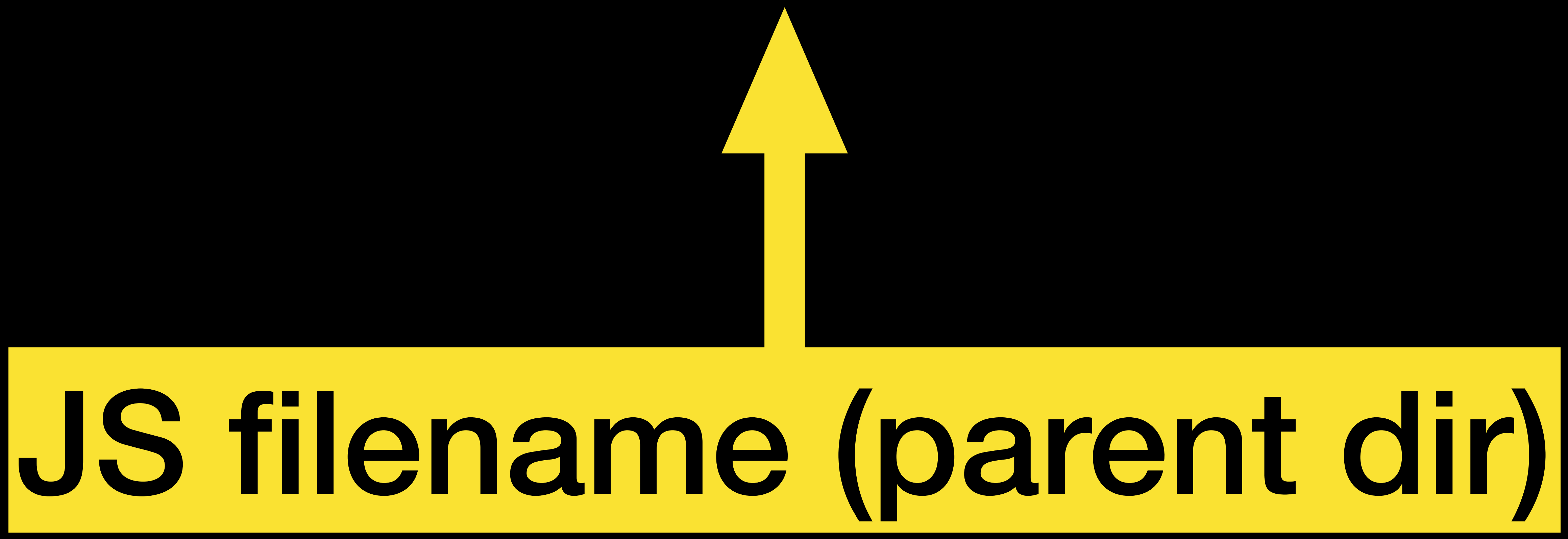
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <script src="index.js"></script>
</head>
<body>

</body>
</html>
```

JS filename (same dir)

index.html

Attaching JS

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <script src=“../index.js”></script>
</head>
<body>

</body>
</html>
```



JS filename (parent dir)

index.html

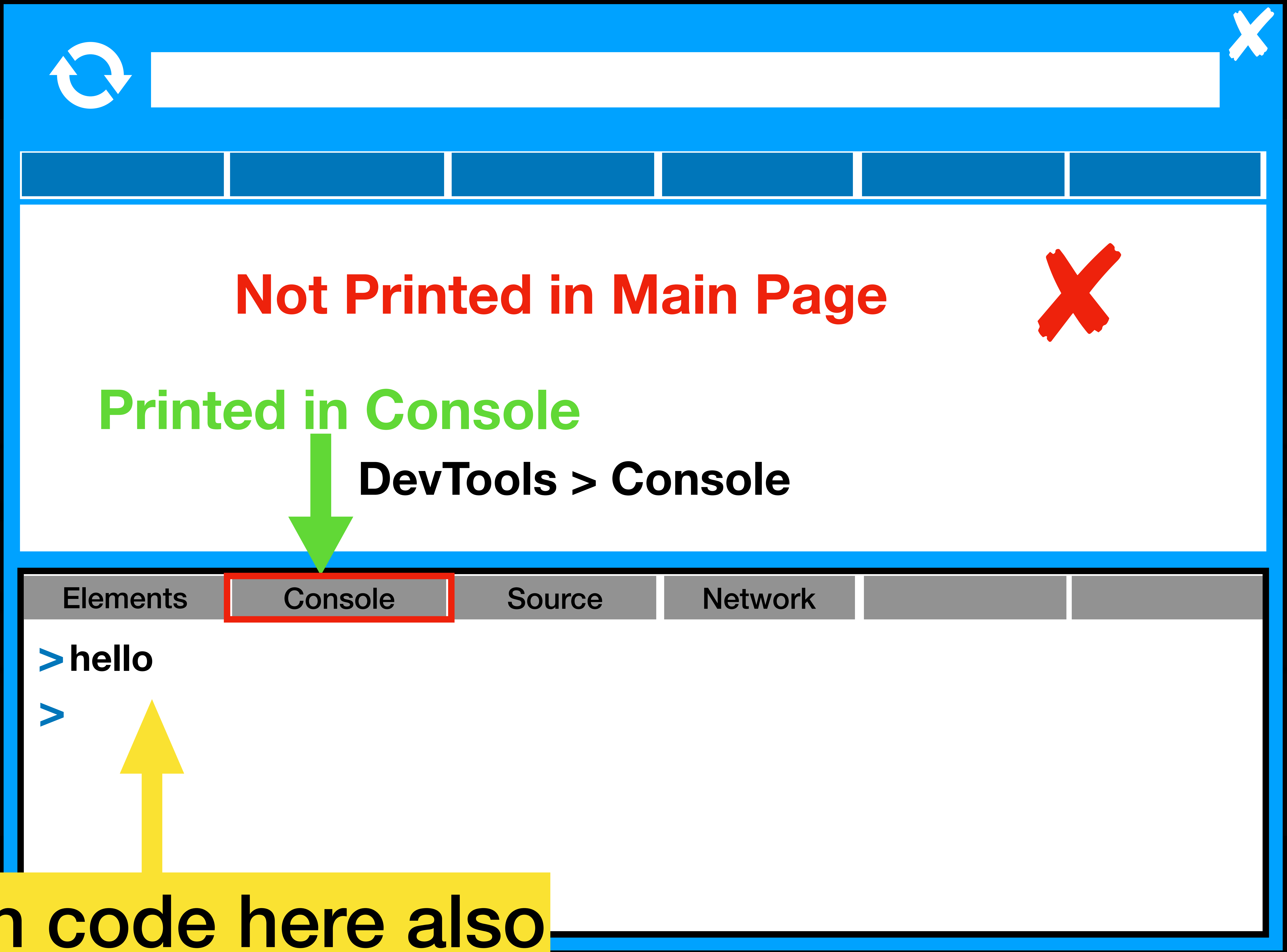
https://www.youtube.com/@coderdost

Print Statement of JavaScript

```
console.log(“hello”);
```

index.js

prints hello



Not Printed in Main Page ✗

Printed in Console

DevTools > Console

```
>hello
>
```

you can code here also

JavaScript Variables

```
let name ;
```

variable declaration

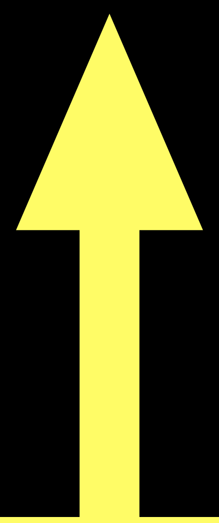
SemiColon

semicolon are optional

```
let name ;
```

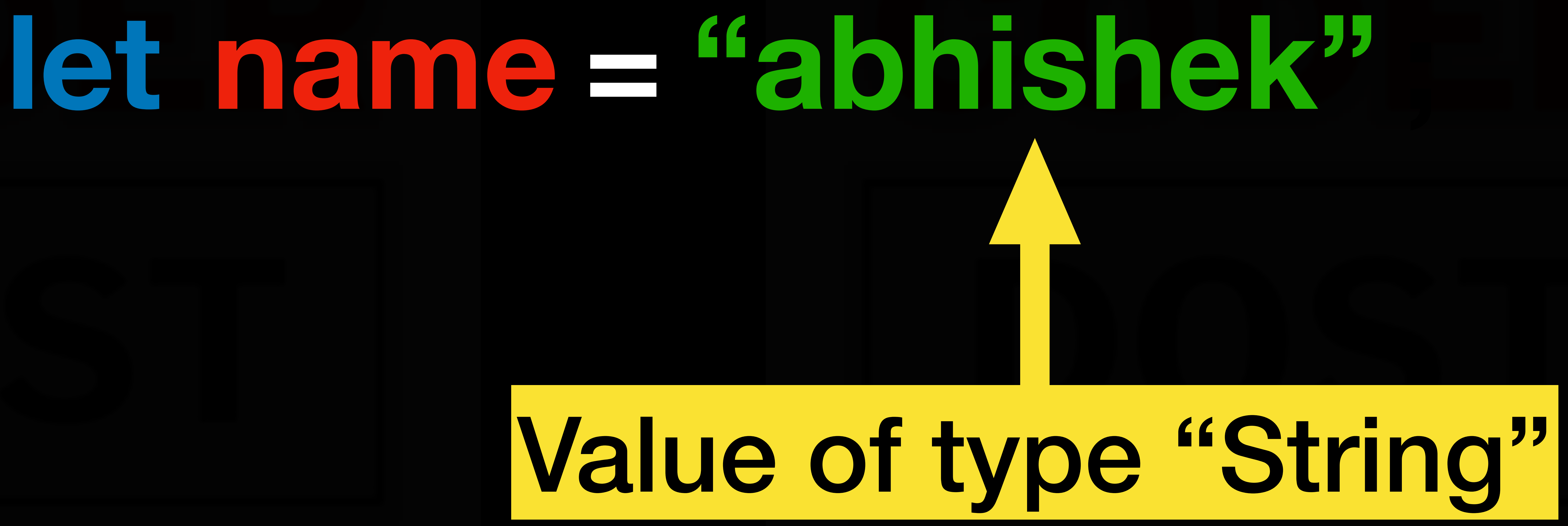
JavaScript Variables

```
let name = “abhishek” ;
```



assignment operator

JavaScript Variables



```
let name = “abhishek”
```

Value of type “String”

JavaScript Variables

```
let name = “abhishek”
```

when assigned at declaration time we call it “initialisation”

Data Types : 1. Number

```
let name =        20
```

Number

Data Types : 1. Number

let name = 20.66

Number

Data Types : 2. String

```
let name = “abhishek”
```

String

Data Types : 3. Boolean

```
let name =     false
```

Boolean

Data Types : 4. Object

```
let person = {name: ‘abhishek’} ;
```

Object

Data Types : 5. Array*

```
let numbers = [3,11,18,4,40,25] ;
```

Array

* Array is Object only. But a Special Kind of Object

6. Undefined Type

```
let name ;
```

if nothing is assigned - value is “undefined”

7. Null Type

```
let name = null
```

null is also a special “object”

Printing in JS

```
console.log( name )
```

Value of name will be printed

No quotes = variable

Printing in JS

```
console.log(“name”)
```

“name” will be printed

quotes = String

VAR vs LET vs CONST

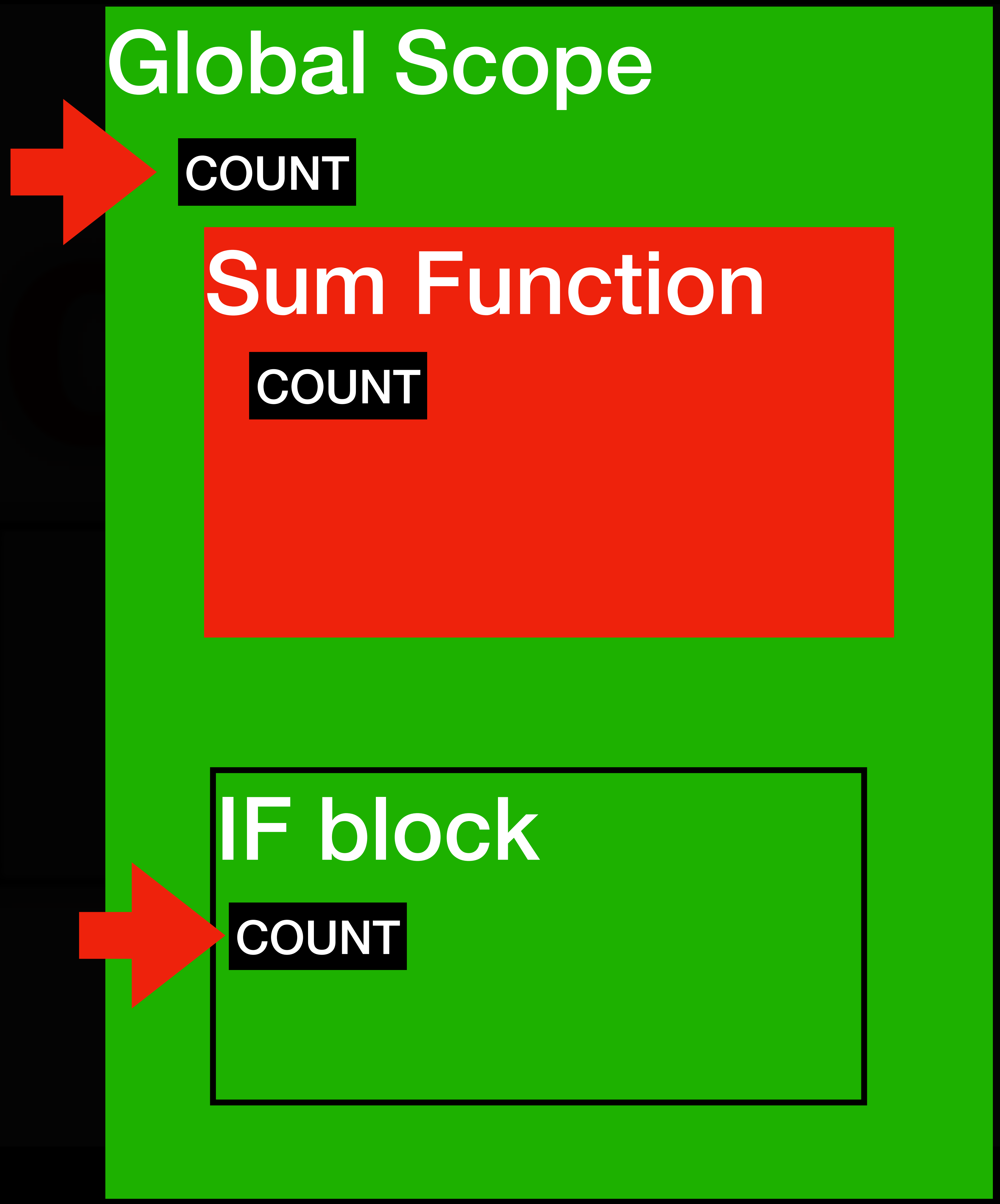
var never use it (old style, creates error)

let when you need to re-assign values,
may or may not be initialised at declaration

const when you never want to re-assign
, also always initialised at declaration

Scope of VAR

```
 var count = 1;

function sum(a, b, c){
    var count = 0;
    return a + b + c;
}

if(age>18){
    var count = 2;
    console.log(count)
 }
```



https://www.youtube.com/@coderdost

Scope of VAR

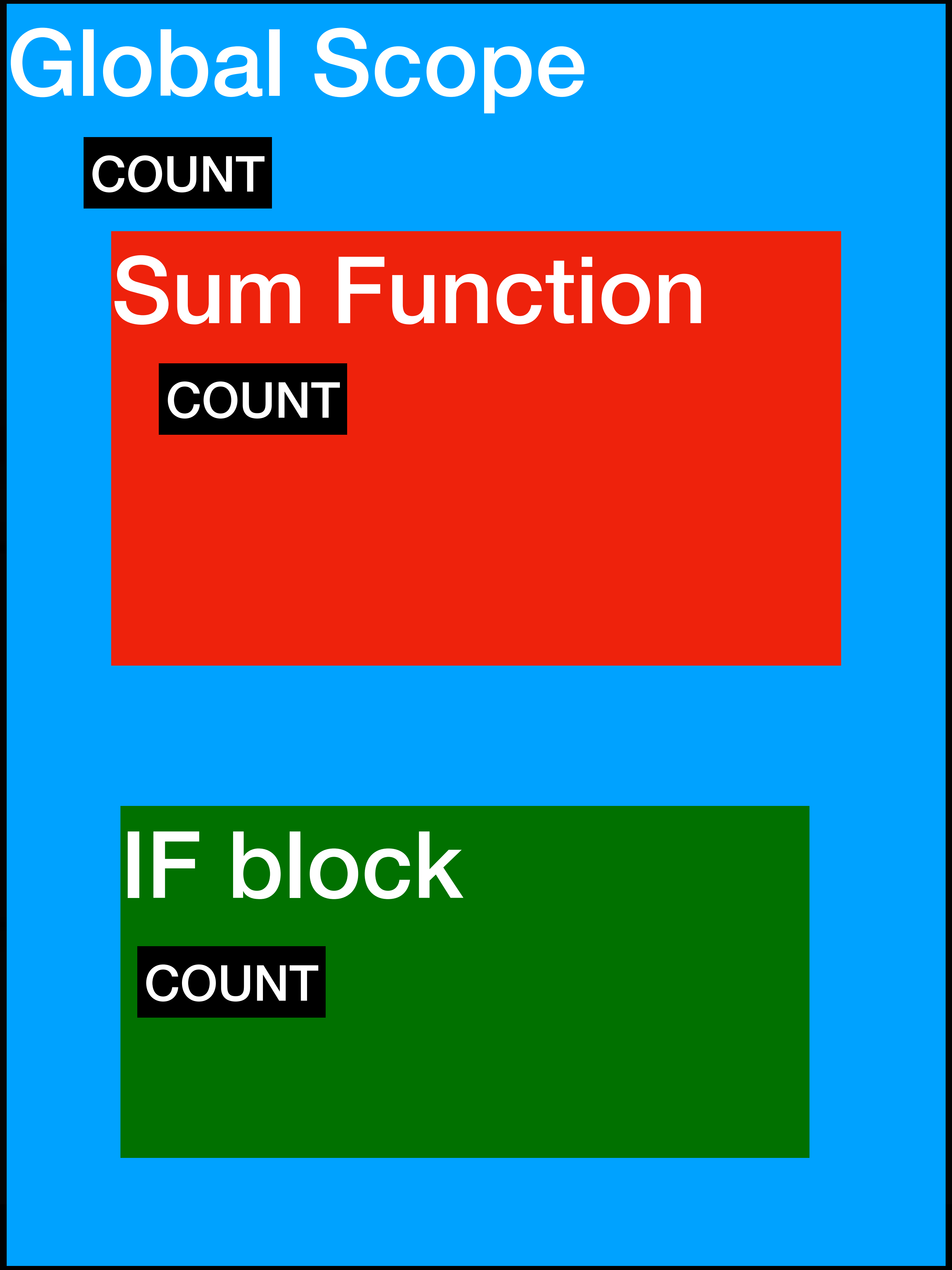
```
var count = 1;

function sum(a, b, c){
   var count = 0;
   return a + b + c;
}

if(age>18){
    var count = 2;
    console.log(count)
 }
```

Only Function Blocks creates new Scope with Var

https://www.youtube.com/@coderdost

Scope of Variables (let)

```
 let count = 1;

function sum(a, b, c){
   let count = 0;
   return a + b + c;
}

if(age>18){
   let count = 2;
   console.log(count)
}
```



Global Scope

COUNT

Sum Function

COUNT

IF block

COUNT

VAR v/s LET

VAR	LET
No block { } scope is created	All block { } have separate scope
Can be re-declared	Only declared once in scope
✗	✓

Const

```
const count = 1;
```

✗

```
count = 4;
```

ERROR

```
const person = {};
```

✗

```
person = anotherPerson;
```

ERROR

NO Re-assignment

Const

```
   const person = {};
✓ person.name = “abhishek”;
   const cities = [];
✓ cities.push(“mumbai”);
```

this works as “person” is not re-assigned

Some Instruction for Slides

➡ sign will represent **Return value**

camelCase javascript prefers camel case in variable names.

UpperCamelCase Some variables like Class will use upper camel case.

String Literals : Style 1

let title = “hi” ;

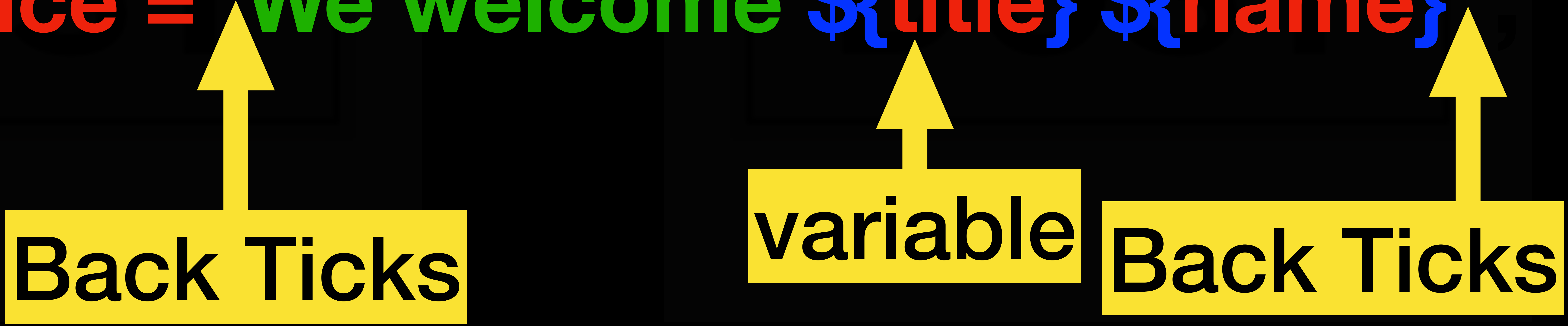
let name = “raj”;

Concat title + name → hiraj

title + “ ” + name → hi raj

String Literals : Style 2 (template)

```
let  title = “mr” ;
let  name = “raj”;

let  sentence = `We welcome ${title} ${name}` ;
```



Back Ticks

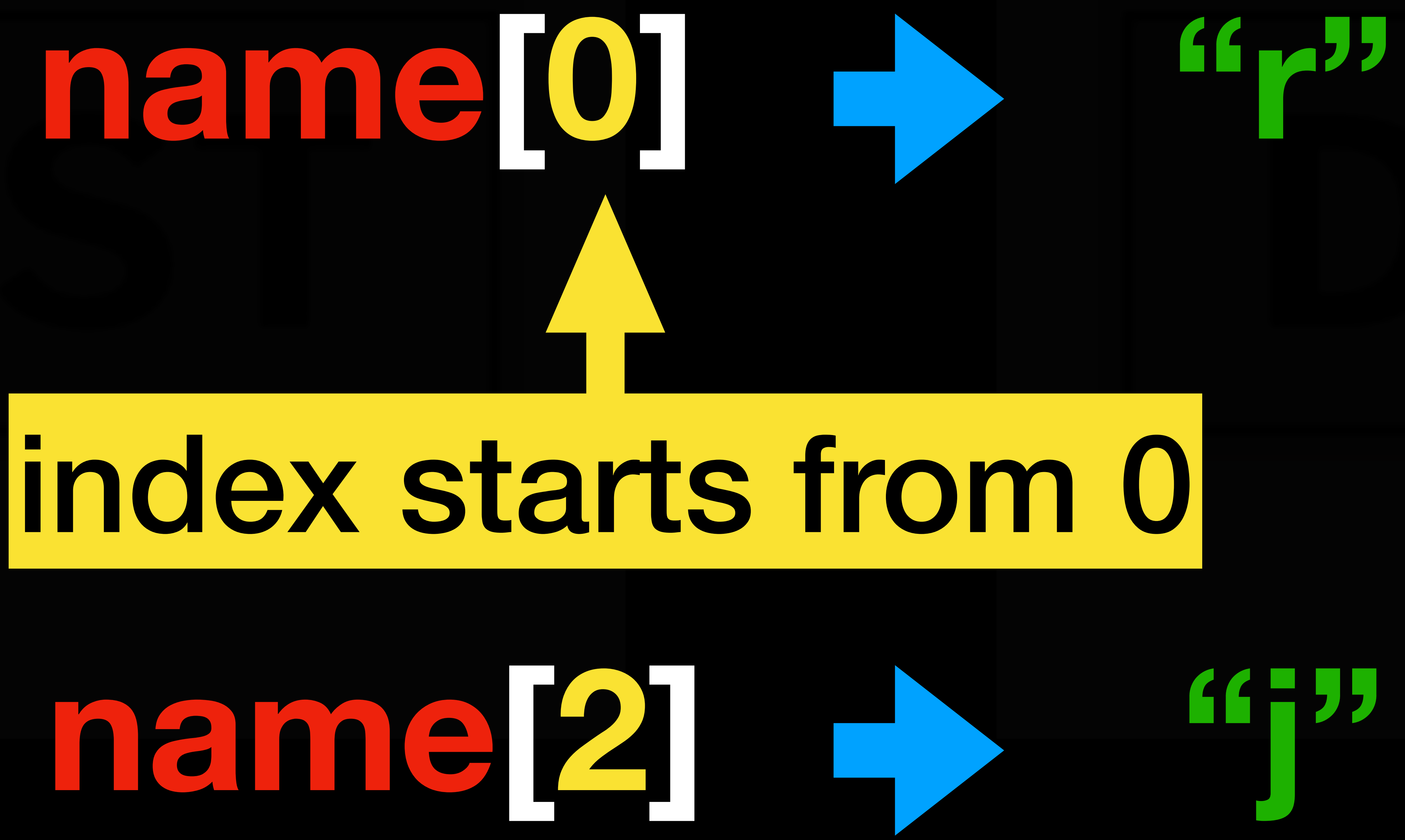
variable

Back Ticks

https://www.youtube.com/@coderdost

String : Access Character by Index

```
let  name = “raj”;
```



name[0] → “r”

index starts from 0

name[2] → “j”

https://www.youtube.com/@coderdost

String : length property

1 space also

```
let words = “Hello World”
```

```
words.length → 11
```

* What is a property ?? we will explain later

String Method* : upperCase / lowerCase

```
let  words = “Hello”

words.toUpperCase()    →    “HELLO”

words.toLowerCase()    →    “hello”
```

* What is a Method ?? we will explain later

https://www.youtube.com/@coderdost

String Method : indexOf

```
let  words = “Hello”

words.indexOf(‘e')        →        1
words.indexOf(‘z’)        →       -1
```

Mutating vs Non-Mutating methods

Mutating

changes variable which called it

example

array.push()

Non-Mutating

doesn’t changes the variable which called it

example

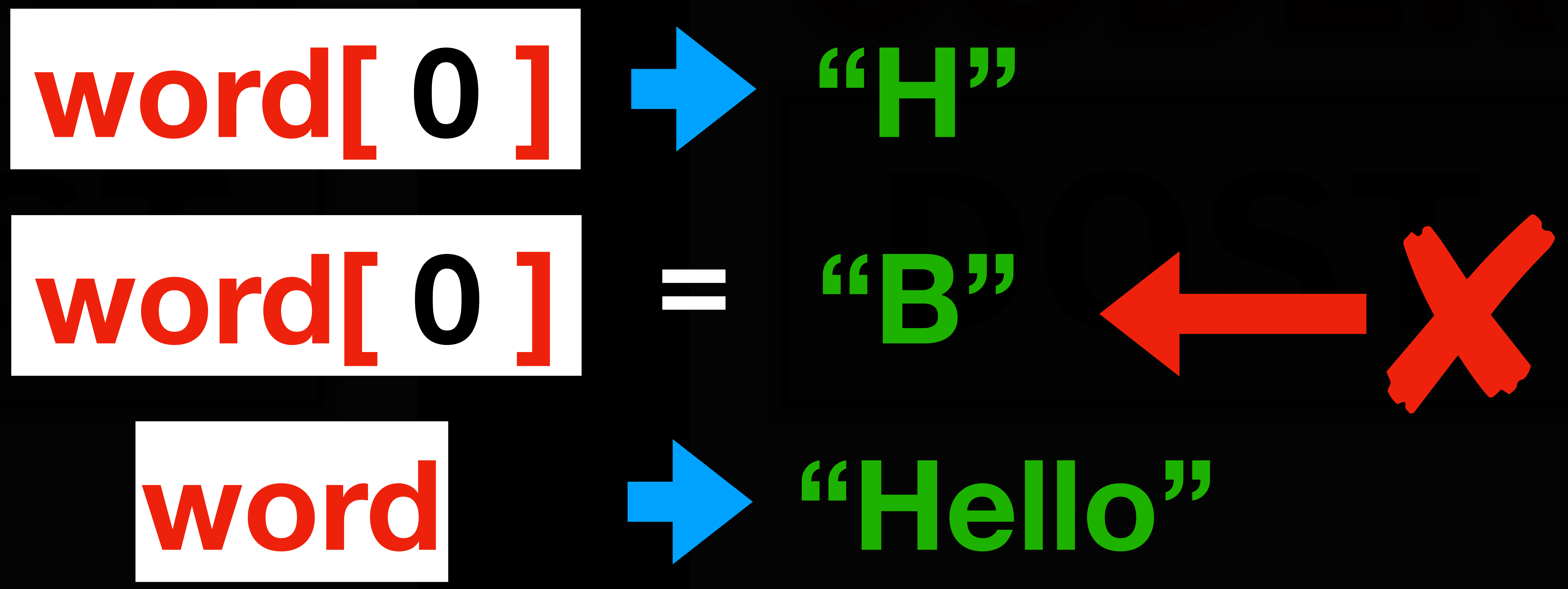
.indexOf()

*** There are no Mutating methods on String => String are Immutable**

https://www.youtube.com/@coderdost

Immutability of String

var word = “Hello”



word[0] → “H”

word[0] = “B” ← ✗

word → “Hello”

String can’t be modified once initialised. Only a new string can be made

https://www.youtube.com/@coderdost

String Method : includes

```
let  words = “Hello”

words.includes(‘e’)   →   true

words.includes(‘z’)   →   false
```

not found

https://www.youtube.com/@coderdost

String Method : trim

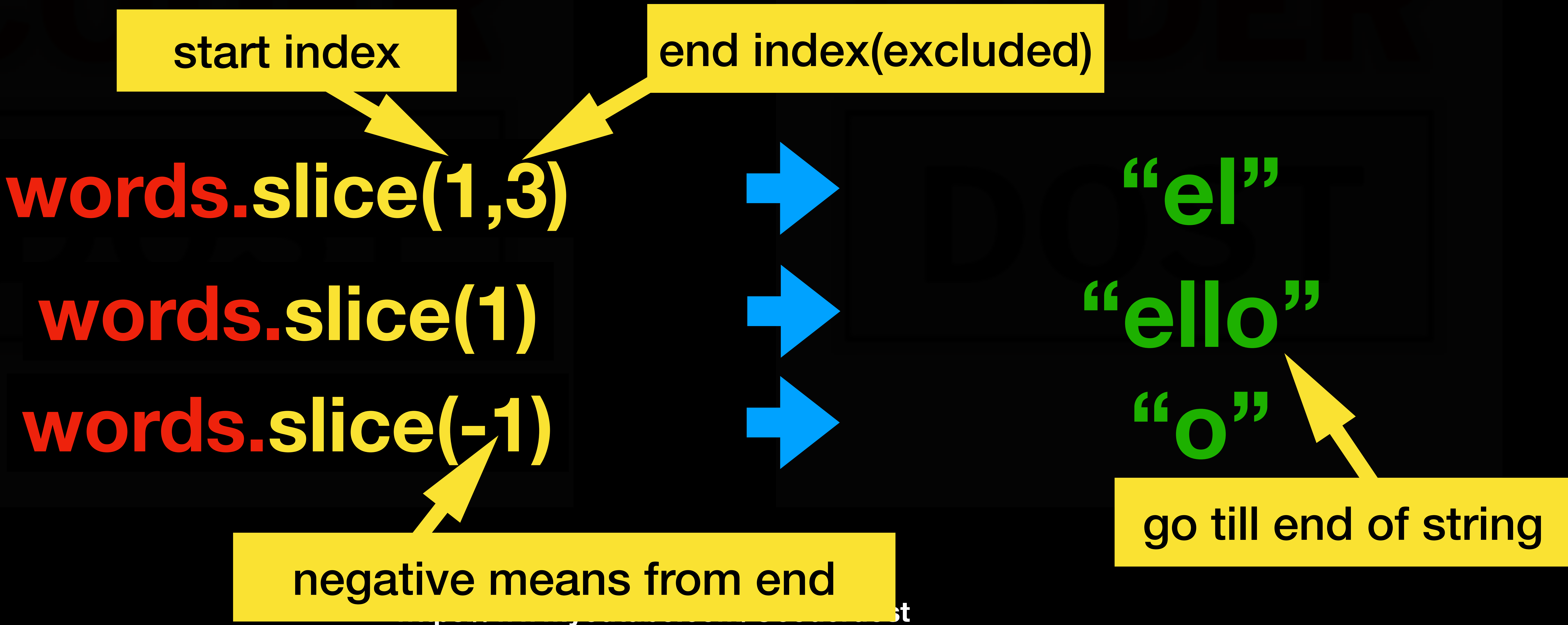
let words = “   Hello ”

remove white space at start & end

words.trim() → “Hello”

String Method : Slice

```
let words = “Hello”
```



```
words.slice(1,3)   →   “el”
words.slice(1)     →   “ello”
words.slice(-1)    →   “o”
```

start index

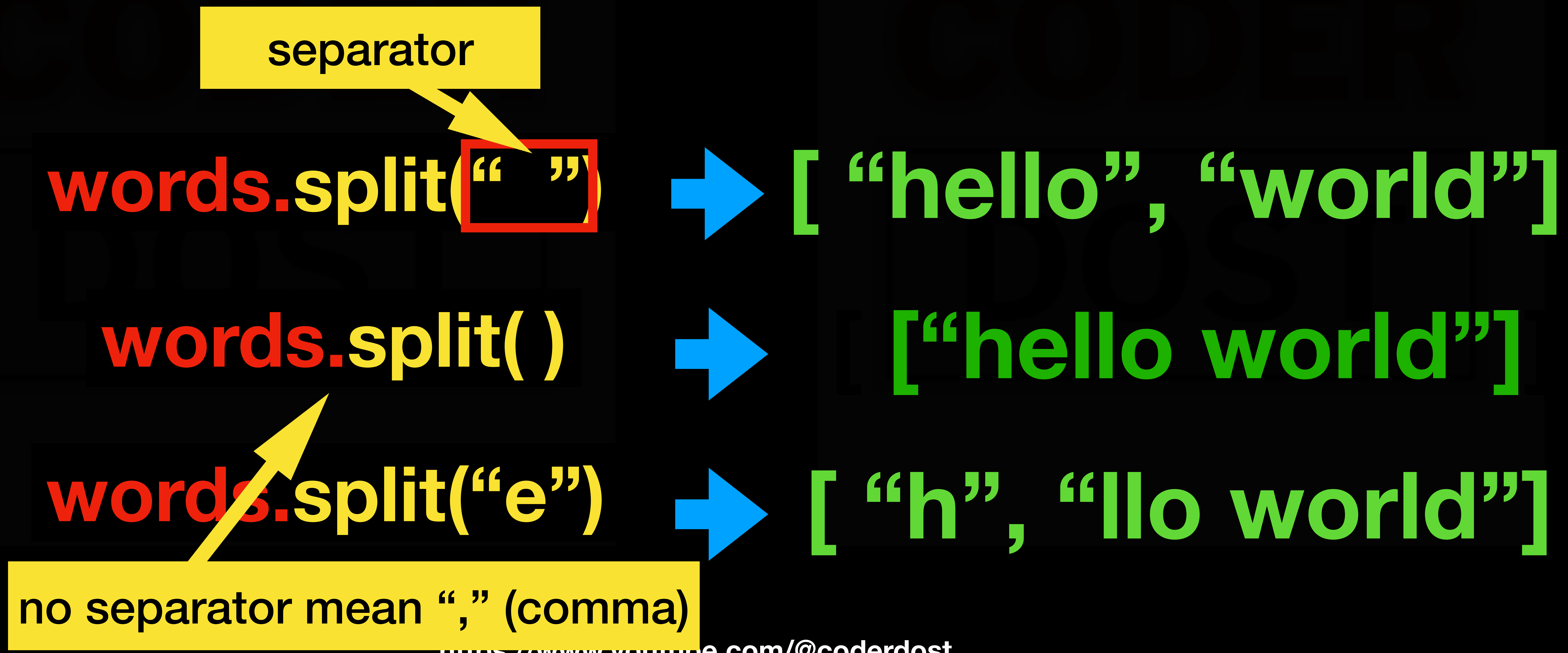
end index(excluded)

negative means from end

go till end of string

String Method : Split

let words = “hello world”



separator

words.split(“ ”) → [“hello”, “world”]

words.split( ) → [“hello world”]

words.split(“e”) → [“h”, “llo world”]

no separator mean “,” (comma)

String Method : Split

```
let  words = “hello”

words.split(“”)    → [ “h”,”e”,”l”,”l”,”o”]

words.split(“l”)   → [ “he”,””,”o”]

words.split(“o”)   → [ “hell”, “”]
```

typeof

```
let age = 20;
let name = “john”;
let address = {};
let cities = [];
let course;
```

`typeof age` → Number

`typeof name` → String

`typeof address` → Object

`typeof cities` → Object

`typeof course` → undefined

Arithmetic Operations

```
let a = 5    let b = 6 ;
```

Sum a + b → 11

Diff a - b → -1

Multiply a * b → 30

Divide a / b → 0.8333

Modulo a % b → 5

https://www.youtube.com/@coderdost

Arithmetic Operations : Precedence

```
let a =  6/6 + 2*7 + (7-2)*8 → 55
```

Brackets	( )	First priority
Power	**	
Multiply / Divide / Modulo	* / %	
Add / Subtract	+ -	Last priority

In case of same priority - Left to Right evaluation happens

Arithmetic Operations

```
let a = 5 ;
```

Increment	a++	→	6
Increment	a+=2	→	7
Decrement	a--	→	4
Decrement	a-=2	→	3

All operation done to “a=5”

https://www.youtube.com/@coderdost

Logical Operations

```
let a = true    let  b = false ;
```

OR	a || b	→	true
AND	a && b	→	false
Equality	a == b	→	false
Non-equality	a != b	→	true

logical operation always return Boolean

Logical Operations

let a = 5 let b = 6 ;

Greater than	a>b	→	false
Less than	a<b	→	true
Greater than equal	a>=b	→	false
Less than equal	a<=b	→	true

https://www.youtube.com/@coderdost

Loose Equality (==)

```
let age = “20”;

if(age == 20){
    console.log("adult")
}
```

true

Strict Equality (===)

```
let age = “20”;

if(age === 20){
    console.log("adult")
}
```

true false

Type Coercion

```
let a = 5    let b = "6" ;
```

concat	a + b	→	"56"
Multiply	a*b	→	30
Subtract	a-b	→	-1

Type Coercion

let a = 5 let b = “hi” ;

Concat	a + b	→	“5hi”
Multiply	a*b	→	NaN
Subtract	a-b	→	NaN

NaN = Not a Number

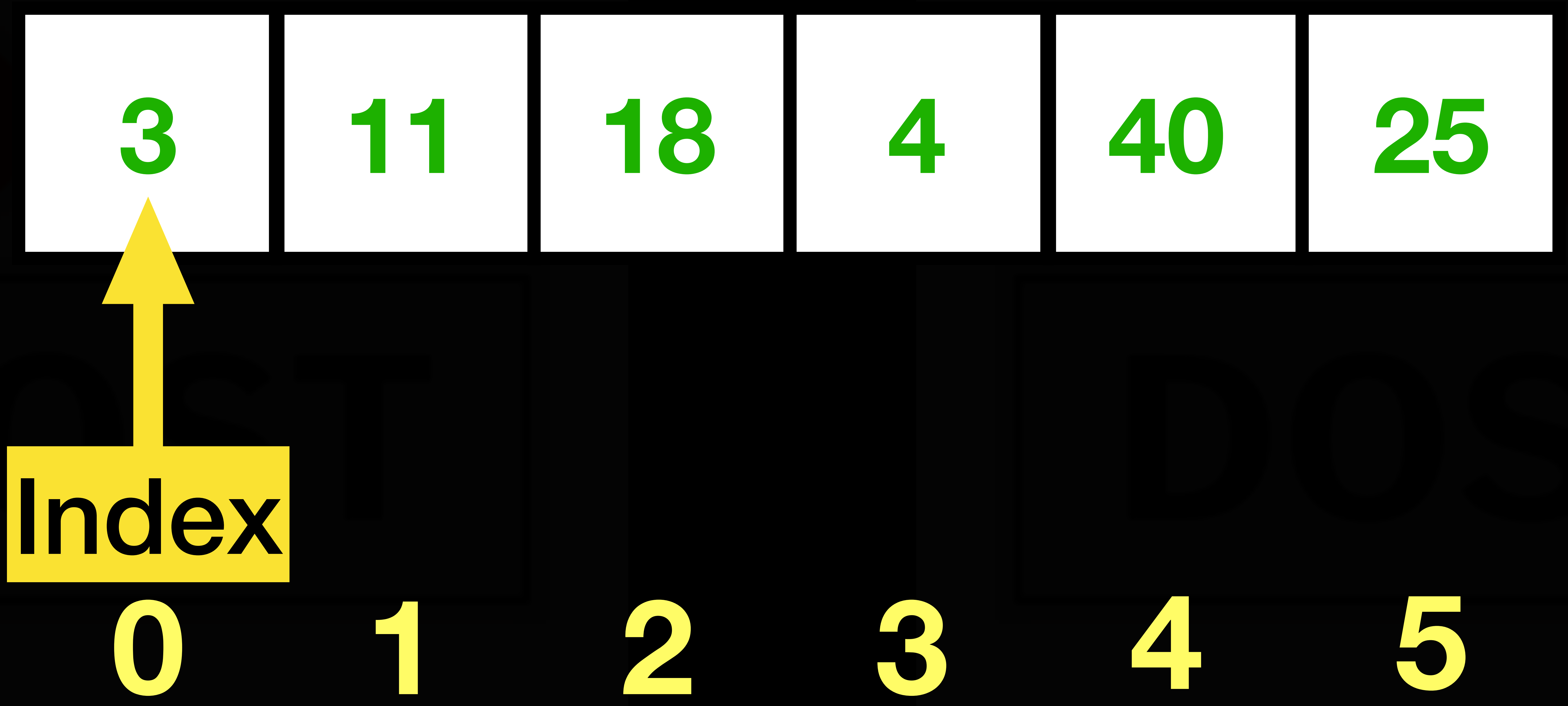
Type Conversion

```
let a = “5”  let  b = 6  ;
```

String to Number Number(a) → 5

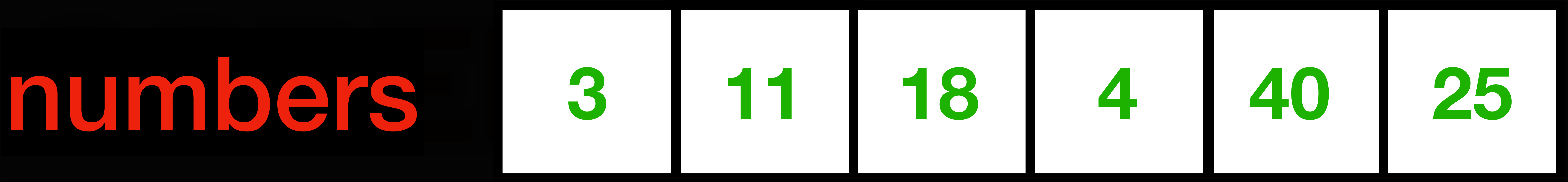
Number to String String(b) → “6”

Array



3	11	18	4	40	25
0	1	2	3	4	5

Initialising Array in JS



numbers

3	11	18	4	40	25

```
let numbers = [ ] ;
```

Empty Array

```
let numbers = [3,11,18,4,40,25] ;
```

Reading Array

numbers	3	11	18	4	40	25

numbers[0] → 3

numbers[4] → 40

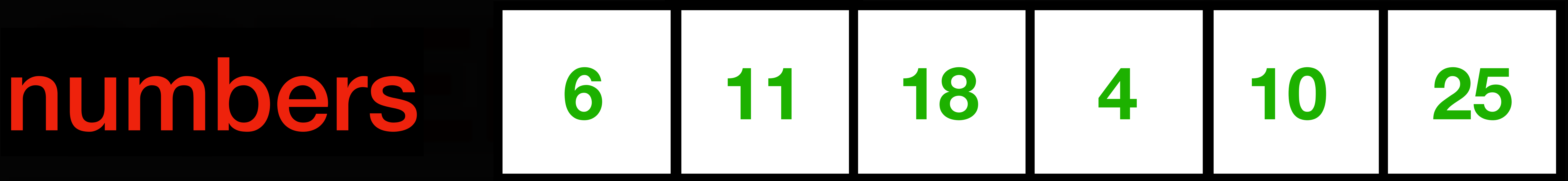
Writing Array

numbers	6	11	18	4	10	25

numbers[0] = 6

numbers[4] = 10

Array : length property



numbers.length → 6

Mutating vs Non-Mutating methods

Mutating

changes variable which called it

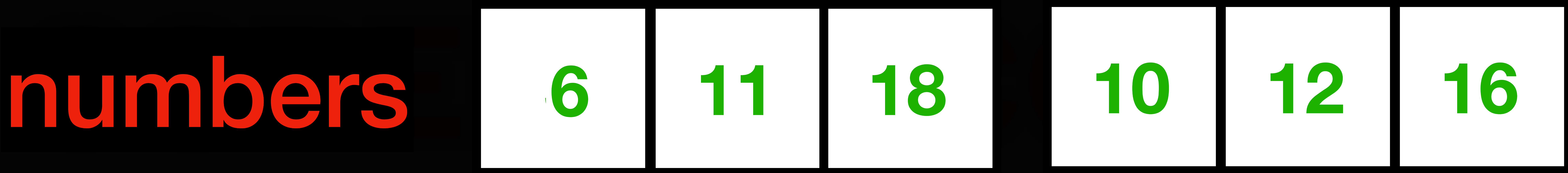
example
array.push()

Non-Mutating

doesn’t changes the variable which called it

example
array.indexOf()

PUSH function



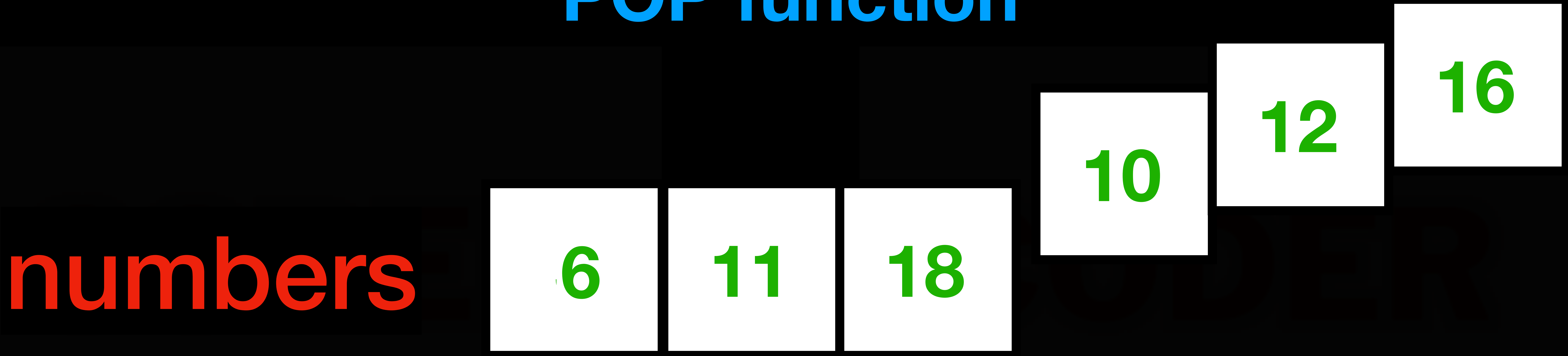
```
numbers.push(10) → 4
numbers.push(12) → 5
numbers.push(16) → 6
```

array length after push

Mutating Method

https://www.youtube.com/@coderdost

POP function



numbers.pop( ) → 16

numbers.pop( ) → 12

numbers.pop( ) → 10

Mutating Method

https://www.youtube.com/@coderdost

indexOf function

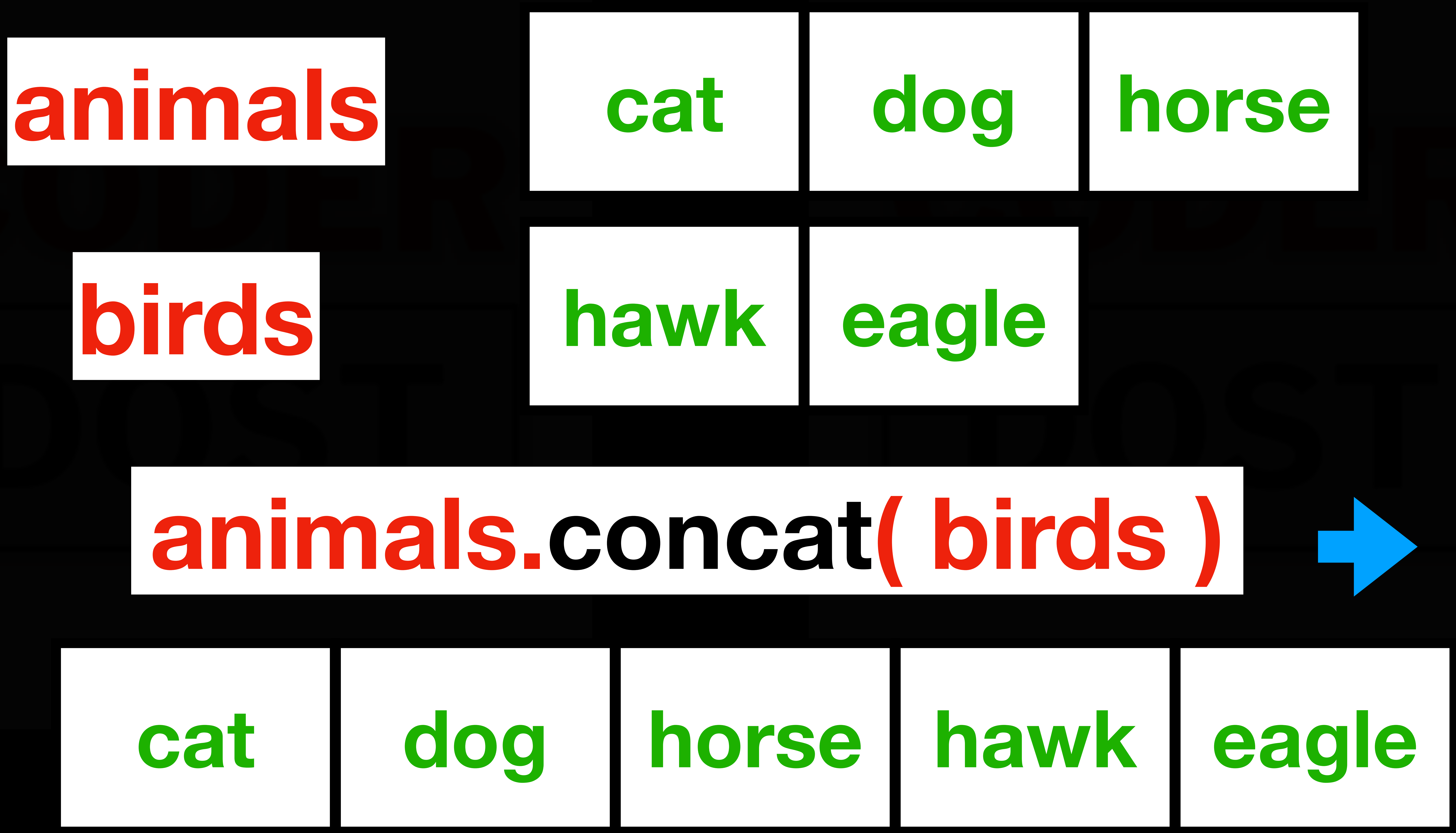
words	cat	dog	horse

words.indexOf(“cat”) → 0

words.indexOf(“fox”) → -1

Non-Mutating Method

CONCAT function



animals

cat dog horse

birds

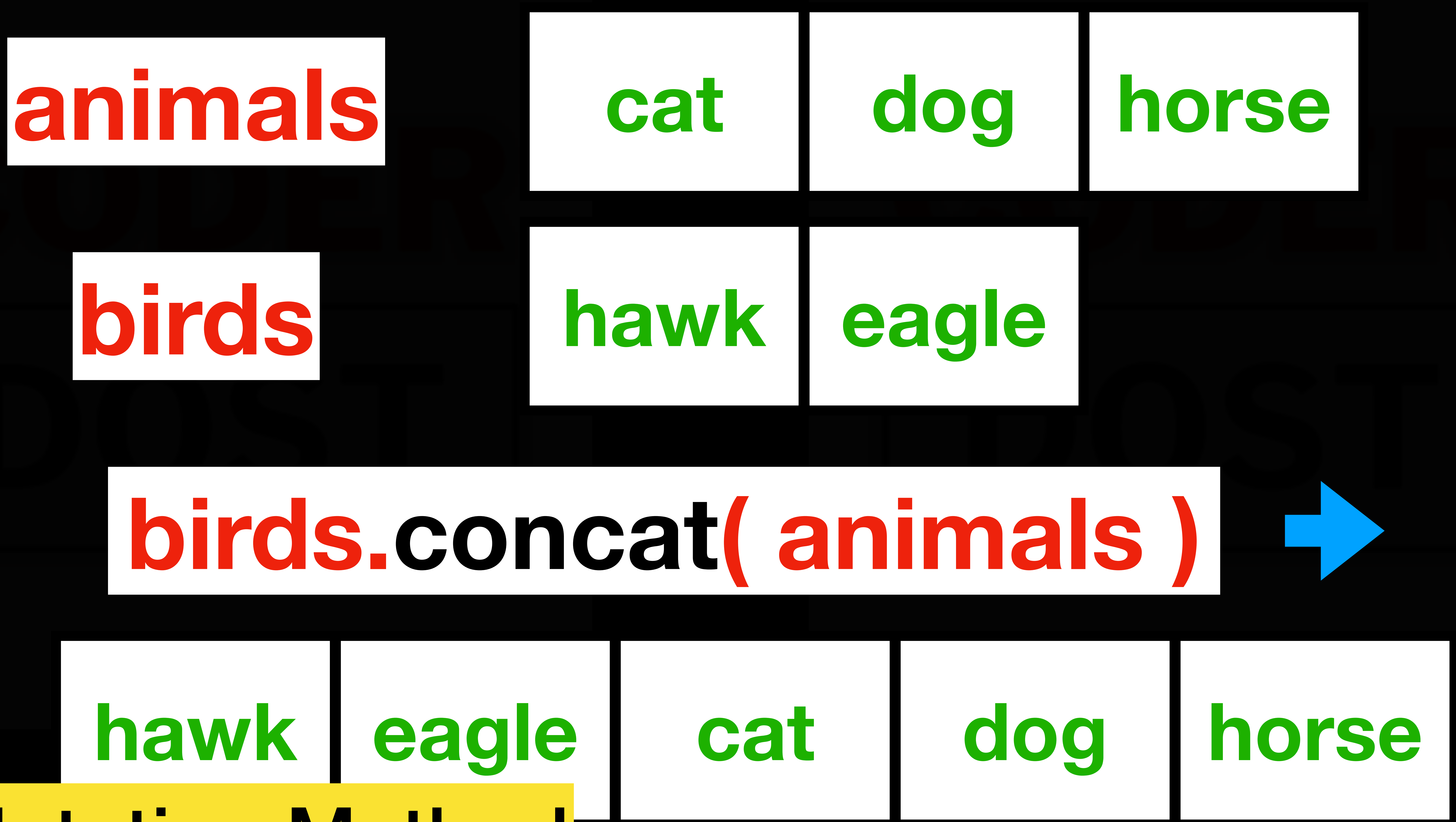
hawk eagle

animals.concat(birds)

cat dog horse hawk eagle

https://www.youtube.com/@coderdost

CONCAT function



animals

cat dog horse

birds

hawk eagle

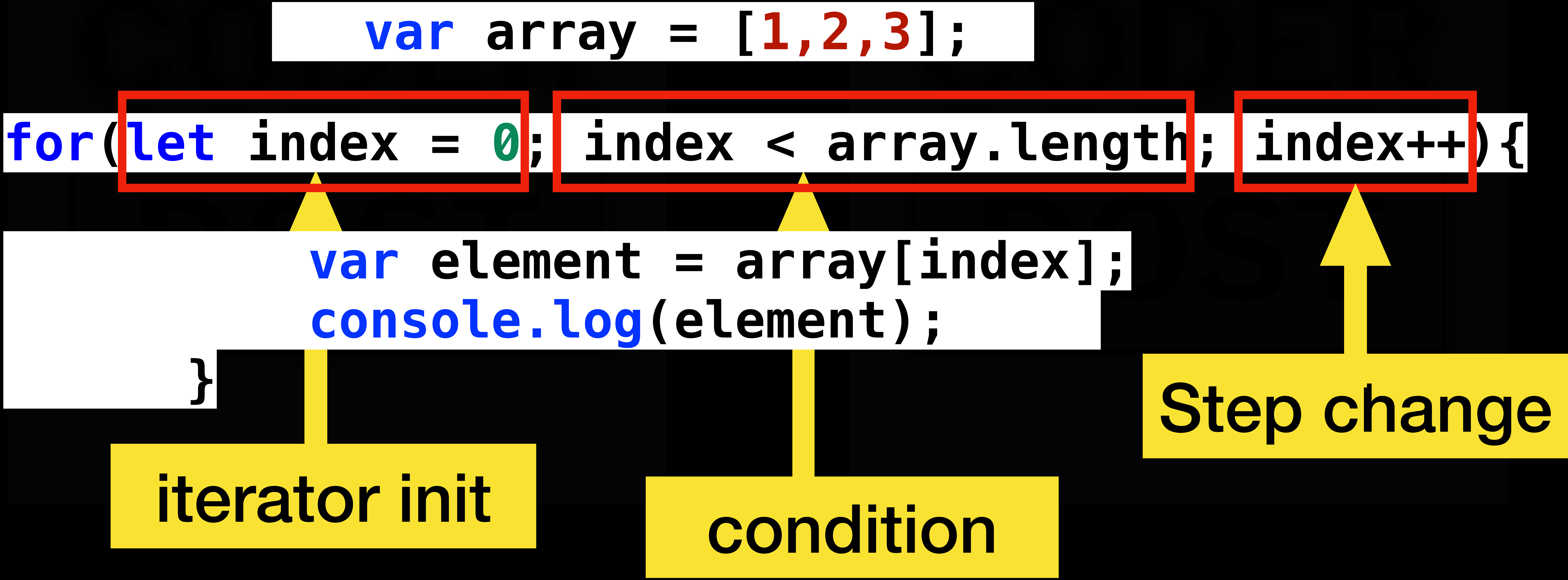
```
birds.concat( animals )
```

hawk eagle cat dog horse

Non-Mutating Method

2. Flow control

FOR Loop

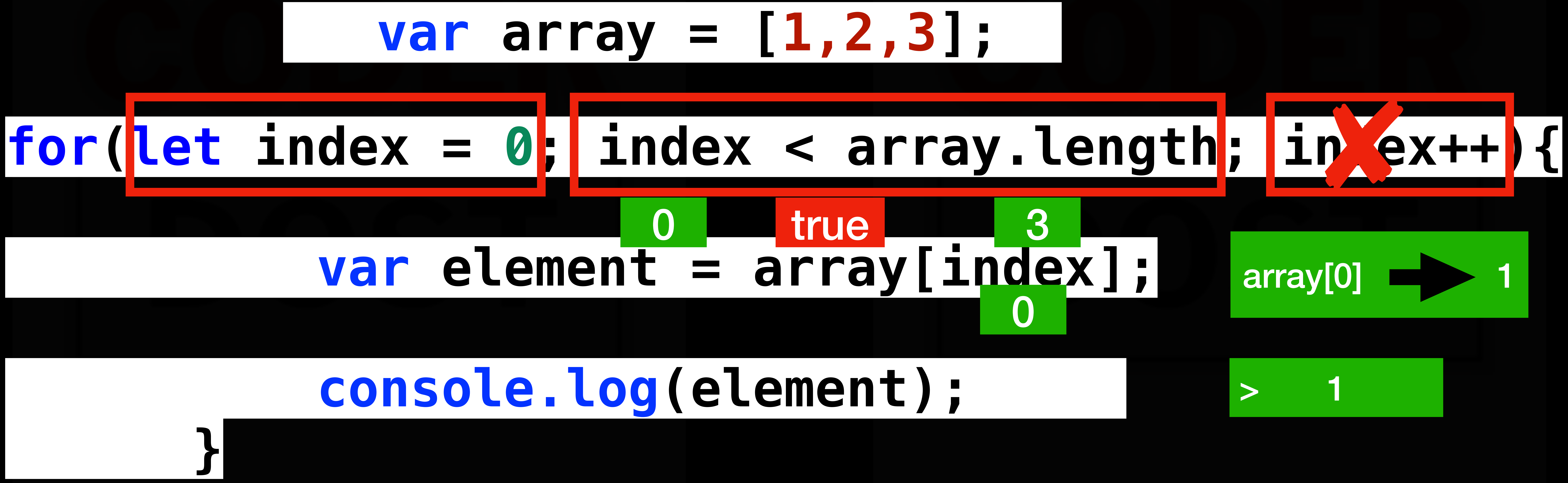


```
var array = [1,2,3];
for(let index = 0; index < array.length; index++){
    var element = array[index];
    console.log(element);
}
```

https://www.youtube.com/@coderdost

FOR Loop

ITERATION 1



```
var array = [1,2,3];
for(let index = 0; index < array.length; index++){
        var element = array[index];
        console.log(element);
}
```

https://www.youtube.com/@coderdost

FOR Loop

ITERATION 2

```
        var array = [1,2,3];
for(let index = 0; index < array.length; index++){
        var element = array[index];
        console.log(element);
    }
```

1 true 3

1

Index → 1

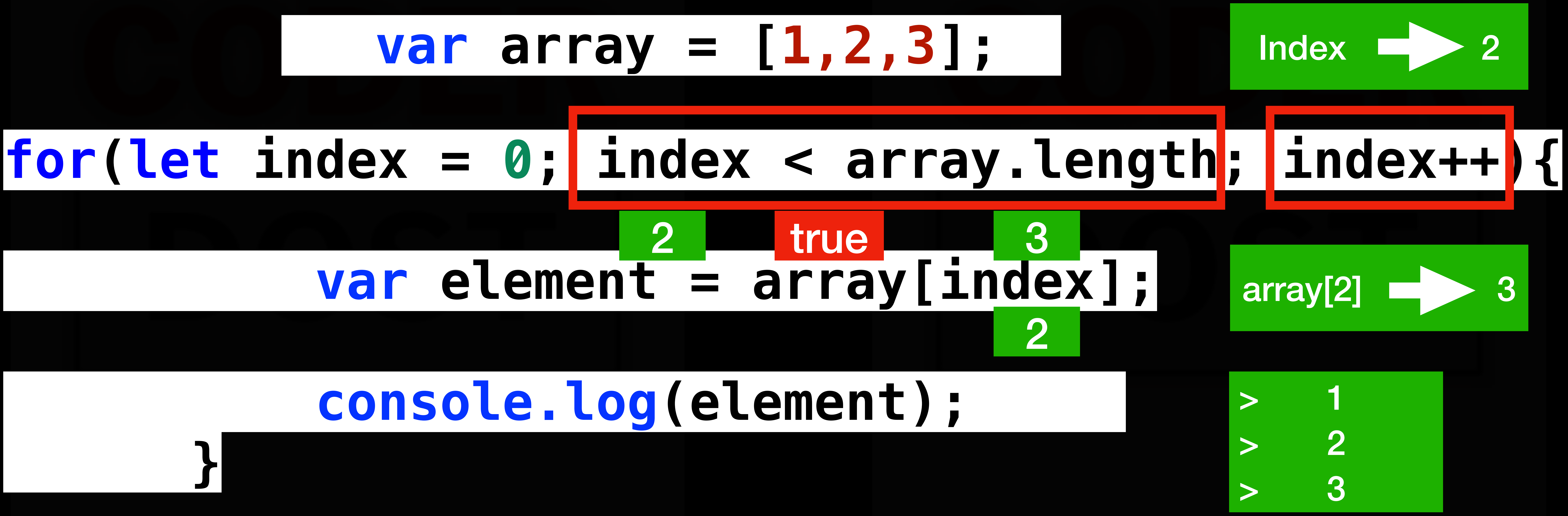
array[1] → 2

```
>    1
>    2
```

https://www.youtube.com/@coderdost

FOR Loop

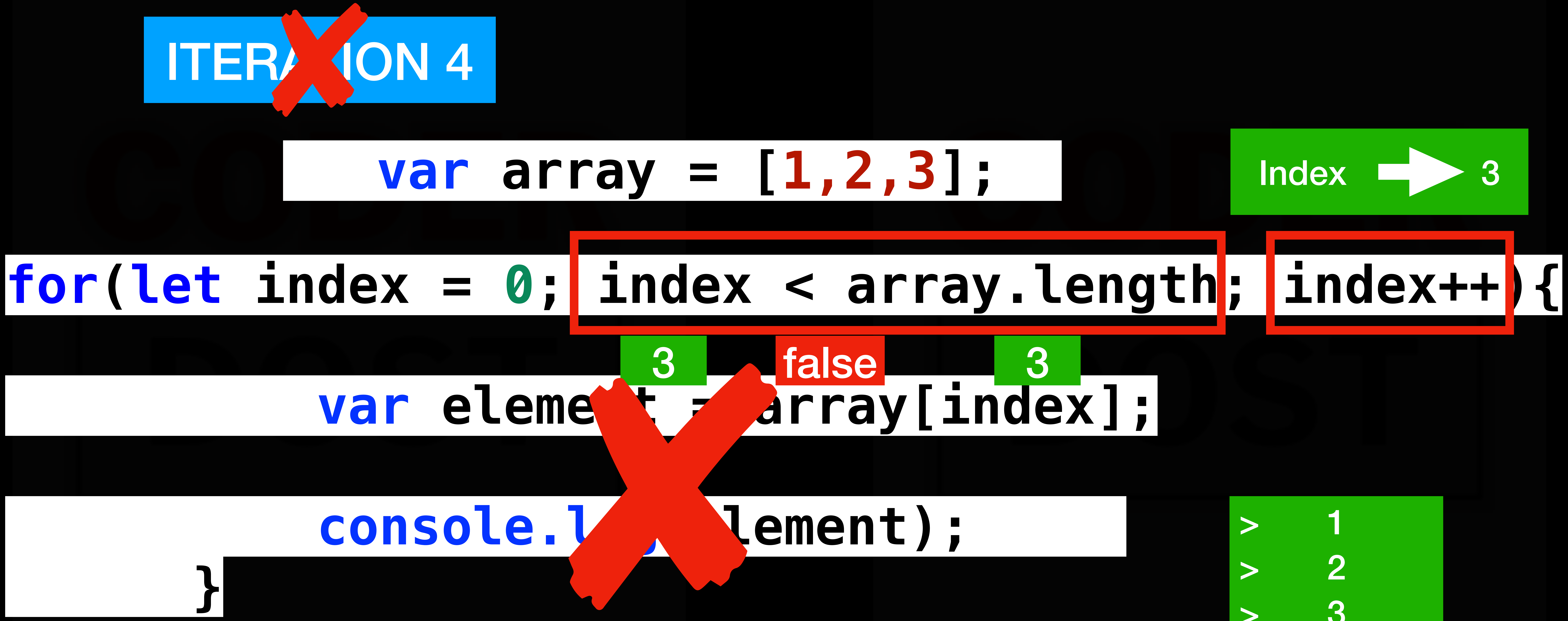
ITERATION 3



```
var array = [1,2,3];
for(let index = 0; index < array.length; index++){
    var element = array[index];
    console.log(element);
}
```

https://www.youtube.com/@coderdost

FOR Loop



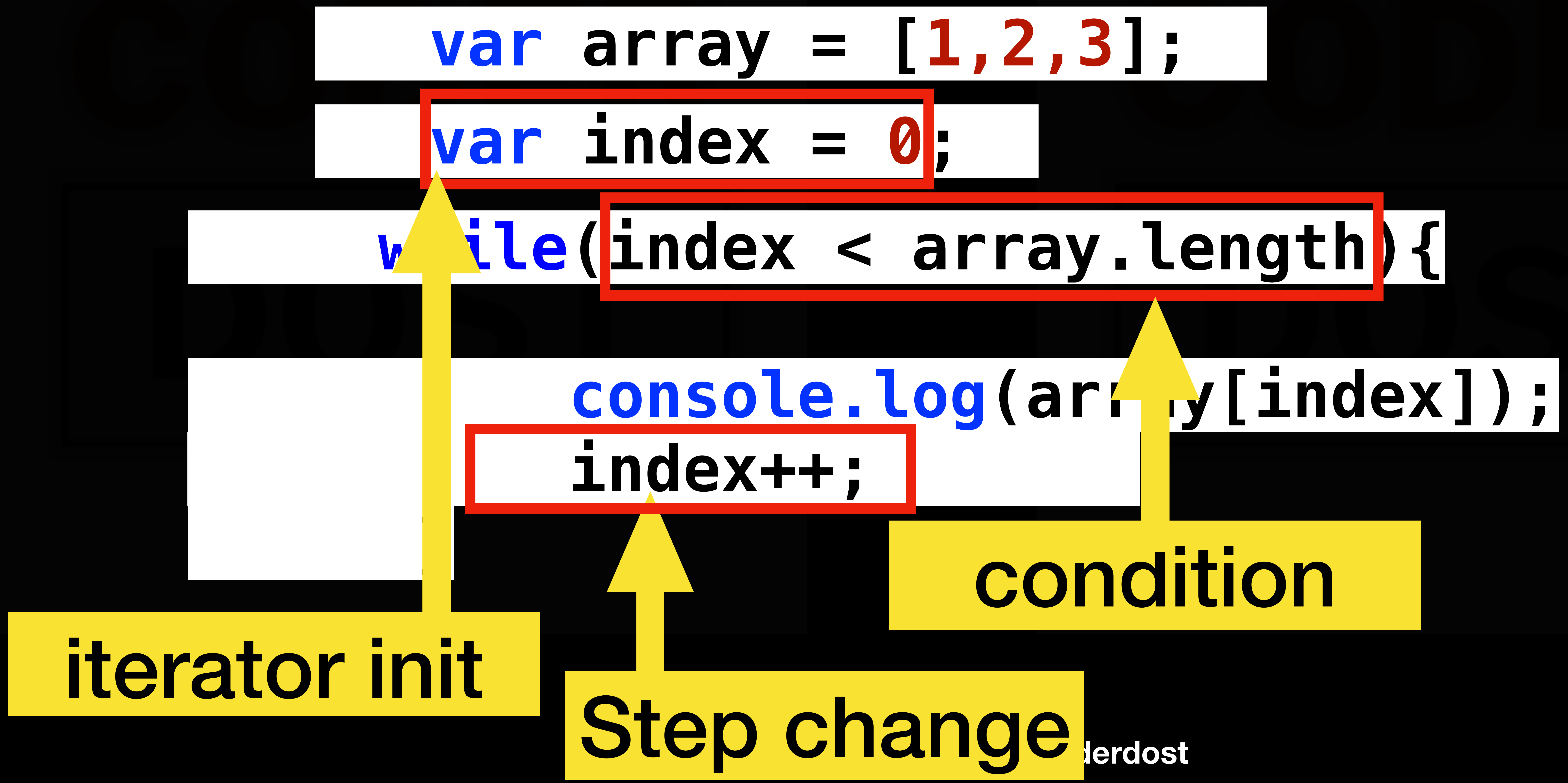
~~ITERATION 4~~

```
var array = [1,2,3];
for(let index = 0; index < array.length; index++){
    var element = array[index];
    console.log(element);
}
```

```
>    1
>    2
>    3
```

https://www.youtube.com/@coderdost

WHILE Loop



```
var array = [1,2,3];
var index = 0;
while(index < array.length){
    console.log(array[index]);
    index++;
}
```

iterator init

Step change

condition

WHILE Loop

BEFORE LOOP

```
 var array = [1,2,3];
 var index = 0;
while(index < array.length){
     console.log(array[index]);
     index++;
 }
```

WHILE Loop

ITERATION 1

```
var array = [1,2,3];
var index = 0;
while(index < array.length){
        console.log(array[index]);
        index++;
}
```

0 true 3

> 1

Index → 1

https://www.youtube.com/@coderdost

WHILE Loop

ITERATION 2

```
var array = [1,2,3];
var index = 0;
while(index < array.length){
    console.log(array[index]);
    index++;
}
```

1 true 3

```
>    1
>    2
```

Index → 2

WHILE Loop

ITERATION 3

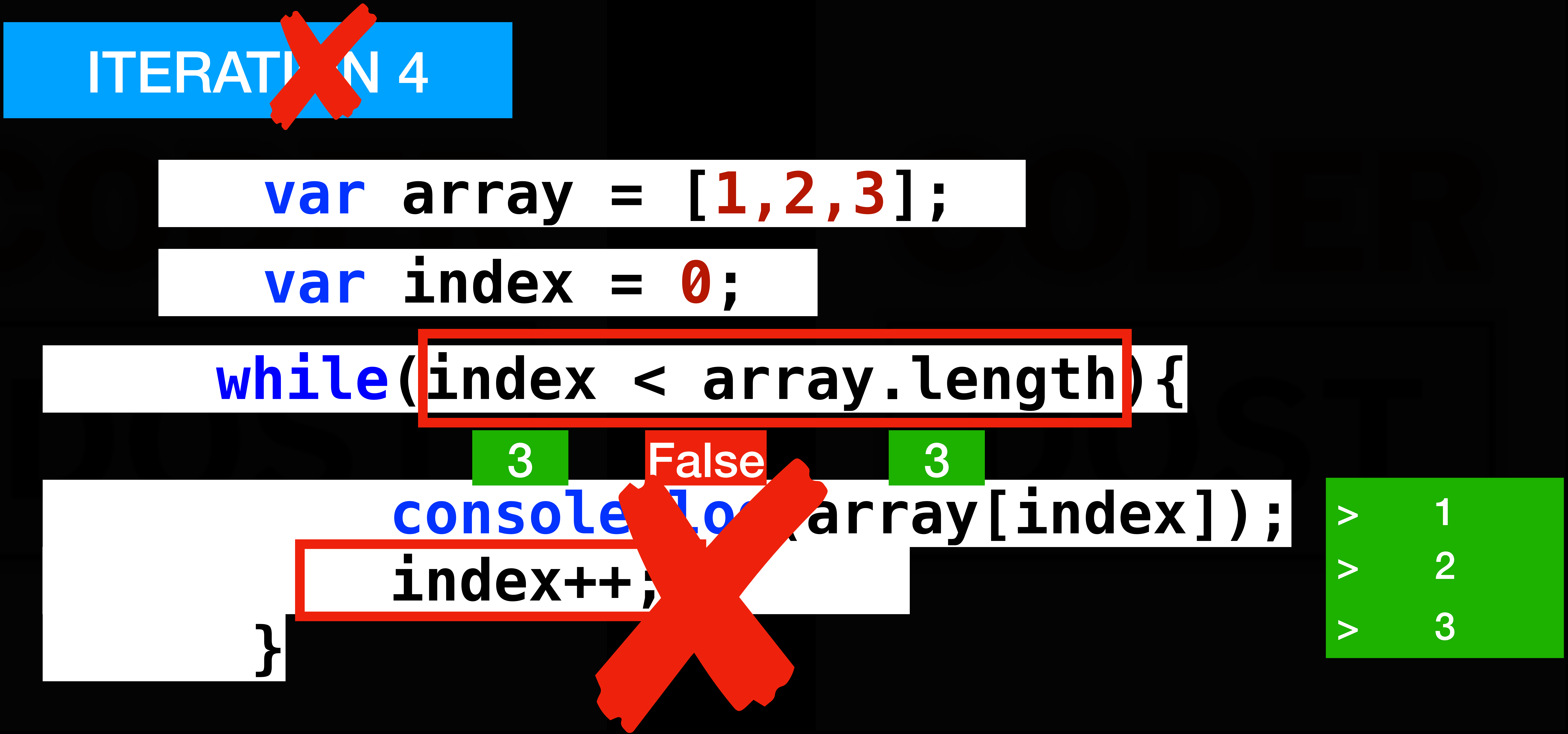
```
var array = [1,2,3];
var index = 0;
while(index < array.length){
        console.log(array[index]);
        index++;
}
```

2 true 3

```
>    1
>    2
>    3
```

Index → 3

WHILE Loop



ITERATION 4

```
var array = [1,2,3];
var index = 0;
while(index < array.length){
        console.log(array[index]);
        index++;
}
```

```
>    1
>    2
>    3
```

https://www.youtube.com/@coderdost

Break

```
let i = 0;

while (i < 6) {
  if (i === 3) {
    break;
  }
  i = i + 1;
}

console.log(i);
```

Loop ends here

prints 3

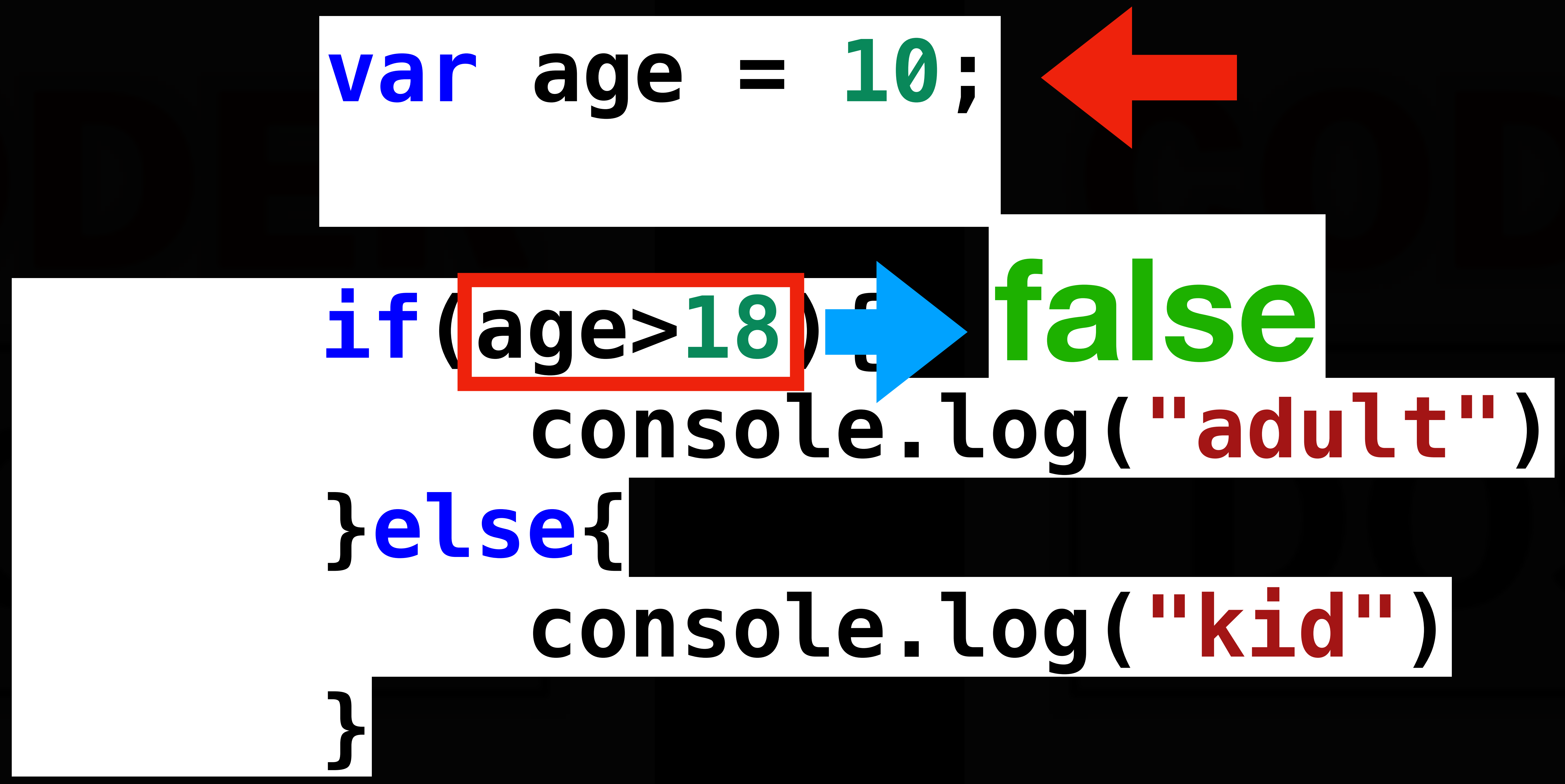
Continue

```
let text = '';

for (let i = 0; i < 10; i++) {
  if (i === 3) {
    continue;
  }
  text = text + i;
}

console.log(text);
```

Loop skips 3 here

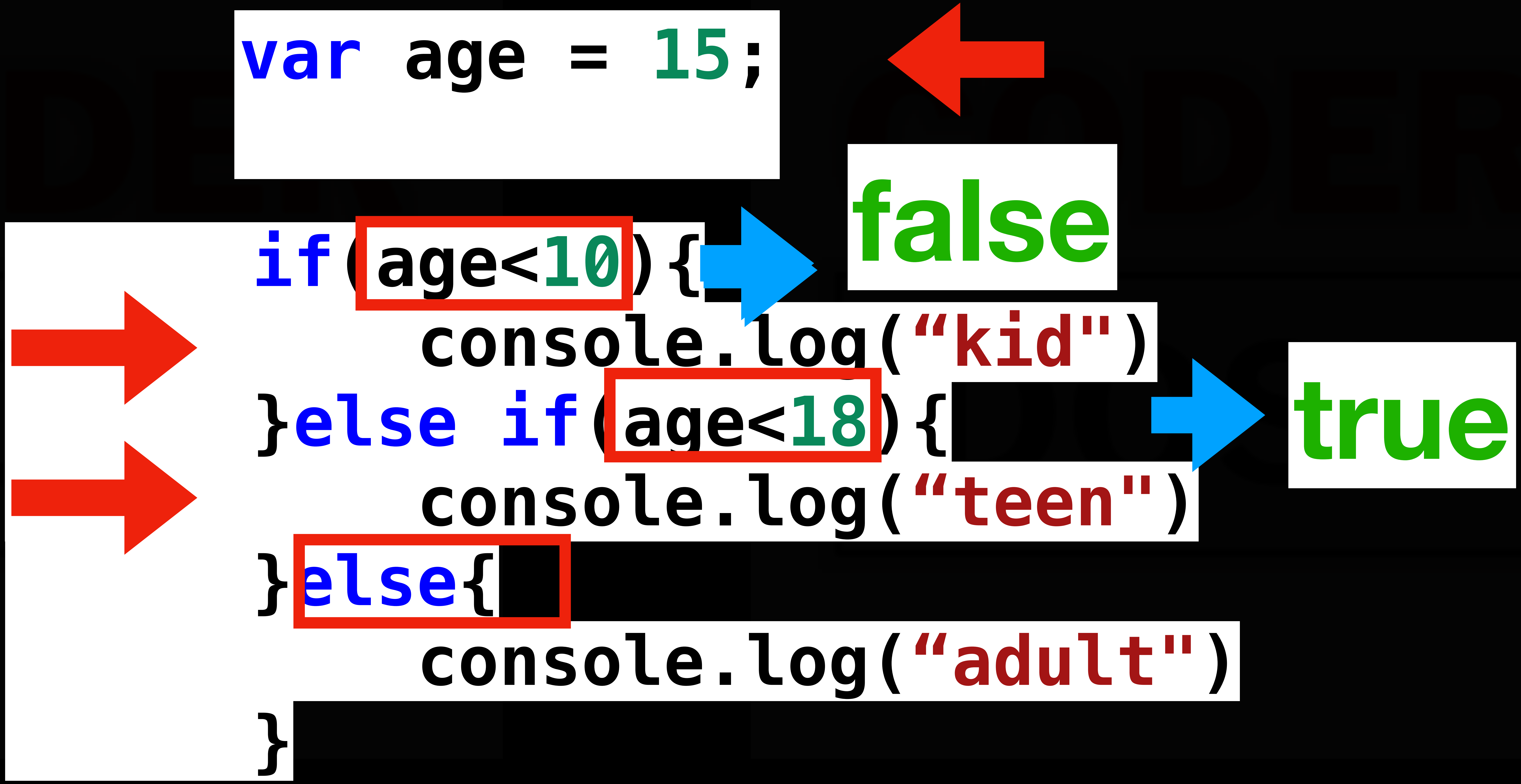
prints 012456789

If/Else conditions



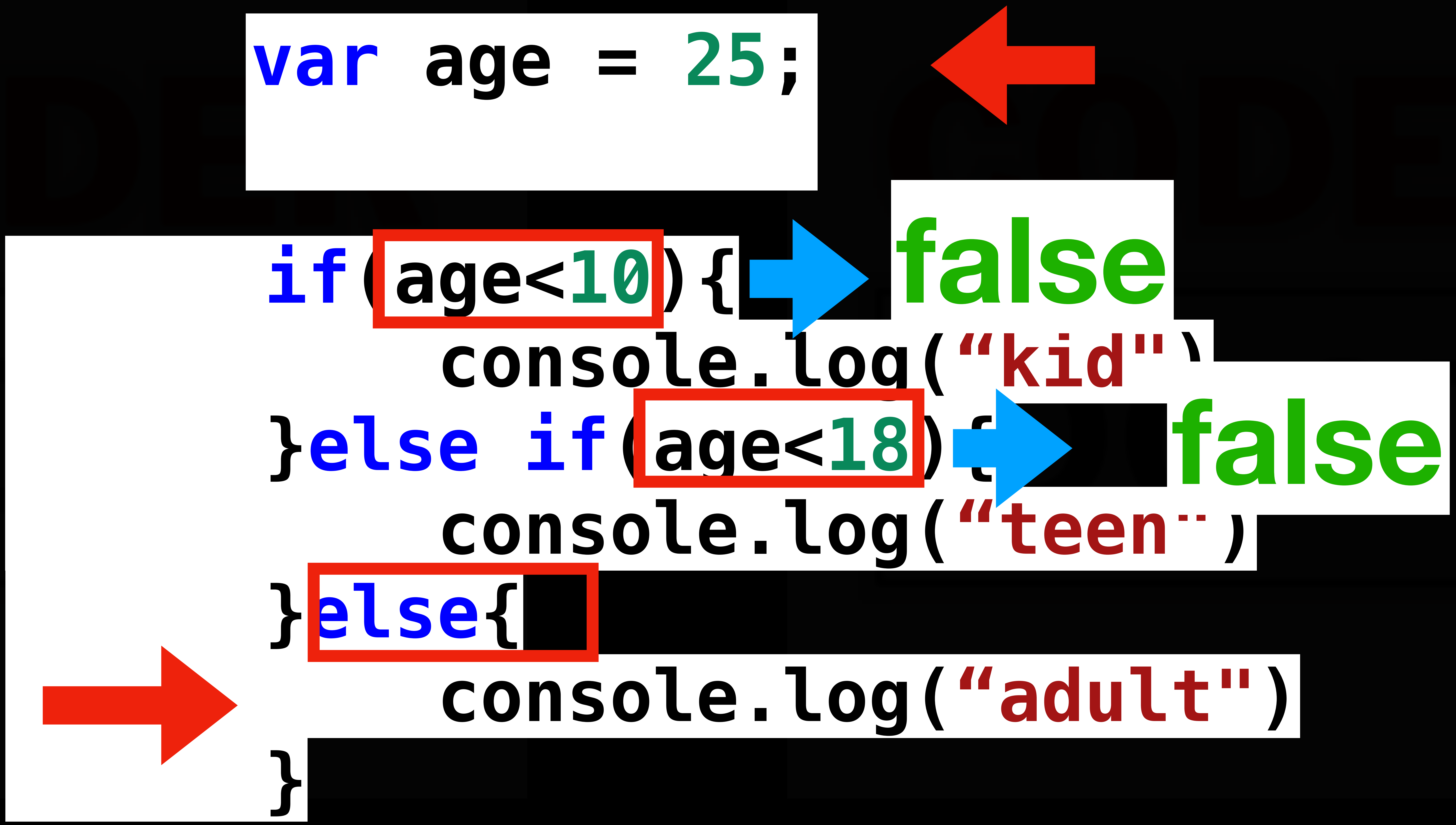
```
var age = 10;

if(age>18){
      console.log("adult")
}else{
      console.log("kid")
}
```

If/Else conditions



```
var age = 15;

if(age<10){
    console.log(“kid")
}else if(age<18){
    console.log(“teen")
}else{
    console.log(“adult")
}
```

If/Else conditions



```
var age = 25;

if(age<10){
    console.log(“kid")
}else if(age<18){
    console.log(“teen")
}else{
    console.log(“adult")
}
```

Switch Statement

```
var code = “IN”;
```

```
switch(code){
          case "IN":
             console.log("India")
          case "US" :
             console.log("United States");
          case "PK" :
             console.log("Pakistan");

      }
```

prints all values

https://www.youtube.com/@coderdost

Switch Statement

```
                   var code = “IN”;
 switch(code){
          case "IN":
             console.log(“India”);
          break;
          case "US" :
             console.log("United States”);
          break;
          case "PK" :
             console.log("Pakistan");
          break;
       }
```

prints “India”

https://www.youtube.com/@coderdost

Switch Statement

```
                    var code = “US”;
 switch(code){
          case "IN":
             console.log(“India”);
          break;
          case "US" :
             console.log("United States”);
          break;
          case "PK" :
             console.log("Pakistan");
          break;
      }
```

prints “United States”

https://www.youtube.com/@coderdost

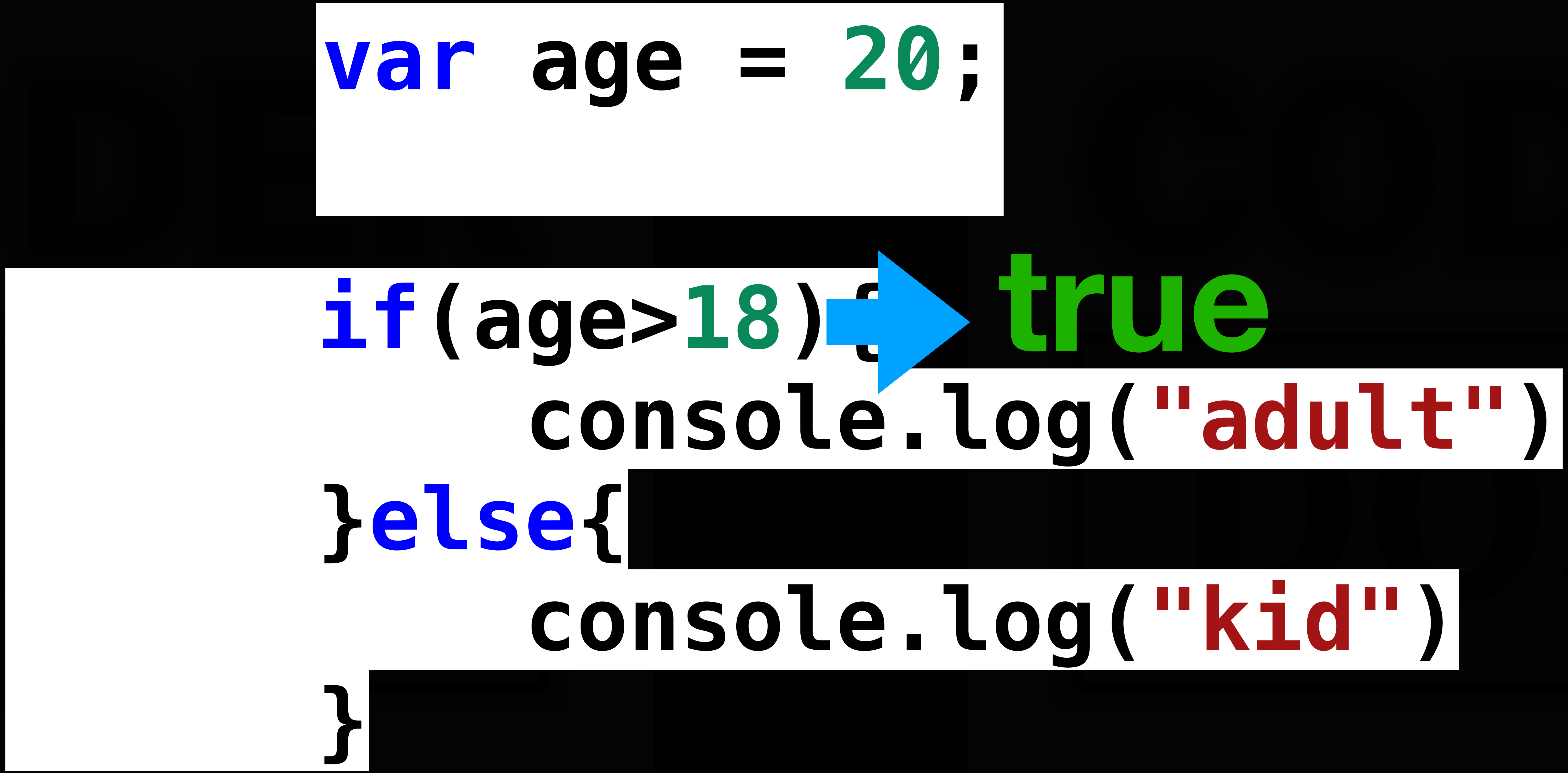
Switch Statement

```
                var code = “CN”;
switch(code){
         case "IN":
            console.log(“India”);
         break;
         case "US" :
            console.log("United States”);
         break;
         case "PK" :
            console.log("Pakistan");
         break;
         default :
            console.log(“Not Matched");

      }
```

prints “Not Matched

https://www.youtube.com/@coderdost

Truthy / Falsy values



```
var age = 20;

if(age>18){
    console.log("adult")
}else{
    console.log("kid")
}
```

true

Truthy / Falsy values

```
var age = 20;

if(age){
    console.log("adult")
}else{
    console.log("kid")
}
```

→ true

Truthy / Falsy values

true

10

“0”

“a”

“hello”

[ ] { }

false

0

“”

undefined

null

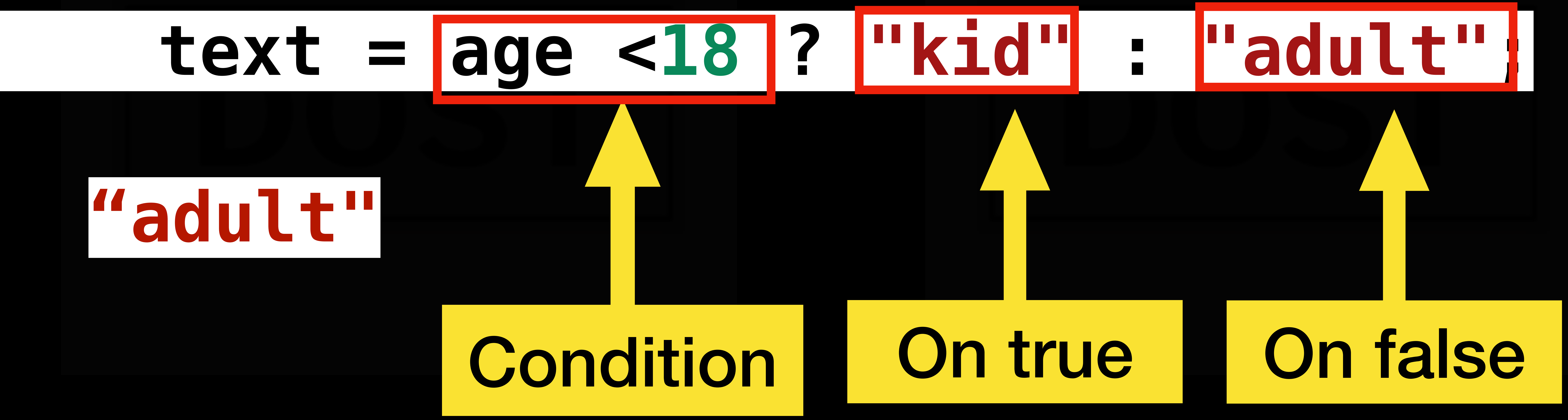
Ternary Operators (? :)

```
var age = 20;

if(age < 18){
    text = “kid”
}else{
    text = “adult”
}
```

This statement can be easily written using TERNARY

Ternary Operators (? :)

```
var age = 20;
```



```
text = age <18 ? "kid" : "adult";
```

```
“adult"
```

3. Functions

Functions

```
move( “right”,10 )
```

functions are special objects which can contain multiple JS statement, and can be re-used

Defining Functions : Normal Style

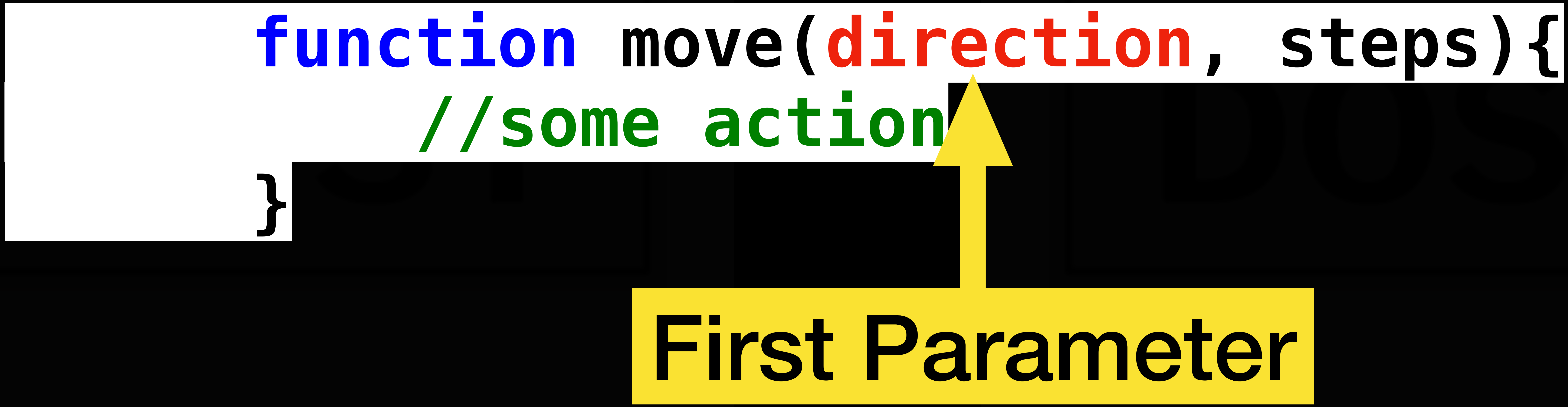
move(“right”,10)

```
function move(direction, steps){
    //some action
}
```

function name

Defining Functions

move(“right”,10)



```
function move(direction, steps){
    //some action
}
```

First Parameter

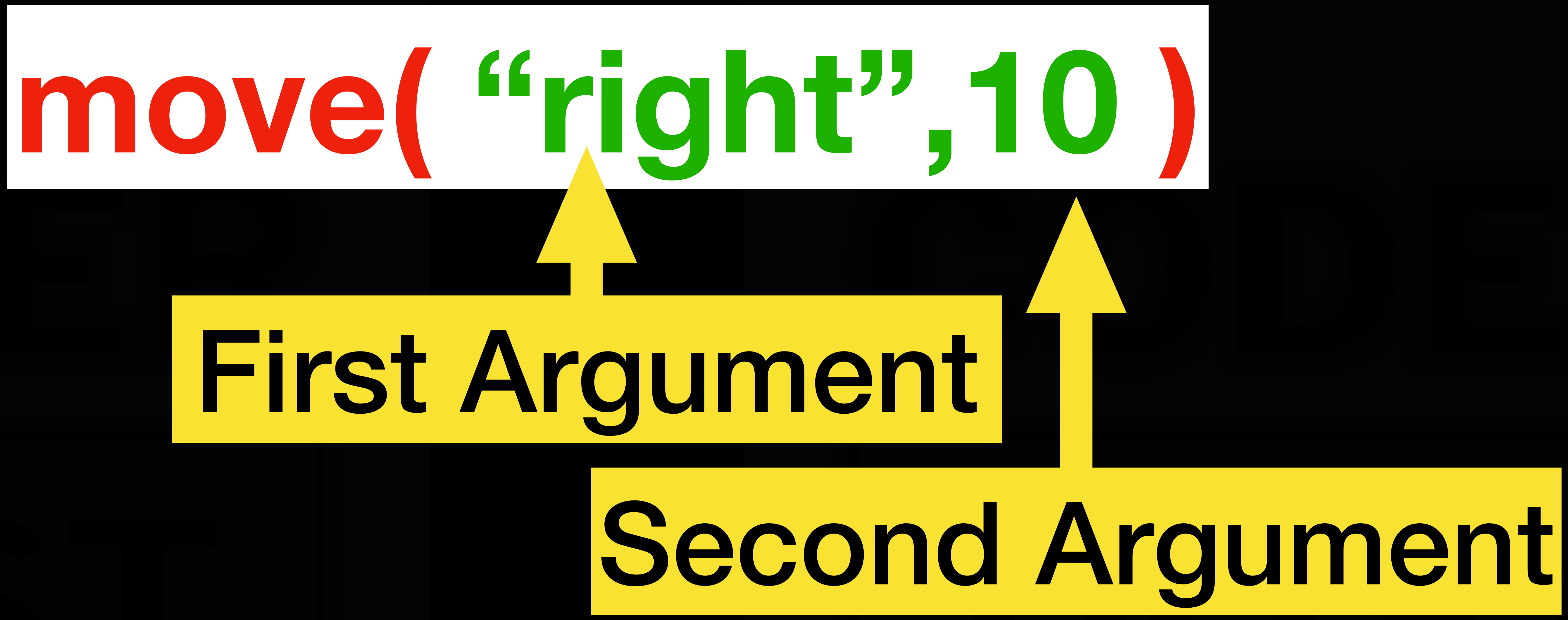
Defining Functions

```
move( “right”,10 )
```

```
        function move(direction, steps){
            //some action
        }
```

Second Parameter

Calling Functions

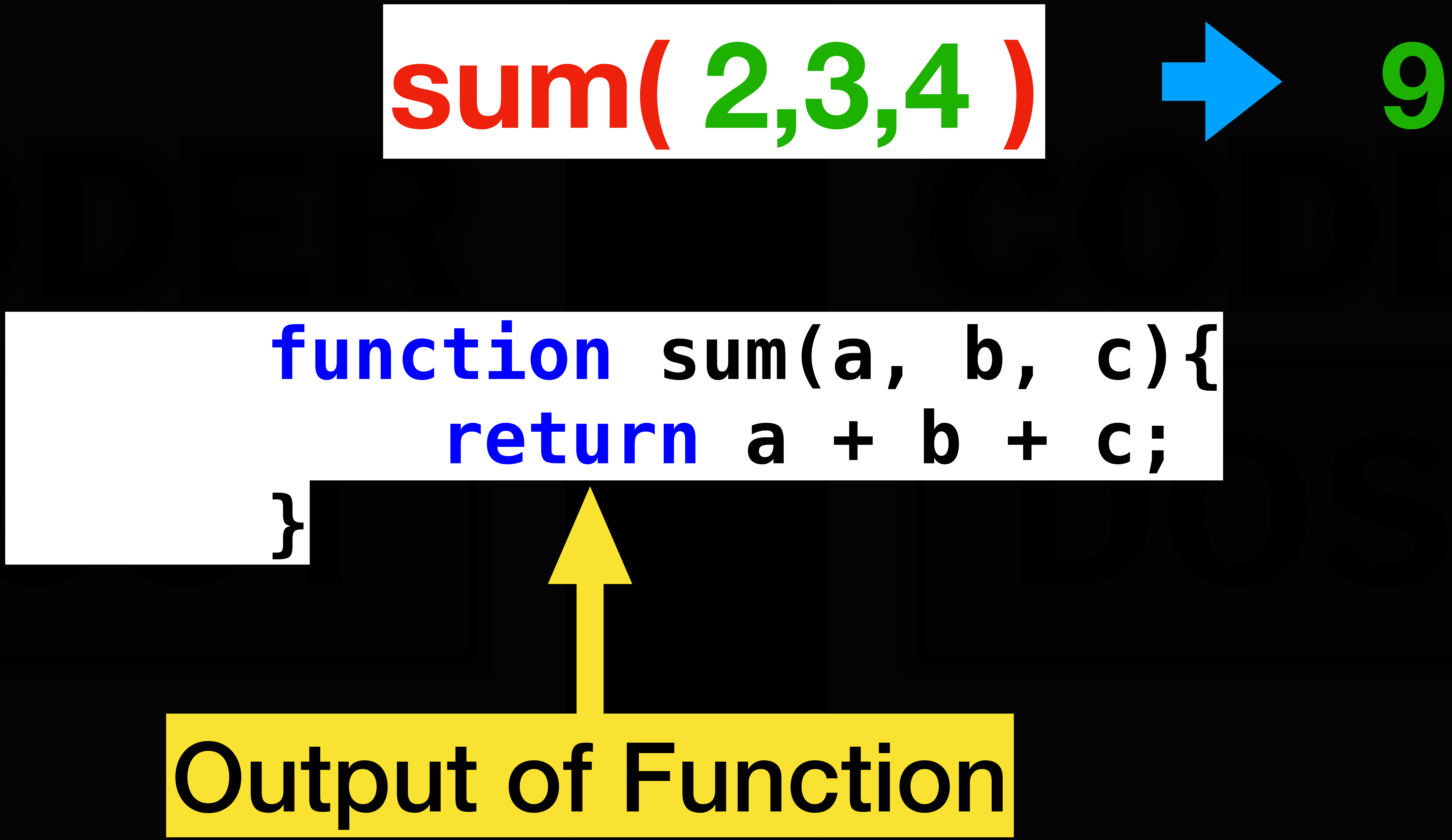


```
move( “right”,10 )
```

First Argument

Second Argument

Defining Functions

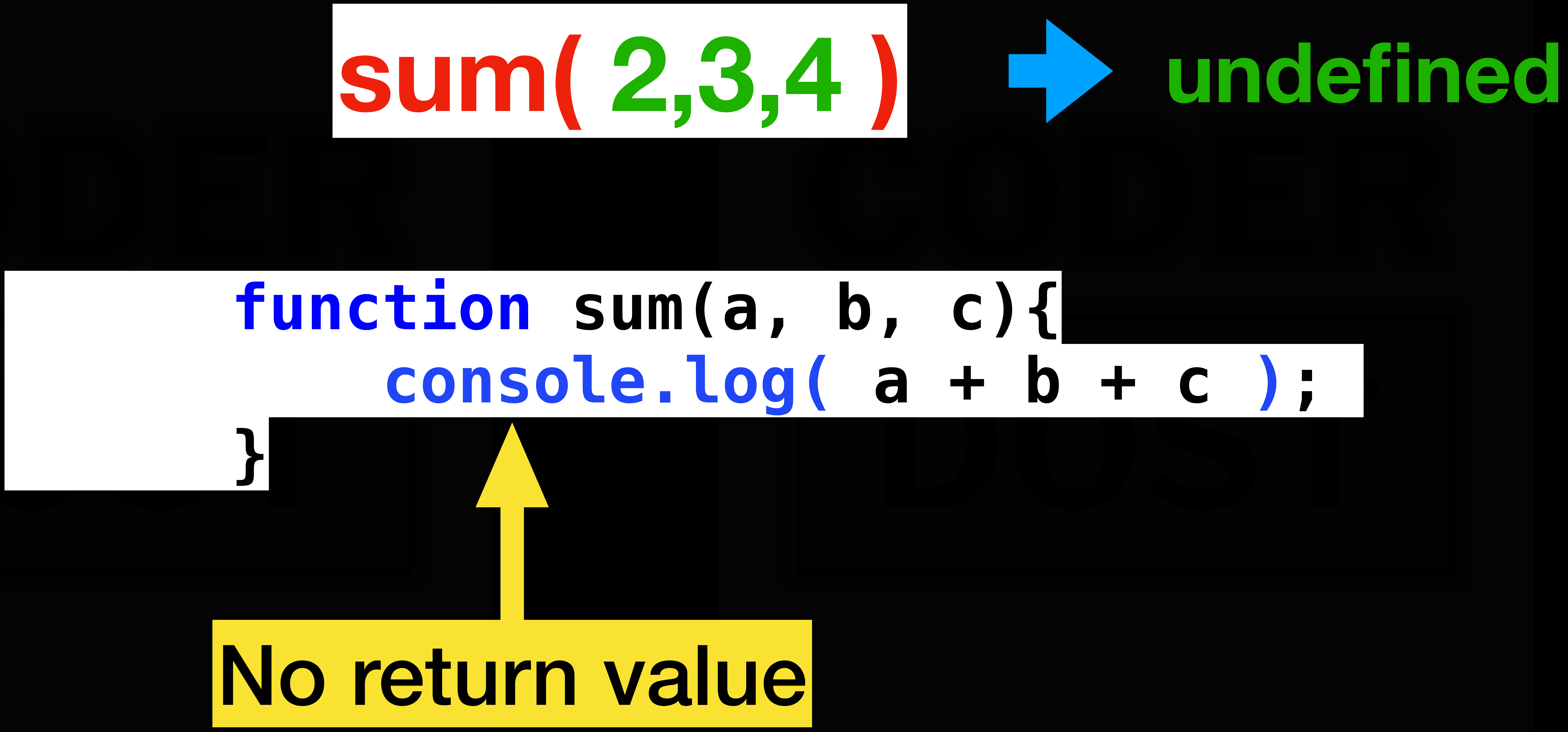


sum(2,3,4) → 9

```
function sum(a, b, c){
    return a + b + c;
}
```

Output of Function

Defining Functions

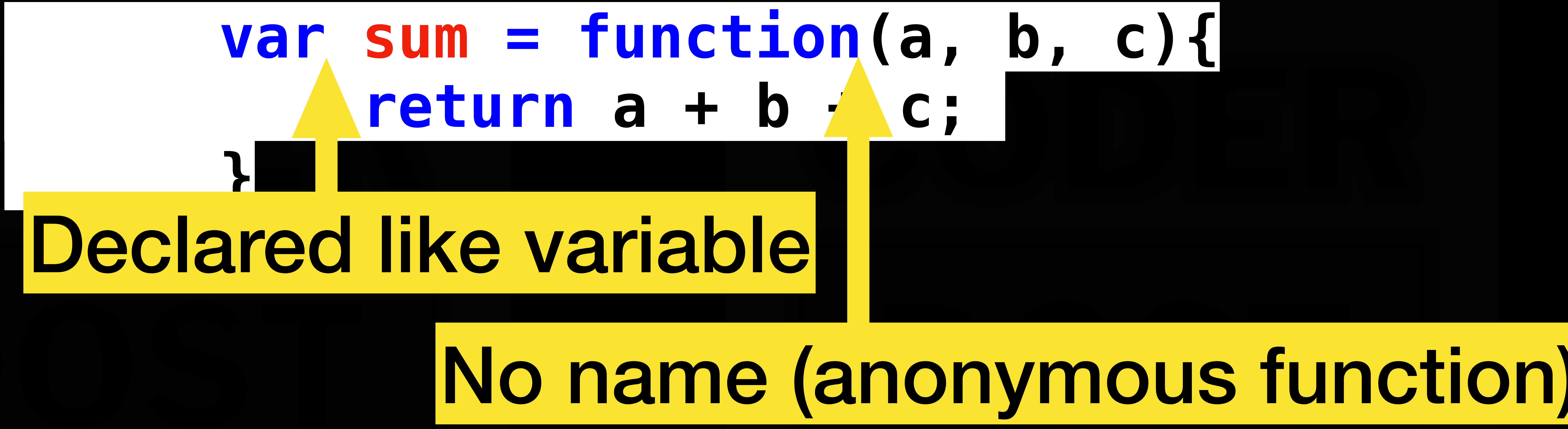


sum(2,3,4) → undefined

```
function sum(a, b, c){
    console.log( a + b + c );
}
```

No return value

Defining Function : function expression



```
var sum = function(a, b, c){
    return a + b + c;
}
```

Declared like variable

No name (anonymous function)

sum(2,3,4) → 9

Both Type of definition work Same

```
function sum(a, b, c){
    return a + b + c;
}
```

```
var sum = function(a, b, c){
    return a + b + c;
}
```

sum(2,3,4) → 9

Normal function definition can be called before initialisation

sum(2,3,4) → 9

```
function sum(a, b, c){
    return a + b + c;
}
```

Reason : Hoisting

https://www.youtube.com/@coderdost

function expression Can’t be called before initialisation

sum(2,3,4) → ERROR

```
      var sum = function(a, b, c){
          return a + b + c;
      }
```

Hoisting

sum(2,3,4) → **9**

```
function sum(a, b, c){
    return a + b + c;
}
```

JS Interpreter reads function definition before executing code

Default Parameters

```
let weight = function(m, g=9.8){
        return m * g;
}
```

```
weight(10,9)
```

→ 90

```
weight(10)
```

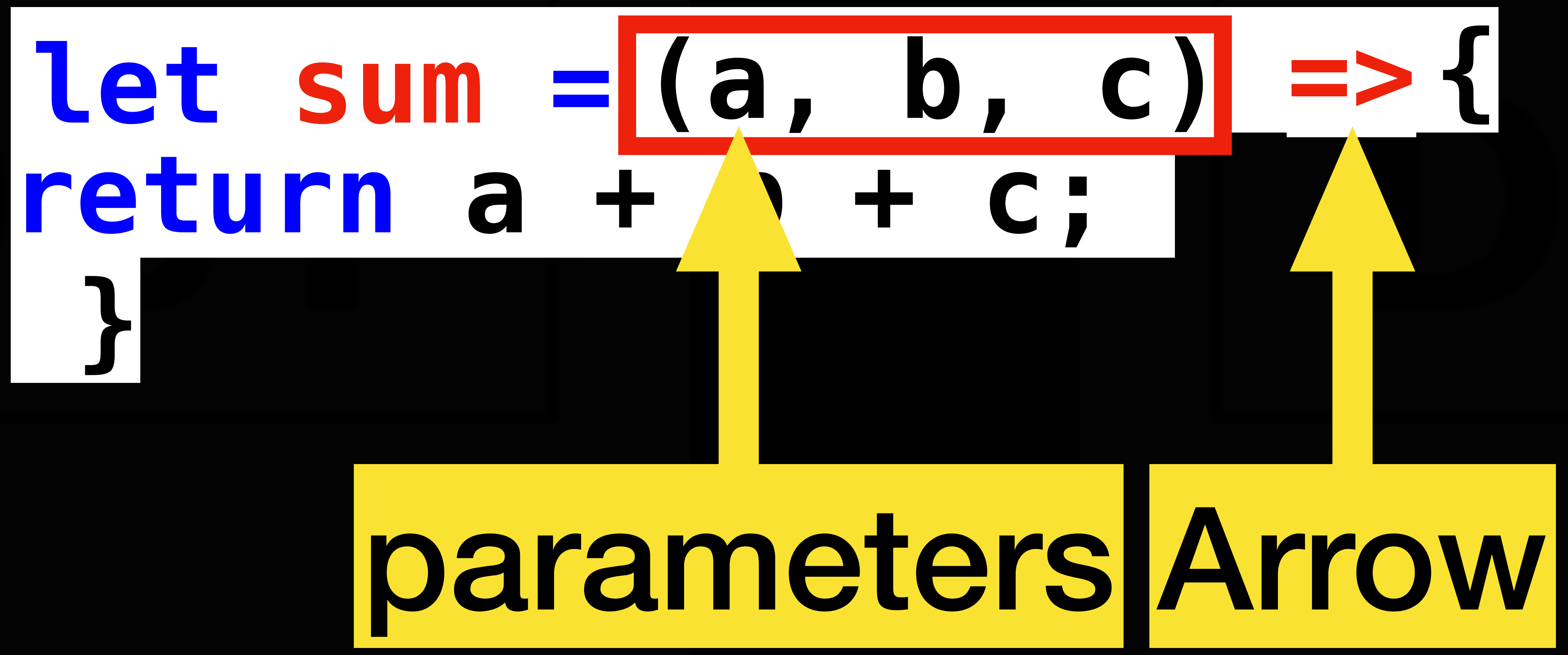
→ 98

10 *9.8

https://www.youtube.com/@coderdost

Arrow Functions

```
let sum = function(a, b, c){
          return a + b + c;
           }
```



```
let sum = (a, b, c) =>{
return a + b + c;
 }
```

parameters

Arrow

Arrow Functions

```
let sum = function(a, b, c){
    return a + b + c;
}
```

```
let sum =(a, b, c) => { return a + b + c;}
```

```
let sum =(a, b, c) => a + b + c;
```

No Braces implicitly mean return

Functions v/s Arrow Functions

Functions

Good for multi-line logic

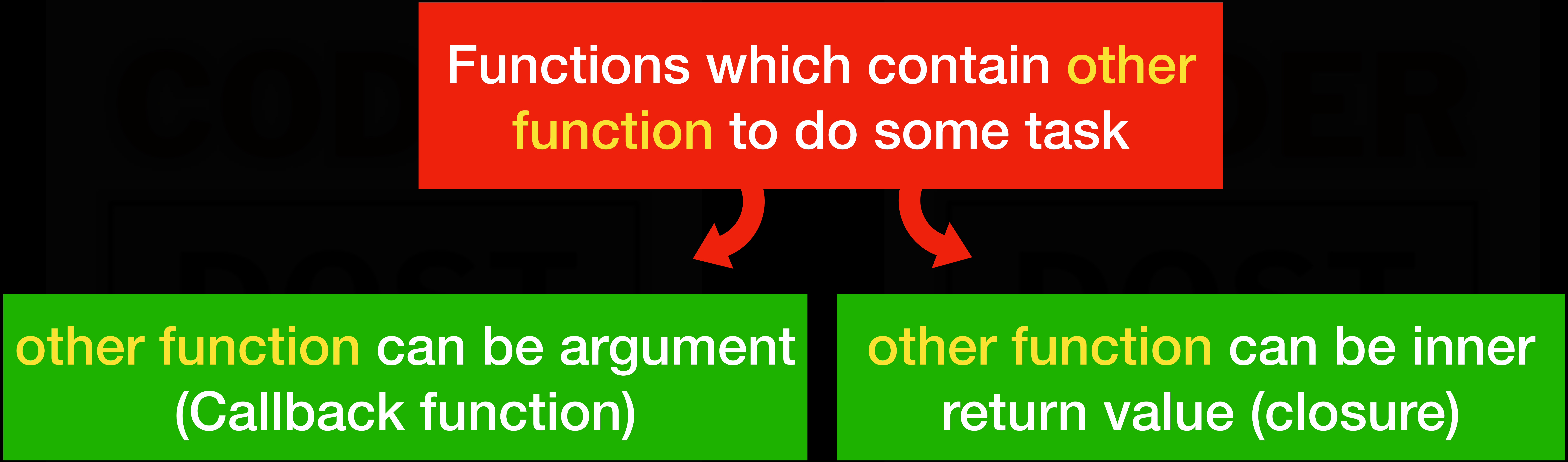
Creates a new "this" context

Arrow functions

Good for single line returns

Doesn't create a "this" context

Higher order functions



Functions which contain other function to do some task

other function can be argument (Callback function)

other function can be inner return value (closure)

https://www.youtube.com/@coderdost

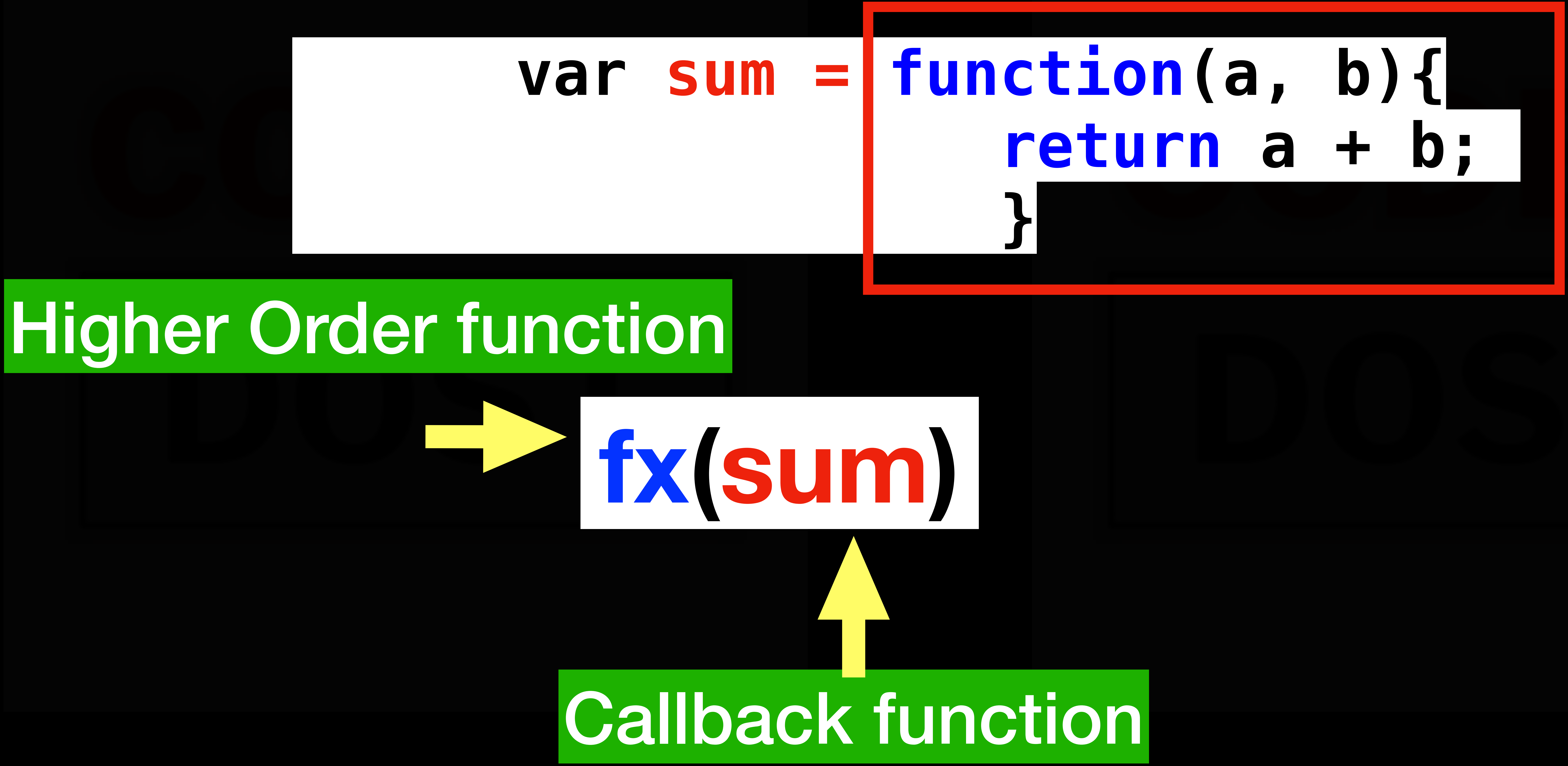
1. Callback Functions

```
function sum(a, b){
    return a + b;
}
```

FUNCTIONS ARE OBJECTS

can be passed to as arguments

1. Callback Functions



```
var sum = function(a, b){
              return a + b;
          }
```

Higher Order function

fx(sum)

Callback function

https://www.youtube.com/@coderdost

1. Callback Functions

```
var talk = function(fx){
        fx();
}
```

sayHi()

‘sayHi’
called by
‘talk’
at later stage

```
var sayHi = function(){
       console.log(“hi”);
}
```

```
talk(sayHi)
```

1. Callback Functions

```
var calc = function(fx,a,b){
      return fx(a,b);
}
```

```
var sum = function(x,y){
    return a+b;
}
```

calc(sum,4,5) → 9

https://www.youtube.com/@coderdost

Callback Functions

```
var calc = function(fx,a,b){
     return fx(a,b);
}
```

```
var diff = function(x,y){
     return a-b;
}
```

calc(diff,4,5) → -1

2. Function returning function

```
function makeFunc() {
  const name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  return displayName;
}

const myFunc = makeFunc();
myFunc();
```

same functionality as displayName, but can access “name” variable

this is also example of a “Closure” which we will cover at last

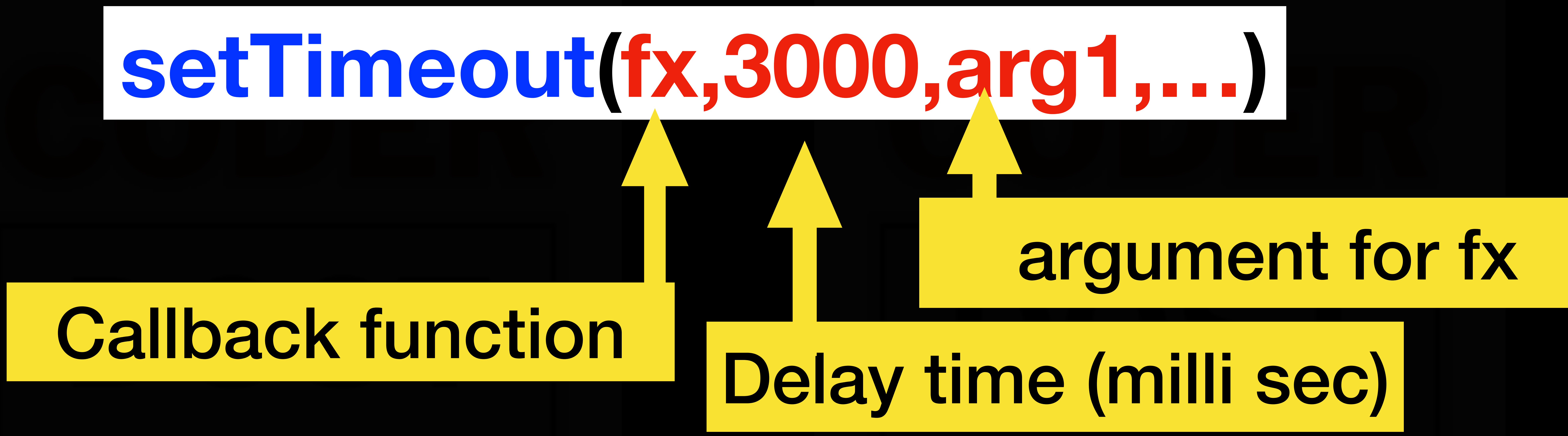
IIFE- Immediately Invoked Function Expression

function is made as expression using
( ) - so function doesn't require name

```
(function () {
  // protect inner code from
 access, protects global scope
})();
```

expression executed immediately

Timer Functions



```
setTimeout(fx,3000,arg1,...)
```

Callback function

Delay time (milli sec)

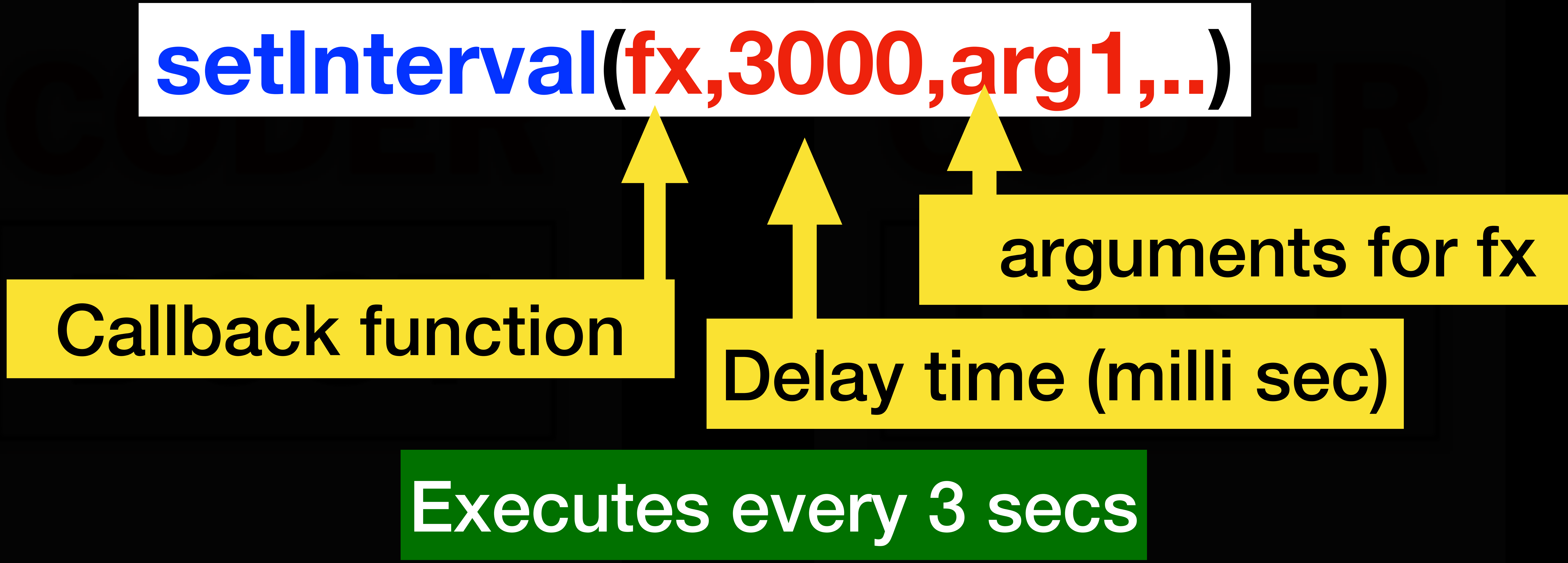
argument for fx

Executes after 3 secs

Timer Functions

```
setTimeout ( function( ) {
                console.log(“hello”)
            },
        3000 )
```

Timer Functions



```
setInterval(fx,3000,arg1,..)
```

Callback function

Delay time (milli sec)

arguments for fx

Executes every 3 secs

Timer Functions

```
setInterval( function( ) {
                console.log(“hello”)
             },
  3000 )
```

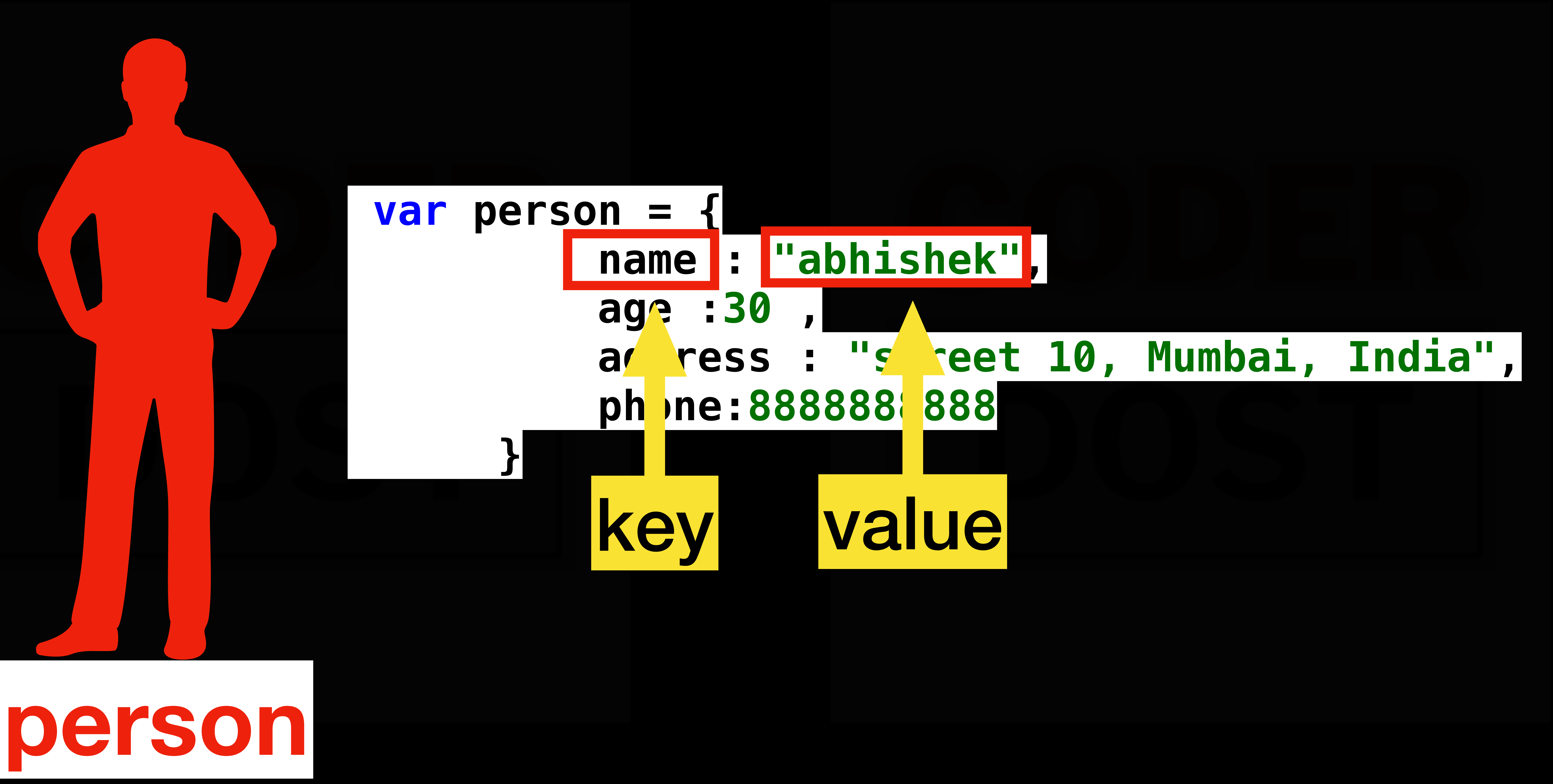
Callback function

4. Objects

Objects

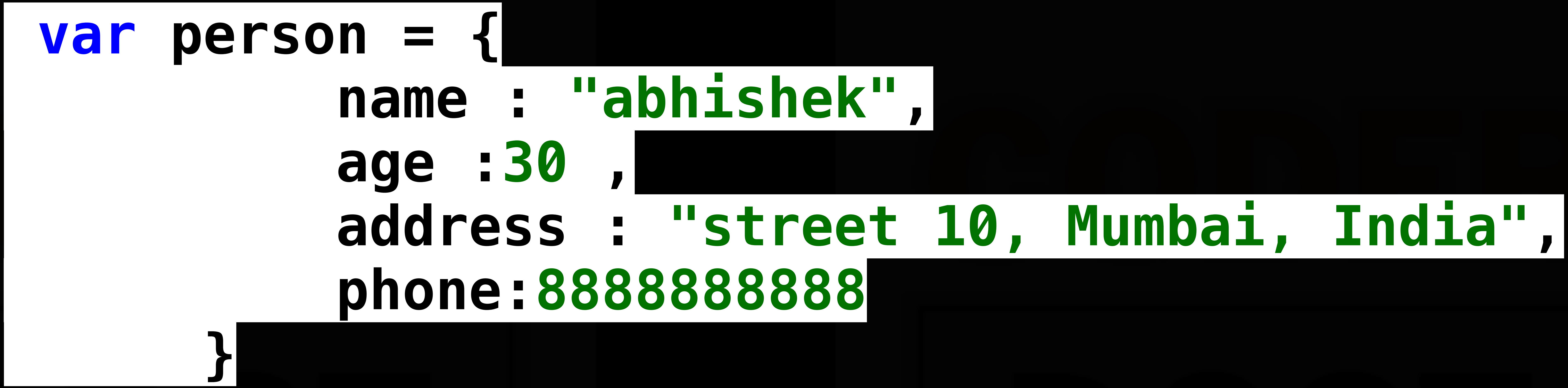
Name	abhishek
Age	30
Address	Street 10, mumbai, india
Mobile	888888888

Objects



```
var person = {
                 name : "abhishek",
                 age :30 ,
                 address : "street 10, Mumbai, India",
                 phone:8888888888
            }
```

Accessing Objects (dot)

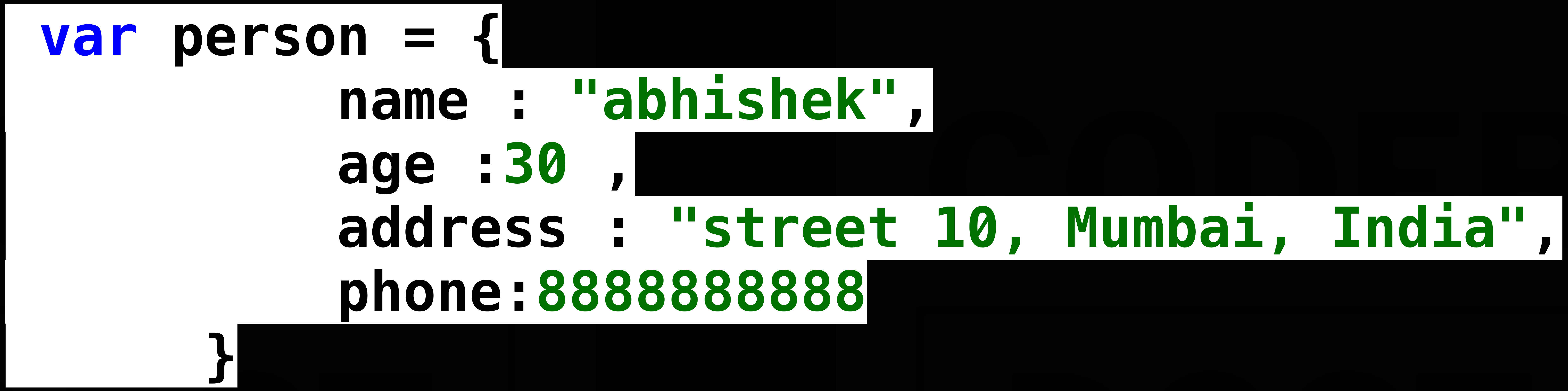


```
 var person = {
           name : "abhishek",
           age :30 ,
           address : "street 10, Mumbai, India",
           phone:8888888888
      }
```

person.name → “abhishek”

person.age → 30

Accessing Objects (bracket style)

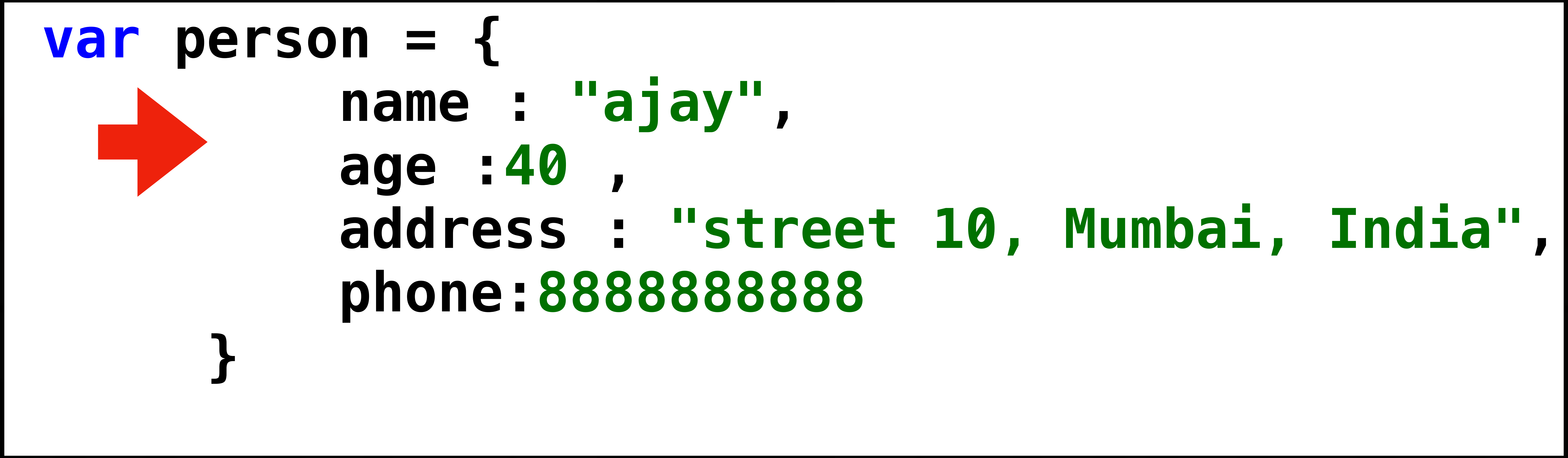


```
 var person = {
              name : "abhishek",
              age :30 ,
              address : "street 10, Mumbai, India",
              phone:8888888888
          }
```

person.[“name”] → “abhishek”

person.[“age”] → 30

Writing Objects (dot)



```
var person = {
            name : "ajay",
            age :40 ,
            address : "street 10, Mumbai, India",
            phone:8888888888
    }
```

person.name = “ajay”

person.age = 40

Nested Objects

person

Name	abhishek
Age	30
Address	Street 10, mumbai, india
Mobile	888888888

Nested Objects

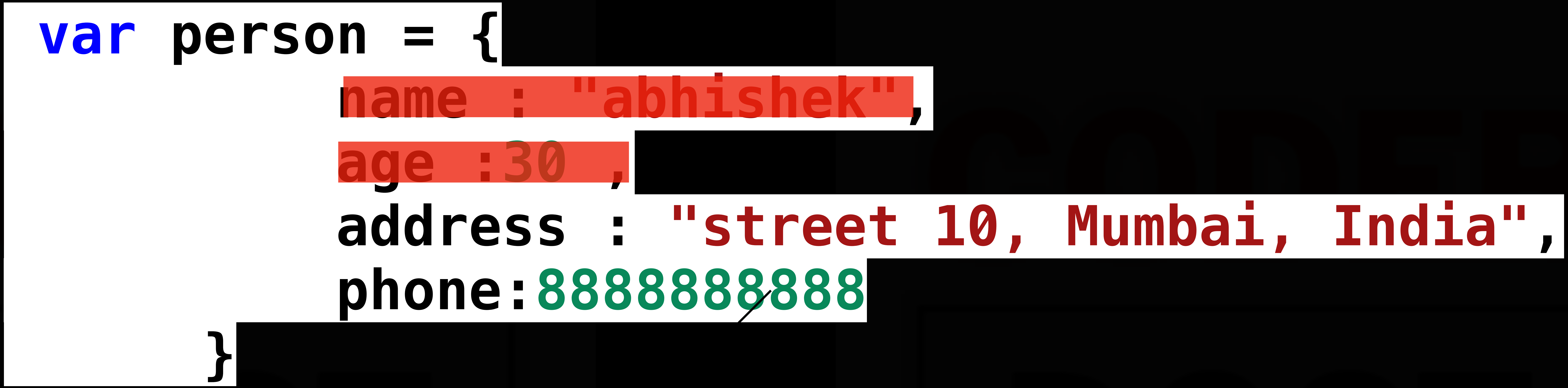
```
var person = {
         name : "abhishek",
         age :30 ,
         address : {
                street:“street 10”,
                city:“mumbai”,
                country:“india”
              },
         phone:8888888888
      }
```

person.address.city → “mumbai”

person.address.country → “india”

https://www.youtube.com/@coderdost

Deleting Properties

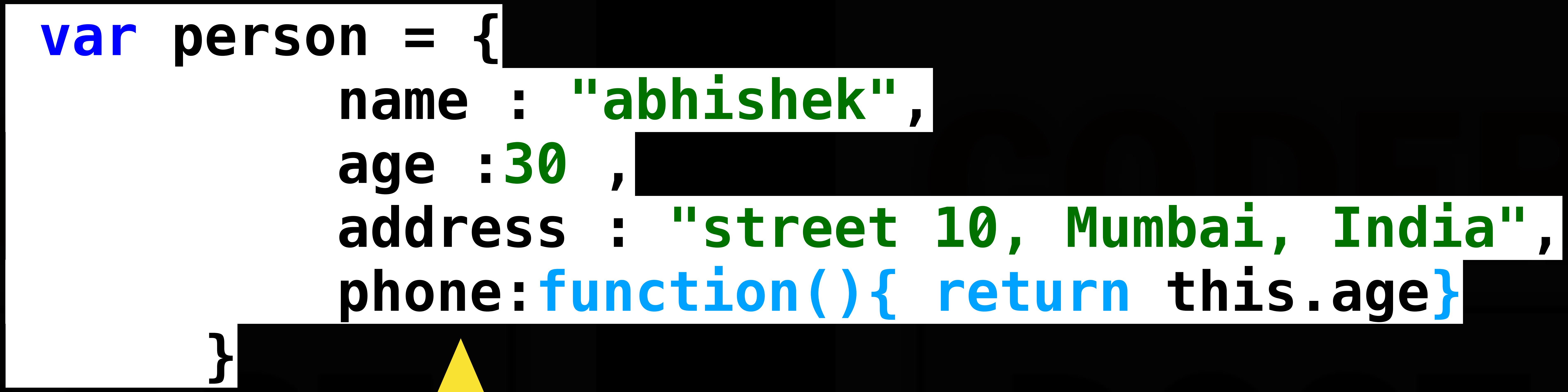


```
var person = {
             name : "abhishek",
             age :30 ,
             address : "street 10, Mumbai, India",
             phone:8888888888
         }
```

delete person.name

delete person.age

Function vs Methods

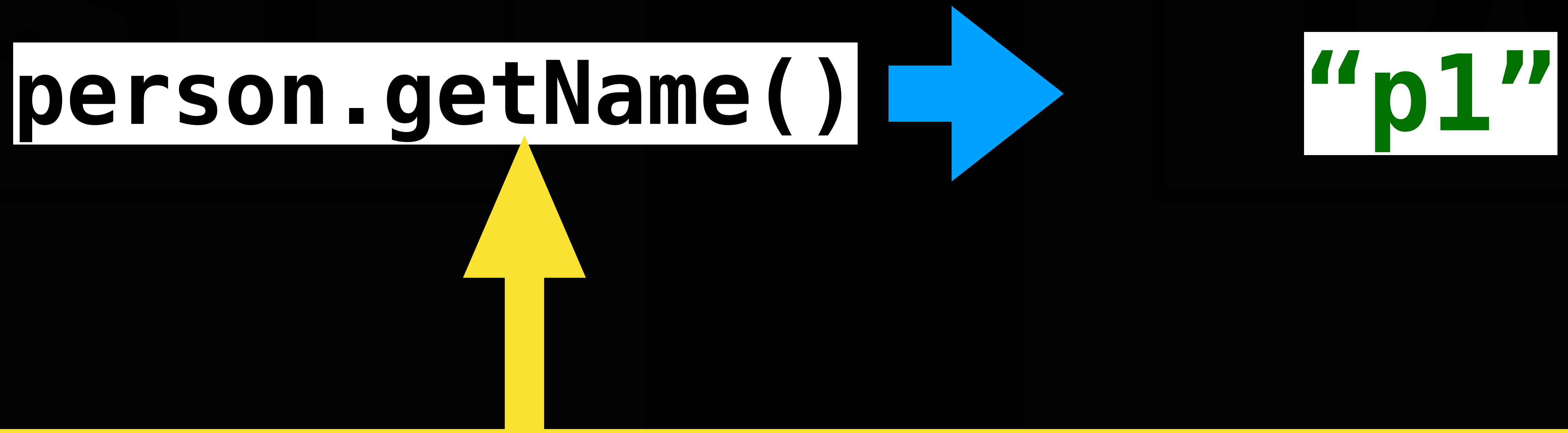


```
var person = {
            name : "abhishek",
            age :30 ,
            address : "street 10, Mumbai, India",
            phone:function(){ return this.age}
        }
```

methods = function of an object

this

```
 const person = {
            name : “p1”,
            getName: function(){
                   return this.name
                  }
}
```



```
person.getName()
```

“p1”

‘this’ here will refer to calling object (person)

https://www.youtube.com/@coderdost

forEach()

```
const cities = [“NY”,”LA”,”TX”];
```

```
const lowerCased = [];
```

```
cities.forEach((city) => lowerCased.push(city))
```

`lowerCased` → `[“ny”,”la”,”tx”]`

https://www.youtube.com/@coderdost

Math Library

Math.abs(-2) → 2

Math.round(2.66) → 3

Math.floor(3.55) → 3

Math.ceil(3.45) → 4

Math.pow(2,3) → 8

Math Library

```
Math.sqrt(4)              → 2
Math.max(4,5,100,0,1)     → 100
Math.min(4,5,100,0,1)     → 0
Math.random()             → 0.45537377
Math.trunc(-99.9)         → -99
```

Call

```
 const person = {
              name : “p1”,
              getName: function(){
                    return this.name
                   }
}

 const p2 = { name : “p2” }
```

args

new value of ‘this’

```
person.getName.call(p2)
```

→ “p2”

```
person.getName.call(p2, args)
```

Apply

```
const person = {
             name : “p1”,      args
             getName: function(){
                    return this.name
                   }
}
```

```
const p2 = { name : “p2” }
```

new value of ‘this’

```
person.getName.apply(p2)
```

“p2”

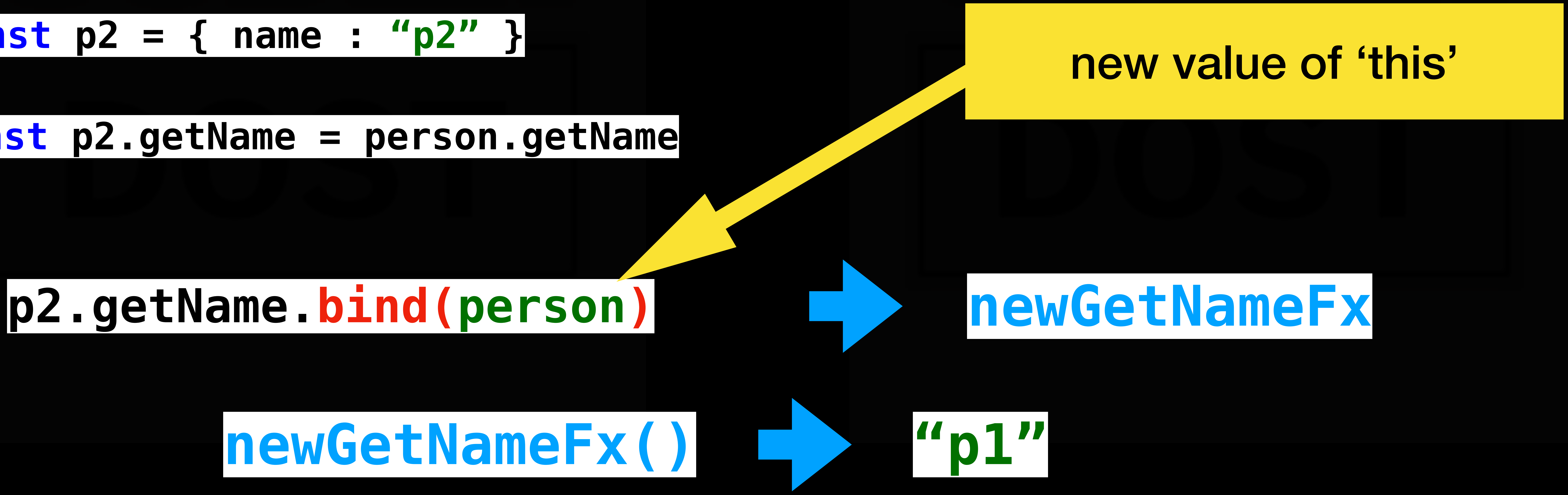
```
person.getName.apply(p2, [args])
```

Bind

```
const person = {
            name : “p1”,
            getName: function(){
                    return this.name
                   }
}
```

```
const p2 = { name : “p2” }
```

```
const p2.getName = person.getName
```



new value of ‘this’

```
p2.getName.bind(person)
```

newGetNameFx

```
newGetNameFx()
```

“p1”

Call by Reference

```
var person = {
                name : "abhishek",
                age :30 ,
                address : "street 10",
                phone:8888888888
        }
```

```
var anotherPerson = person;
```

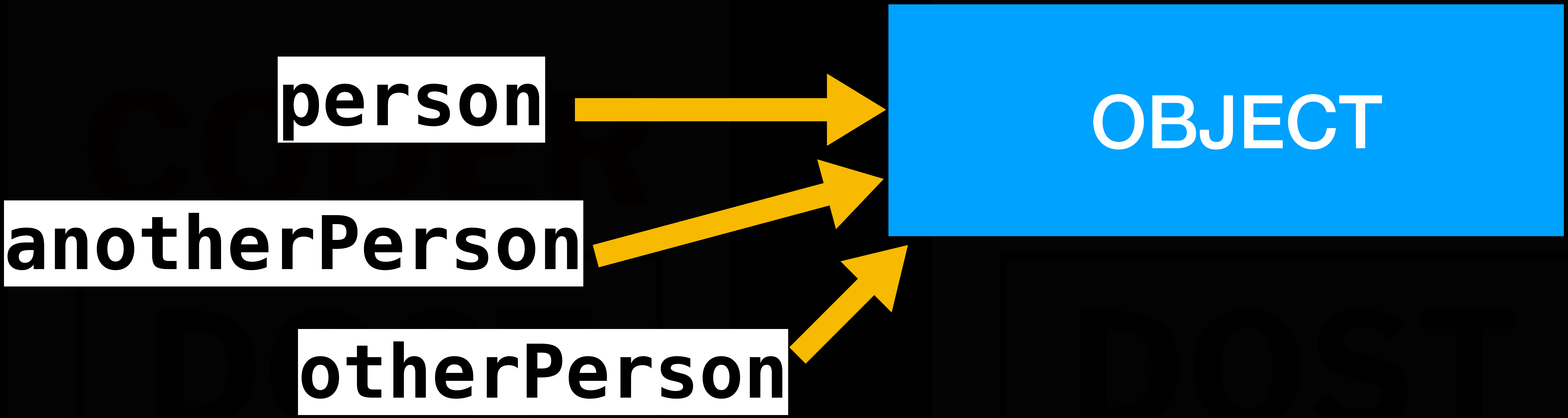
```
anotherPerson.name = “jack”;
```

```
person.name
```

→ “jack” ?

https://www.youtube.com/@coderdost

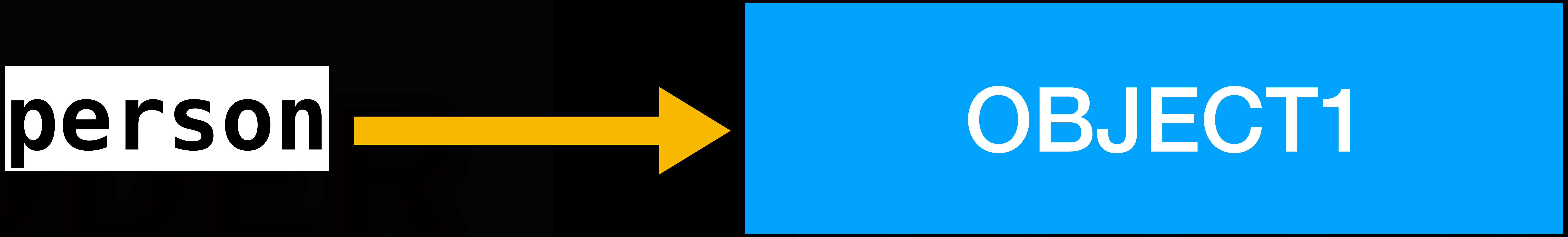
Call by Reference



```
let anotherPerson = person;
let otherPerson = anotherPerson;
```

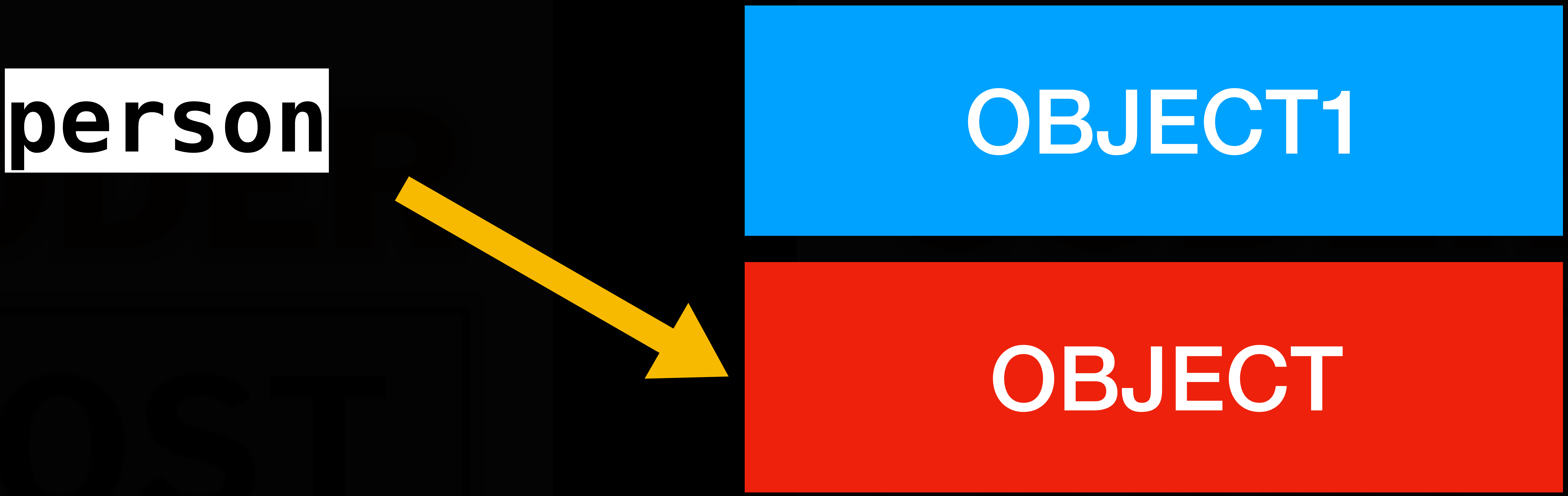
https://www.youtube.com/@coderdost

Reference Change



```
let person = object1;
```

Reference Change



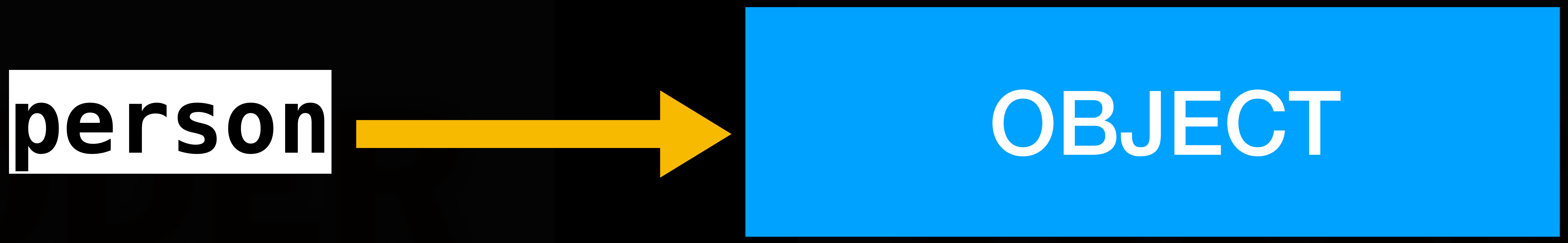
person

OBJECT1

OBJECT

```
    let person = object1;
        person = object2;
```

Const : Avoid changing reference



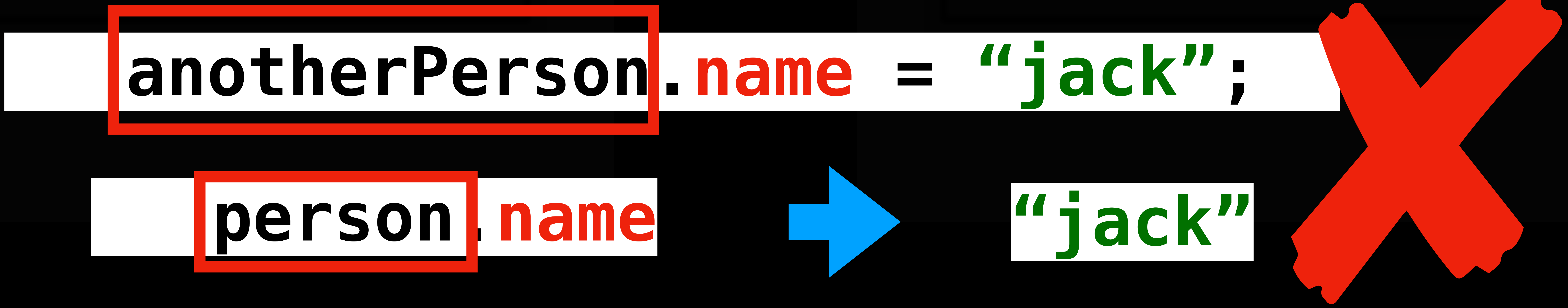
```
const person = {};
person  = somethingElse;
```

❌

Copying JS Objects : Wrong way

```
var person = {
              name : "abhishek",
              age :30 ,
              address : "street 10",
              phone:8888888888
        }
```

```
var anotherPerson = person;
```



```
anotherPerson.name = “jack”;
```

```
person.name → “jack”
```

Copying JS Objects : Right way

```
var person = {
            name : "abhishek",
            age :30 ,
            address : "street 10",
            phone:8888888888
       }
```

```
var anotherPerson = {…person};
```

```
anotherPerson.name = “jack”;
```

```
person.name
```

→

```
“abhishek”
```

Spread operator(...) explained later

for-in loop

```
const object = { a: 1, b: 2, c: 3 };

for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}
```

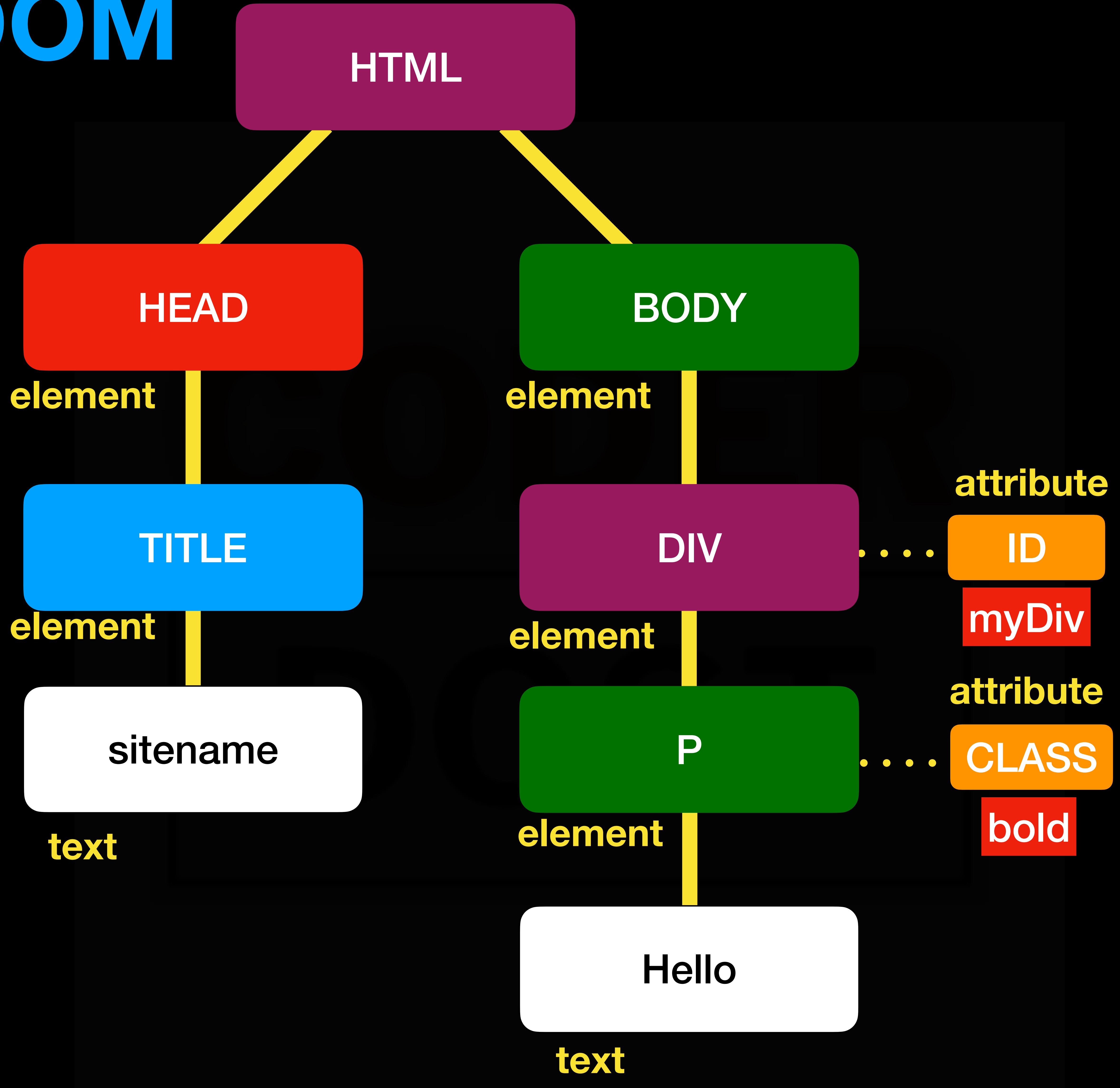
these properties are called enumerable

5. DOM

HTML DOM

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
         <p class=“bold”>
            Hello
          </p>
      </div>
 </body>
</html>
```

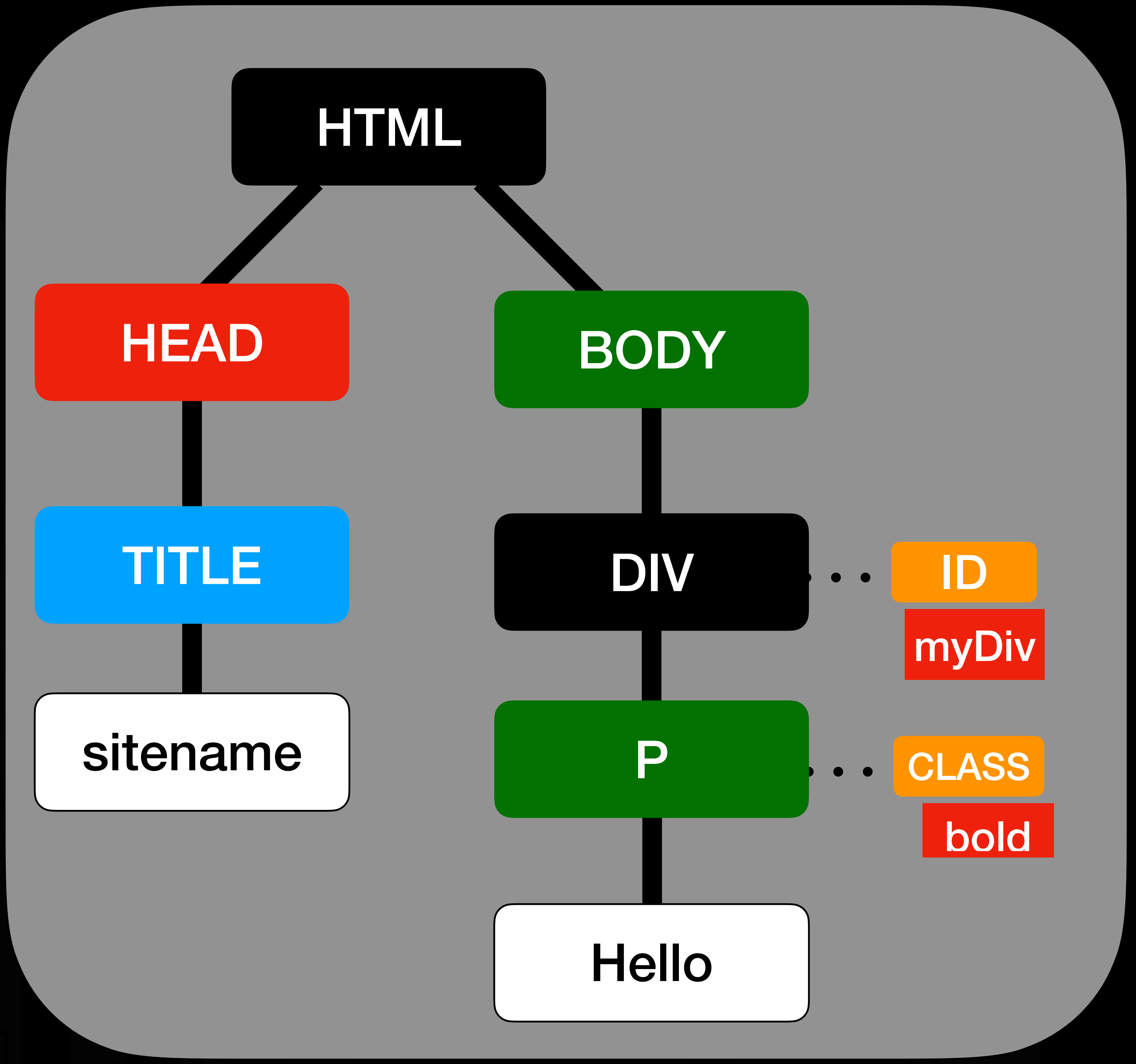
HTML



DOM TREE

document

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
           <p class=“bold”>
            Hello </p>
      </div>
 </body>
</html>
```

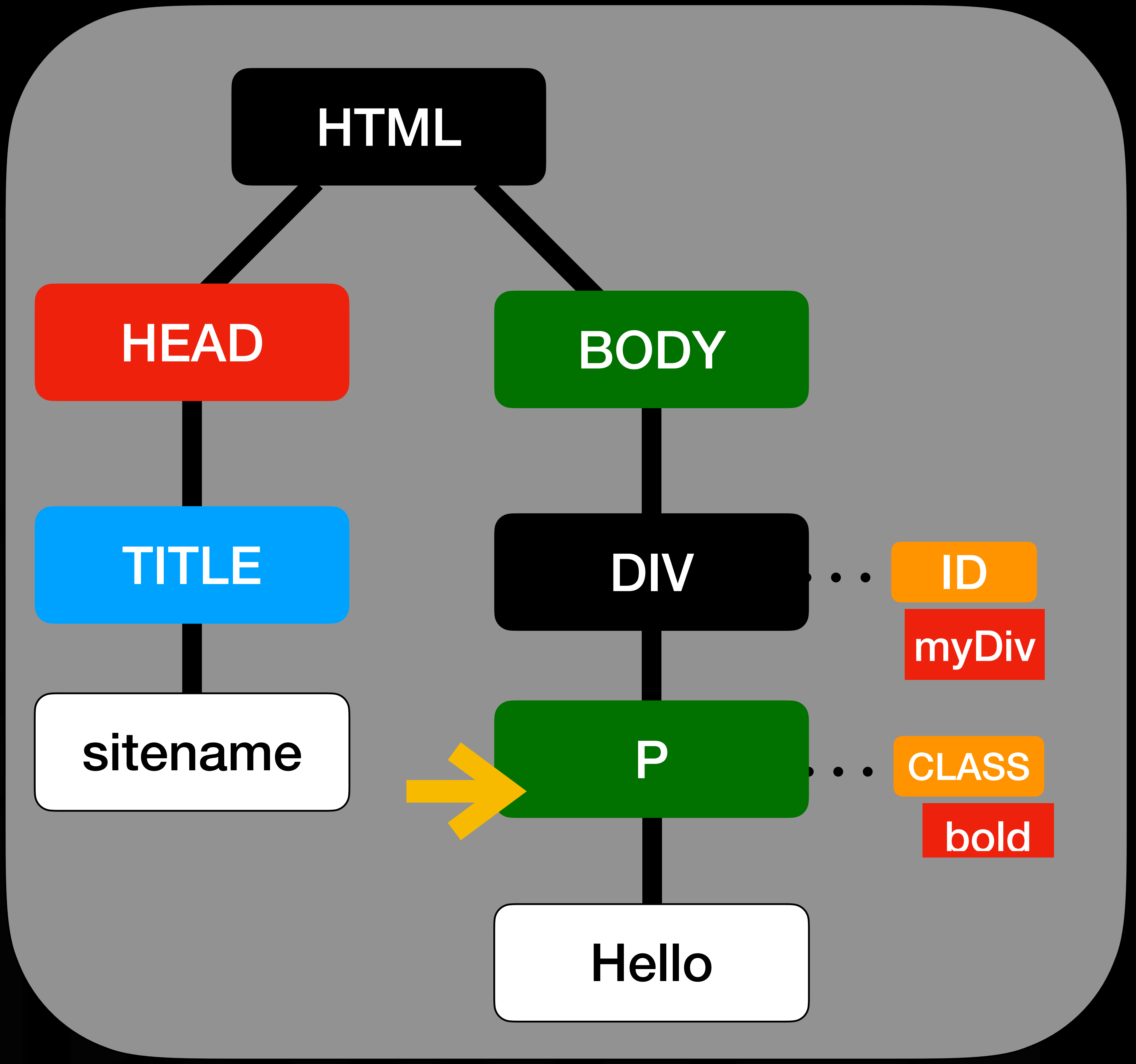


document

```
  document = {
           title : “sitename",
           location :"http://localhost" ,
           getElementById : function(),
           getElementsByClassName : function(),
           getElementsByTagName : function(),
           … … 100 more
       }
```

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
           <p class=“bold”>
            Hello </p>
      </div>
 </body>
</html>
```



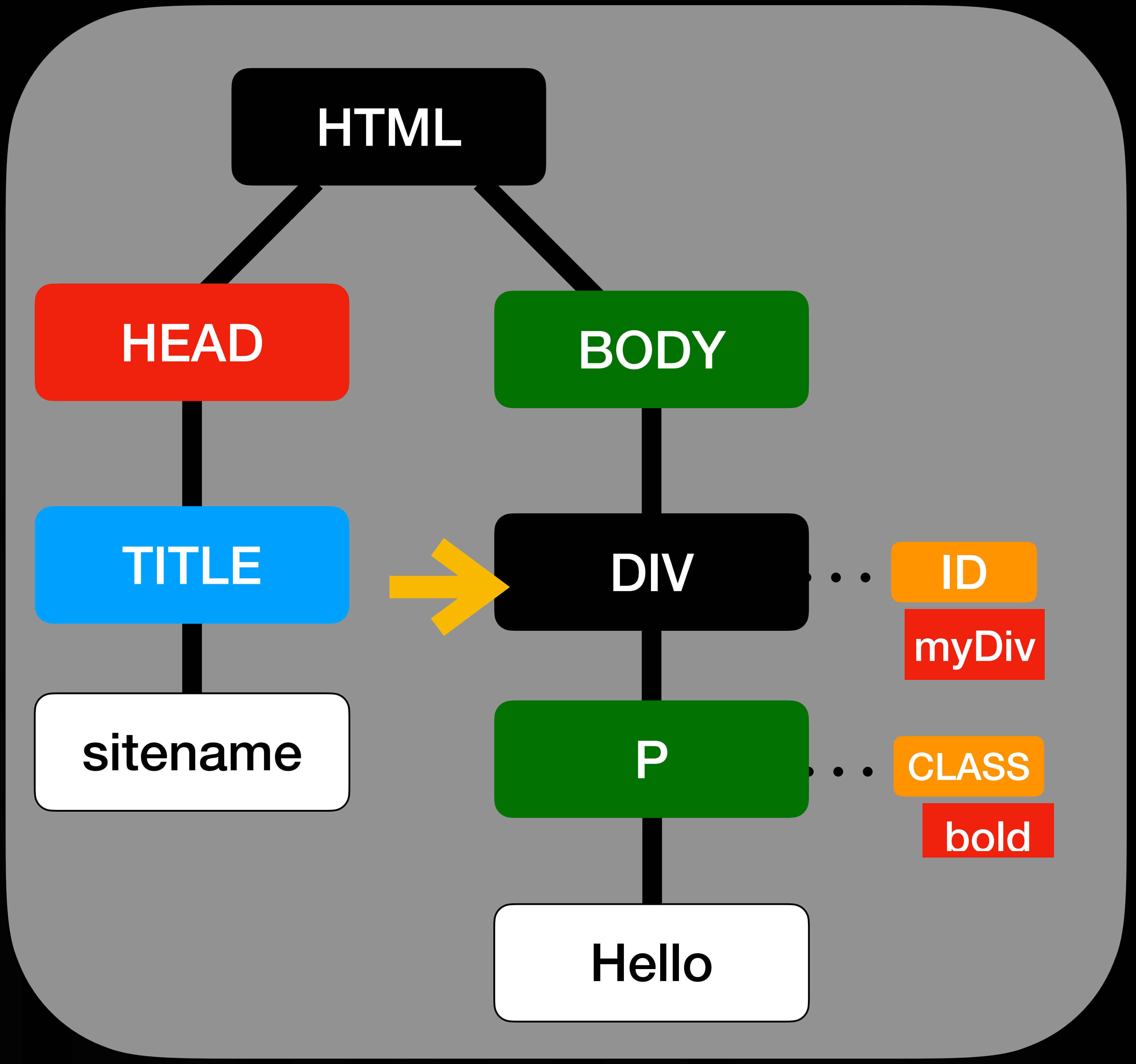
document

document .querySelector (“.bold”)

HTMLElement *JS Object*

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
           <p class=“bold”>
            Hello </p>
      </div>
 </body>
</html>
```



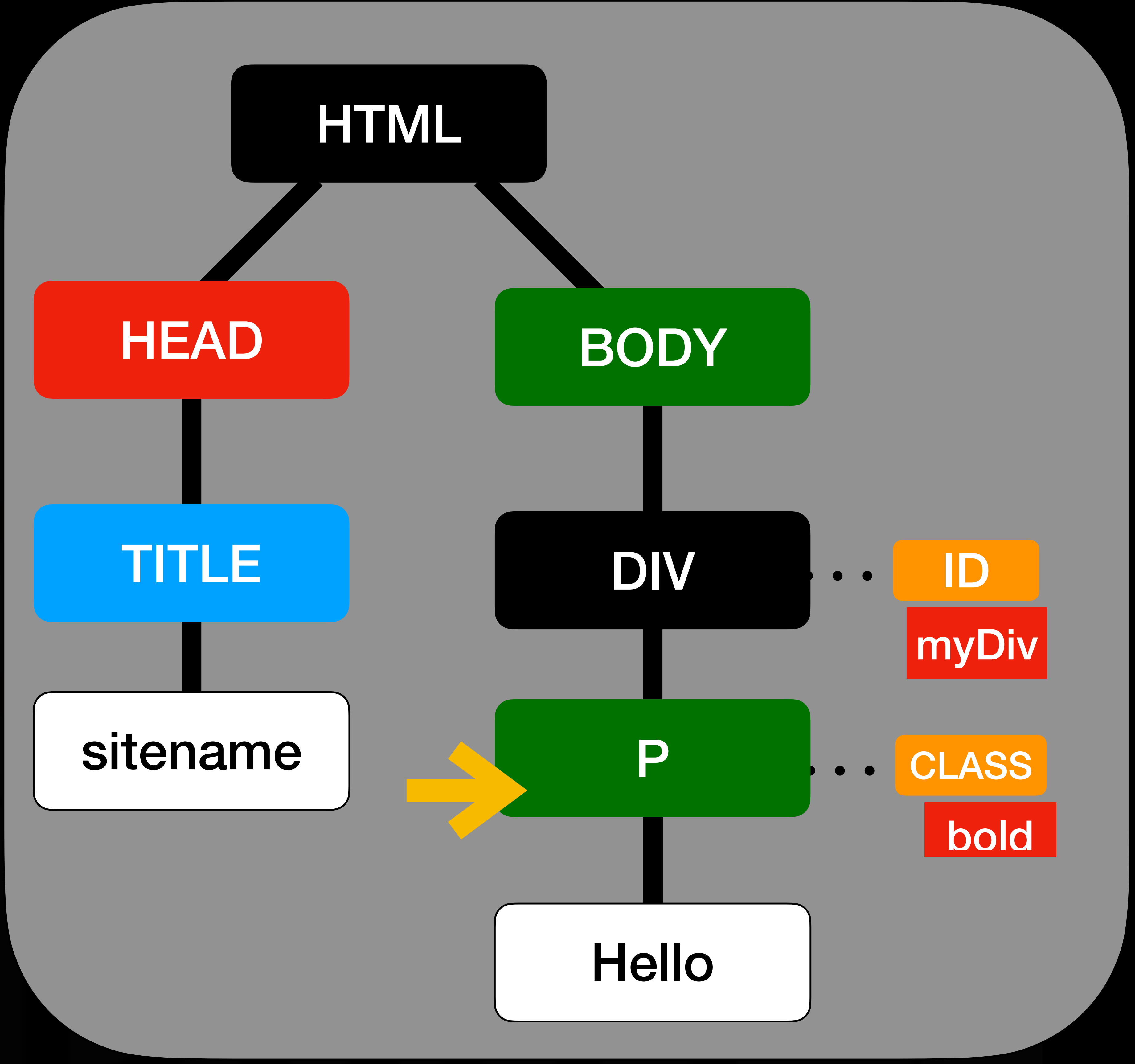
document

document .querySelector (“#myDiv”)

→ HTMLElement

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id="myDiv">
           <p class="bold">
            Hello </p>
      </div>
 </body>
</html>
```



document

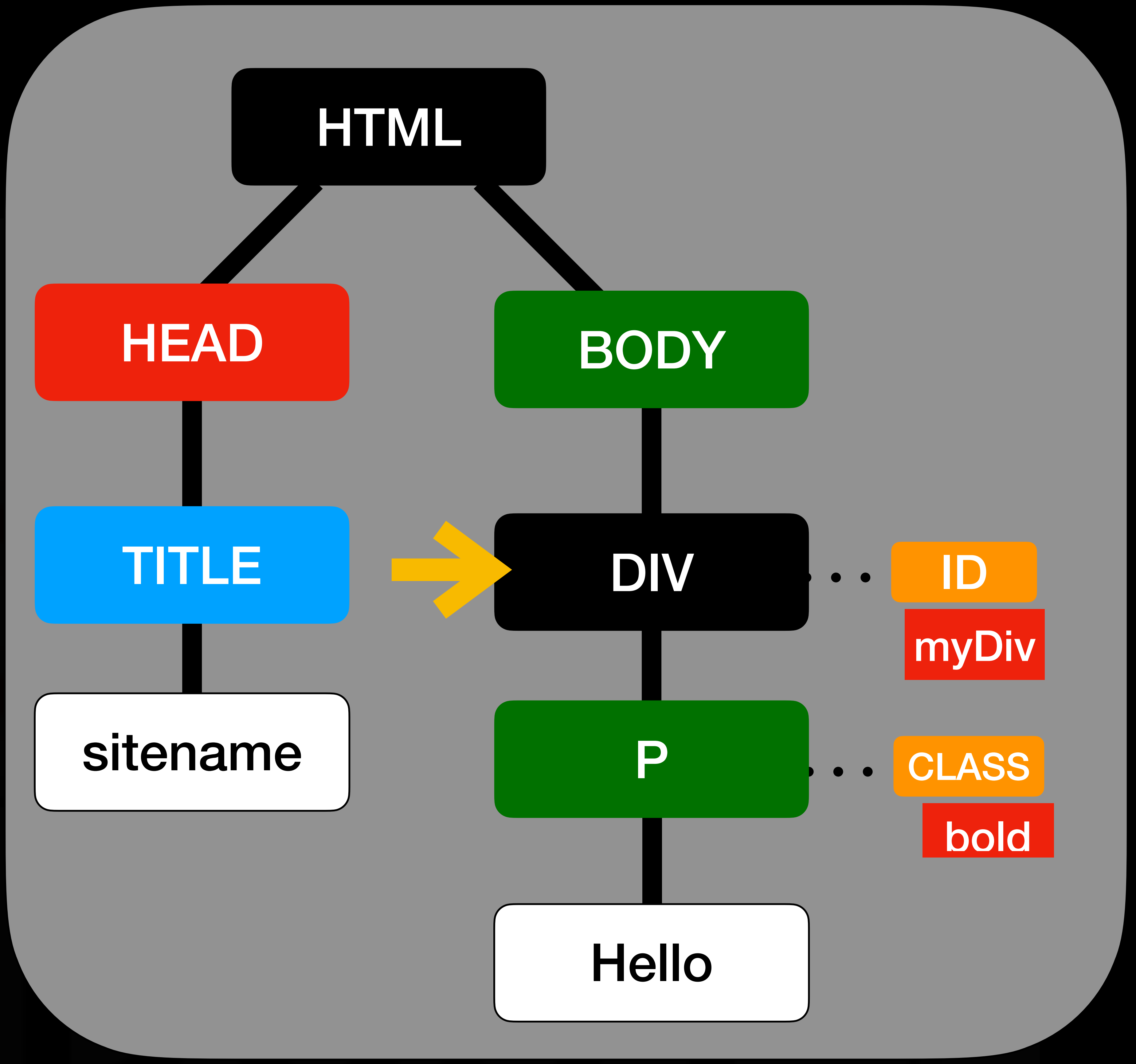
document .querySelectorAll (".bold")

NodeList [p]

[HTMLelememt]

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id="myDiv">
           <p class="bold">
            Hello </p>
      </div>
 </body>
</html>
```



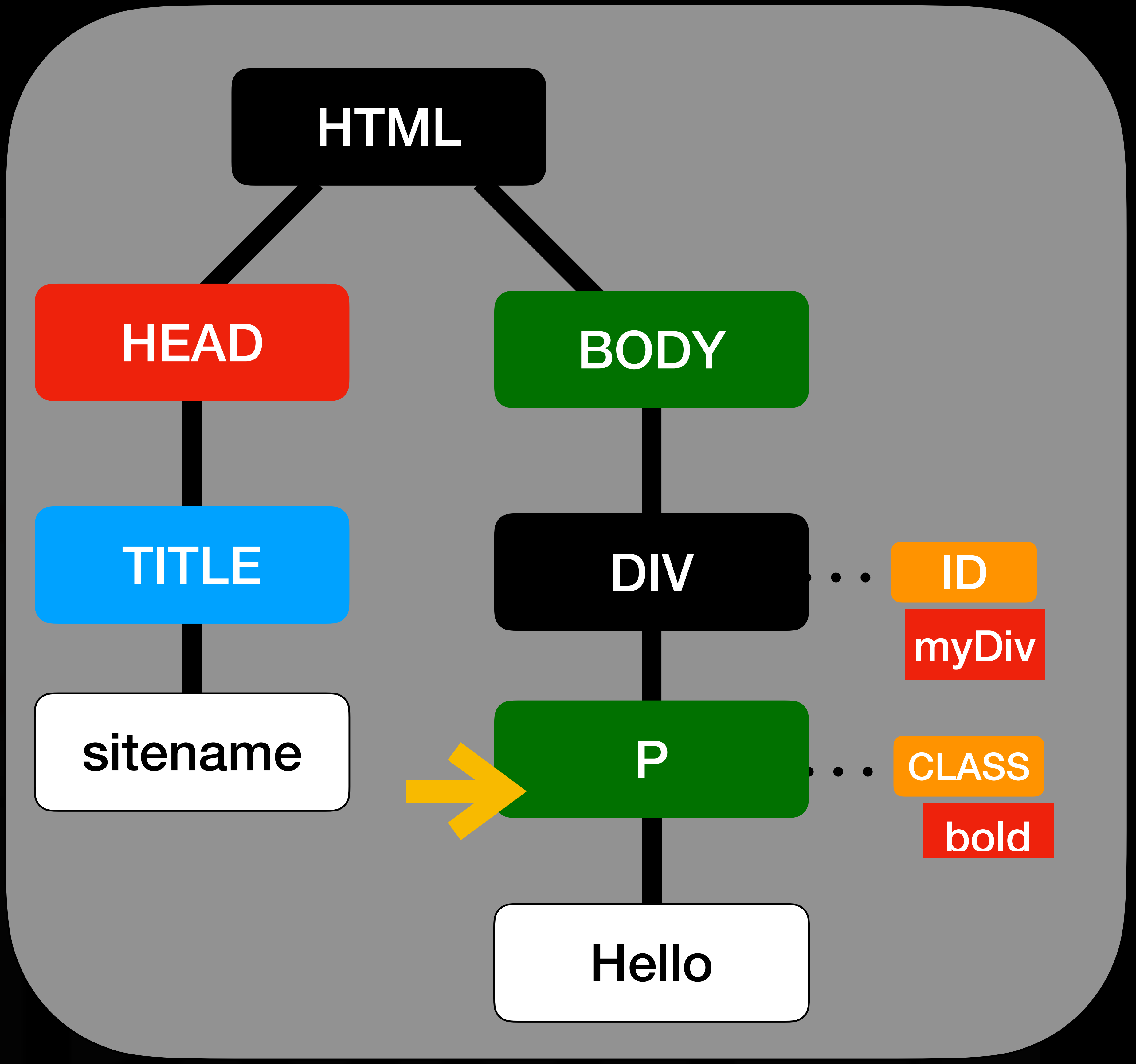
document

document .getElementById ("myDiv")

➡ **HTMLElement**

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
           <p class=“bold”>
            Hello </p>
      </div>
 </body>
</html>
```



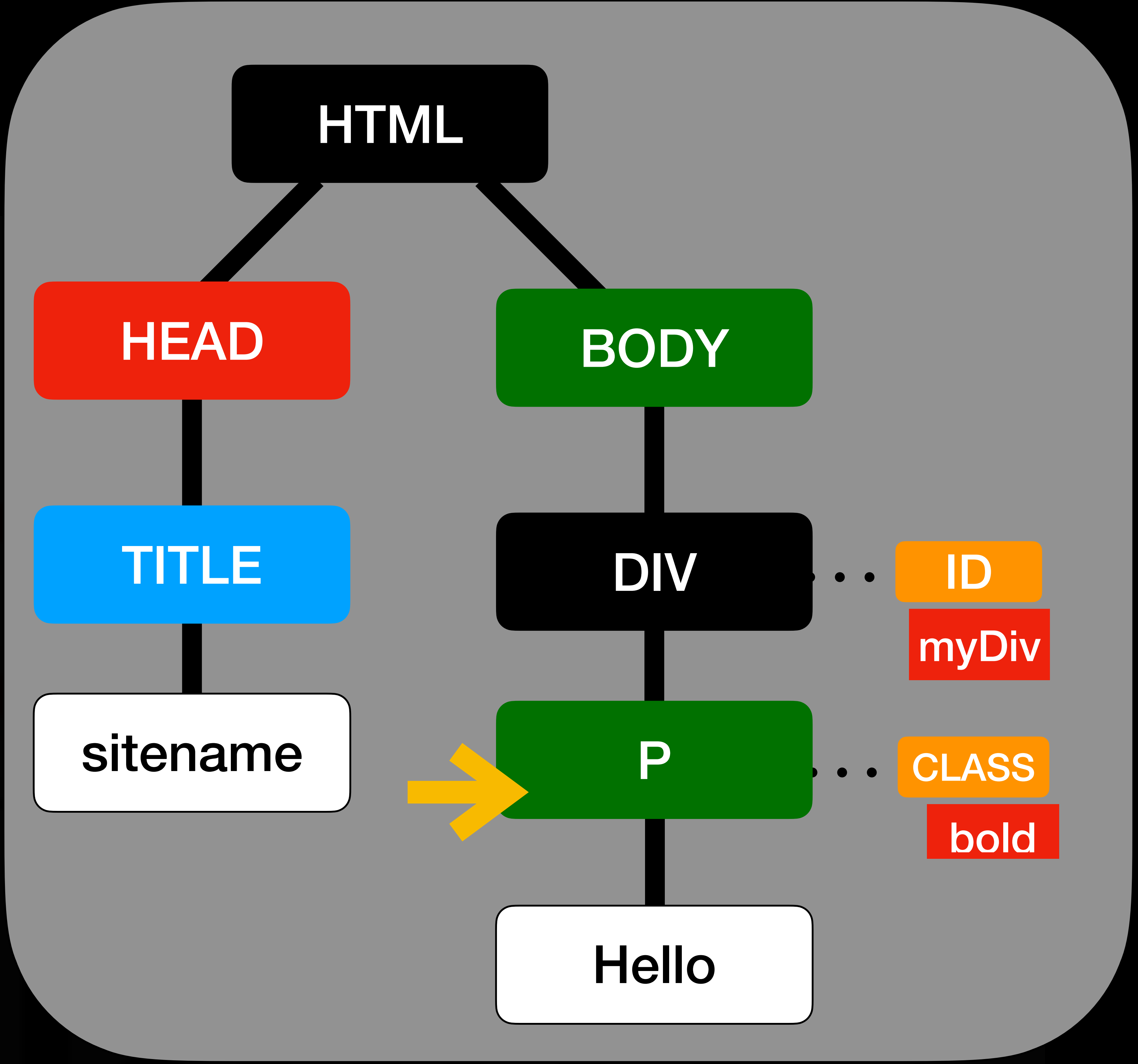
document

document .getElementsByClassName (“bold”)

➡ **HTMLCollection [p]** *[HTMLelememt]*

Select Elements

```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id="myDiv">
           <p class="bold">
            Hello </p>
      </div>
 </body>
</html>
```



document

document .getElementsByTagName ("p")

→ **HTMLCollection [p]** *[HTMLelememt]*

HTMLElement (reading)

const el = document .querySelector (“#myDiv”)

el.innerText → “”

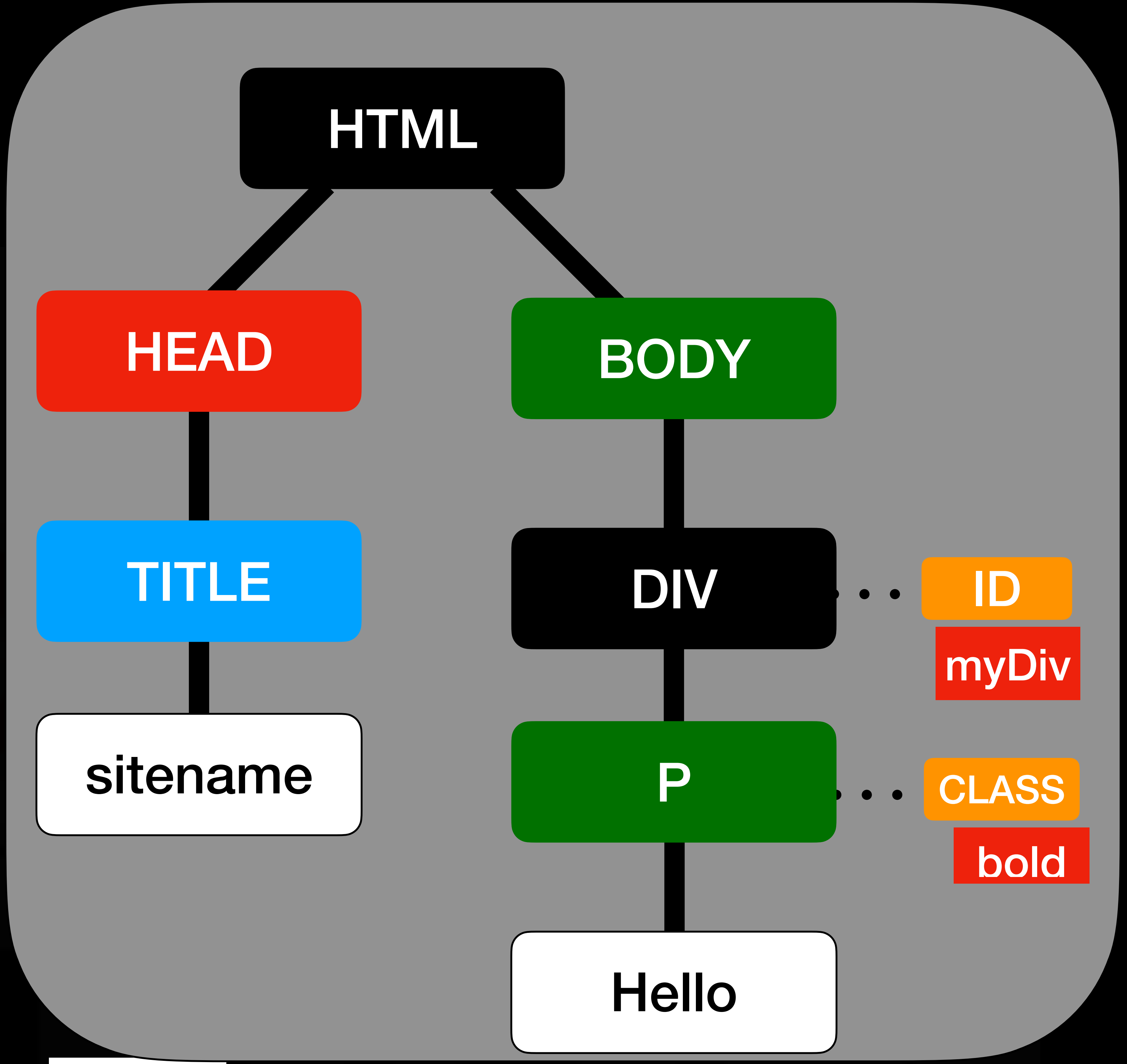
el.innerHTML → `<p class=“bold”> Hello </p>`

el.id → “myDiv”

const el = document .querySelector (“.bold”)

e.className → “bold”

e.innerText → “Hello”



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
           <p class=“bold”>
          Hello </p>
      </div>
 </body>
</html>
```

HTMLElement (writing)

const el = document .querySelector (“#myDiv”)

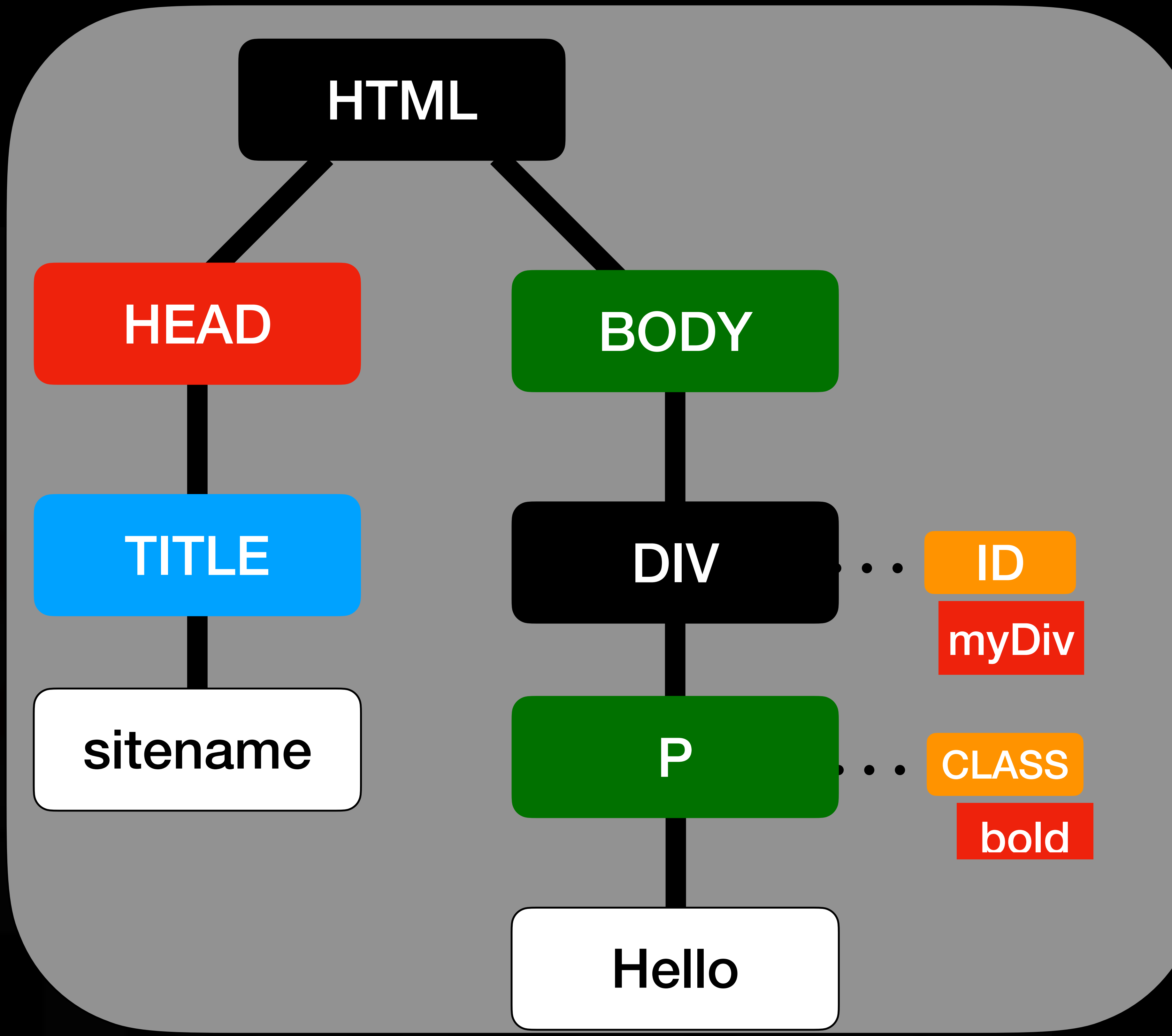
el.innerHTML = `<p class=“bold”> Hey </p>`

el.id = `“myDiv”`

const el = document .querySelector (“.bold”)

e.className = `“bold”`

e.innerText = `“Hello”`

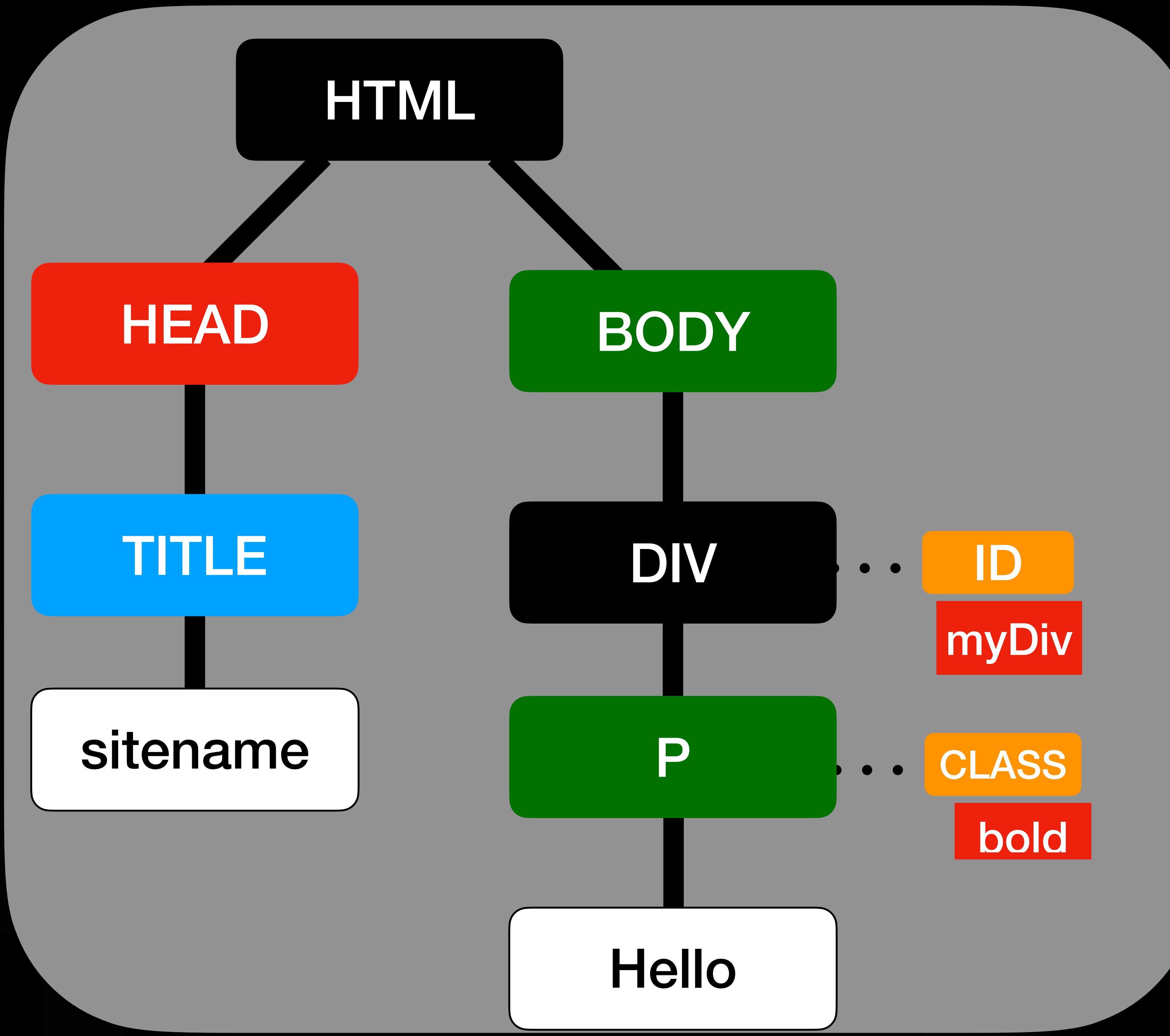


```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
         Hey </p>
      </div>
 </body>
</html>
```

Attributes

```
const el = document .querySelector ( “.bold” )
```

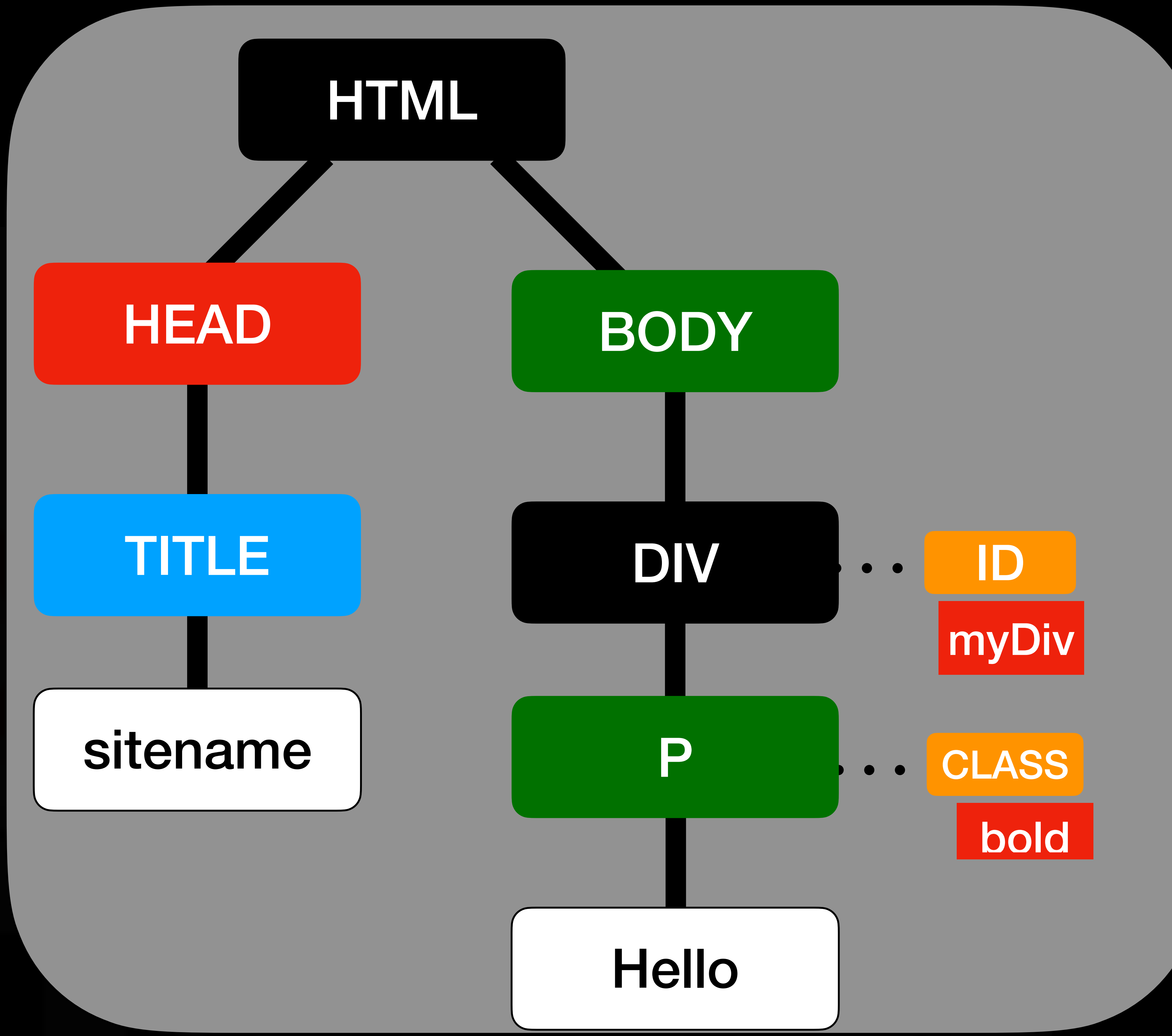
```
el.getAttribute(“class”) → “bold”

el.setAttribute(“class”, “bold dark”)
```



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
         <p class=“bold”>
         Hello </p>
      </div>
 </body>
</html>
```

CSS Styles

```
const el = document .querySelector ( “.bold” )

el.style.color → “black”

el.style.color = “blue”

el.style.backgroundColor = “yellow”

el.style.border = “1px solid red”
```



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
         Hello </p>
      </div>
 </body>
</html>
```

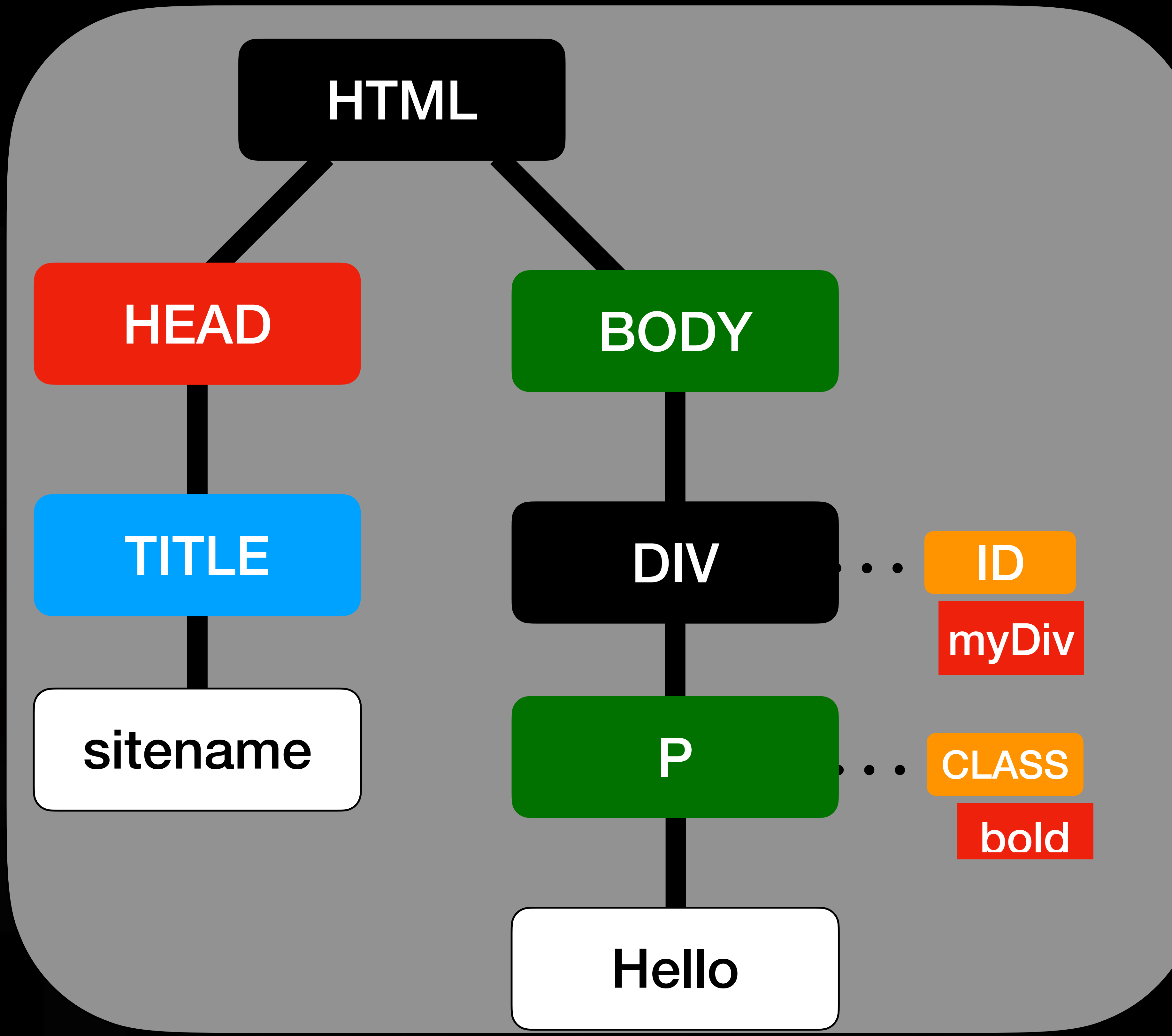
ClassList

```
const el = document .querySelector ( “.bold” )
```

```
el.classList.add(“dark”)
```

```
el.classList.remove(“dark”)
```

```
el.classList.replace(“bold”, “dark”)
```



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
        <p class=“bold”>
       Hello </p>
      </div>
 </body>
</html>
```

Children / Parent / Sibling

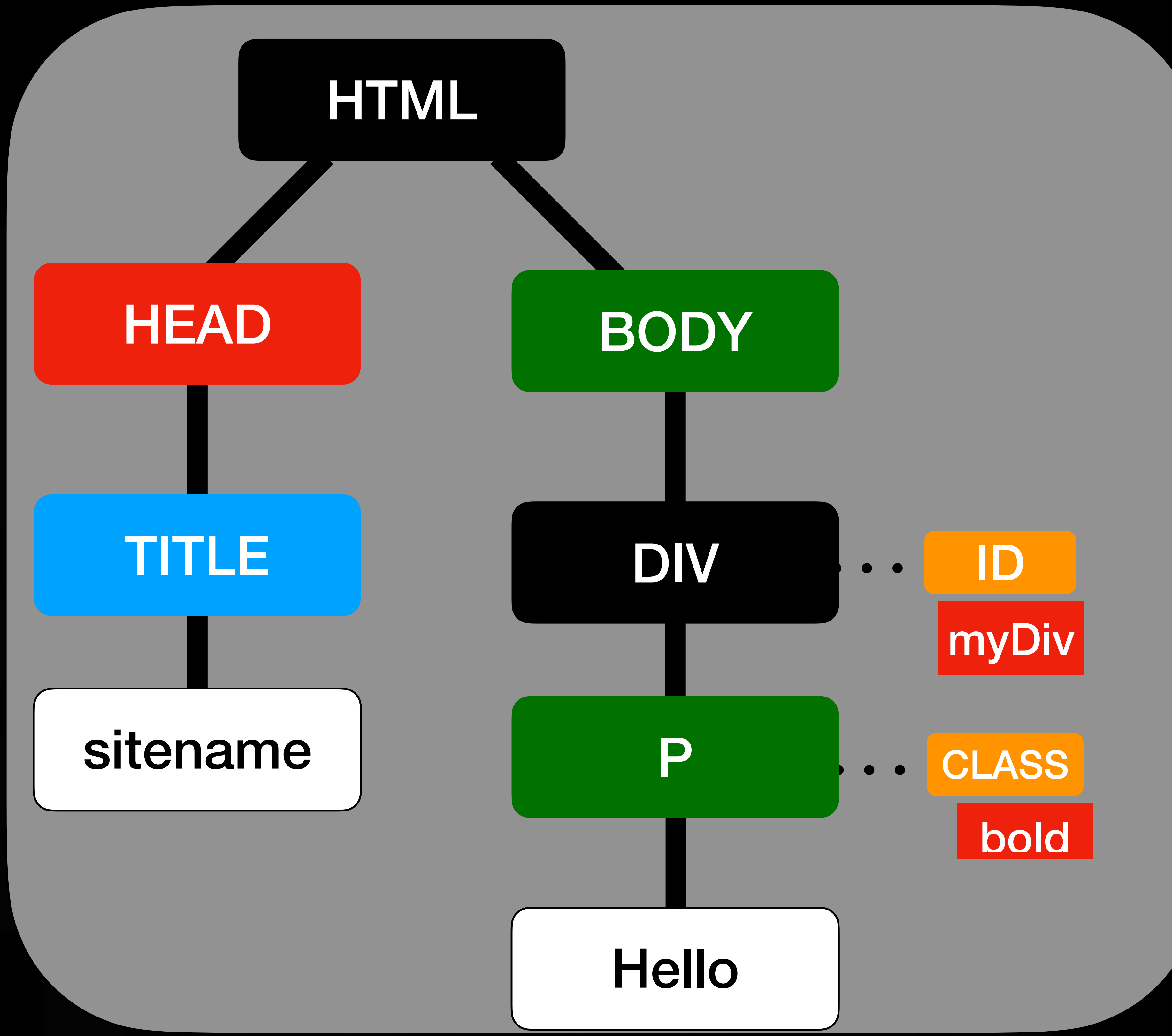
const el = document .querySelector (“#myDiv”)

el.children → P

el.parentElement → Body

el.previousSibling → null

el.nextSibling → null

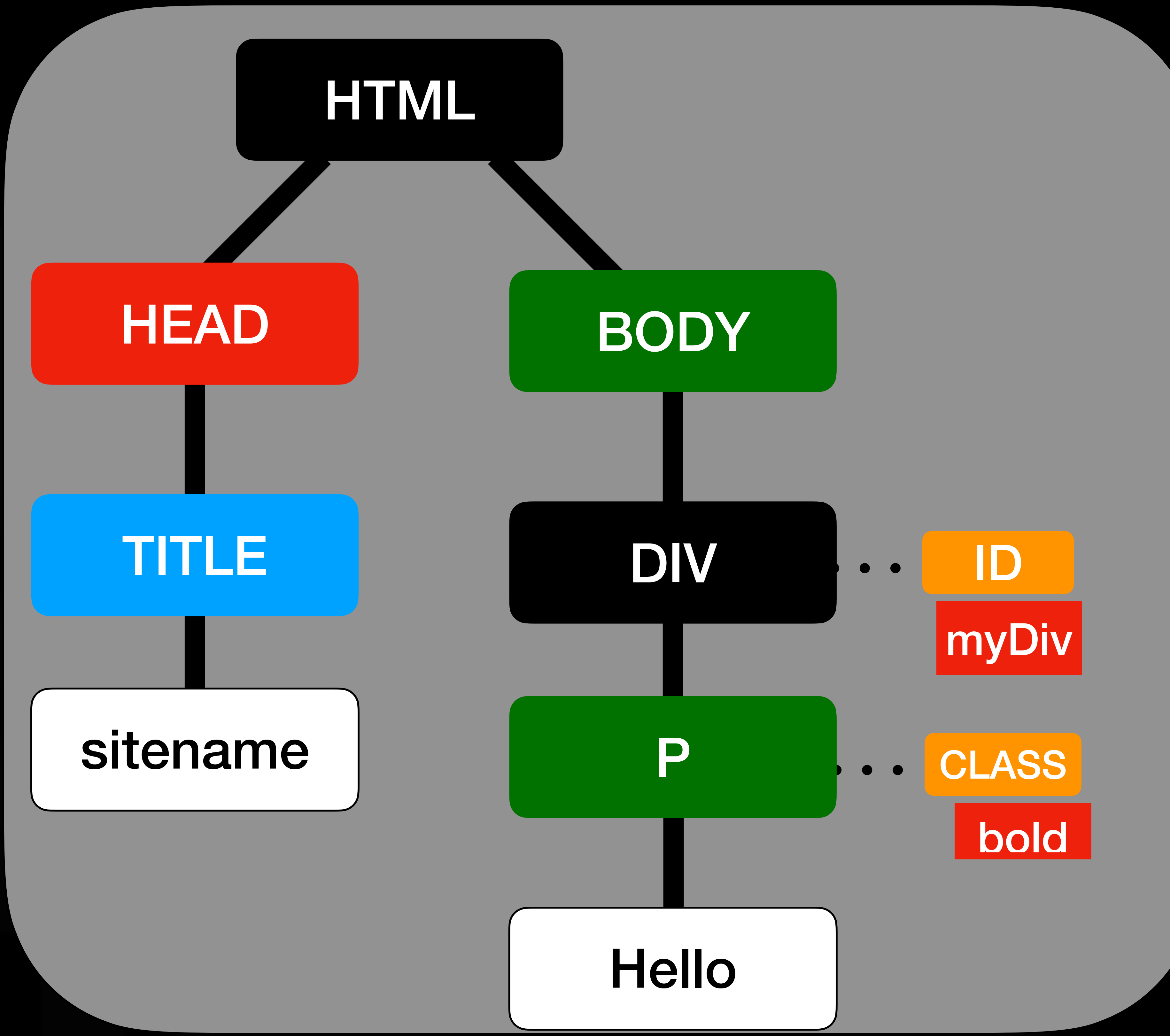


```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
     <div id=“myDiv”>
          <p class=“bold”>
         Hello </p>
     </div>
 </body>
</html>
```

Events

```
const el = document .querySelector ( “.bold” )

el.addEventListener( “click”, function(){

                            } )
```

runs on every click



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
        Hello </p>
      </div>
 </body>
</html>
```

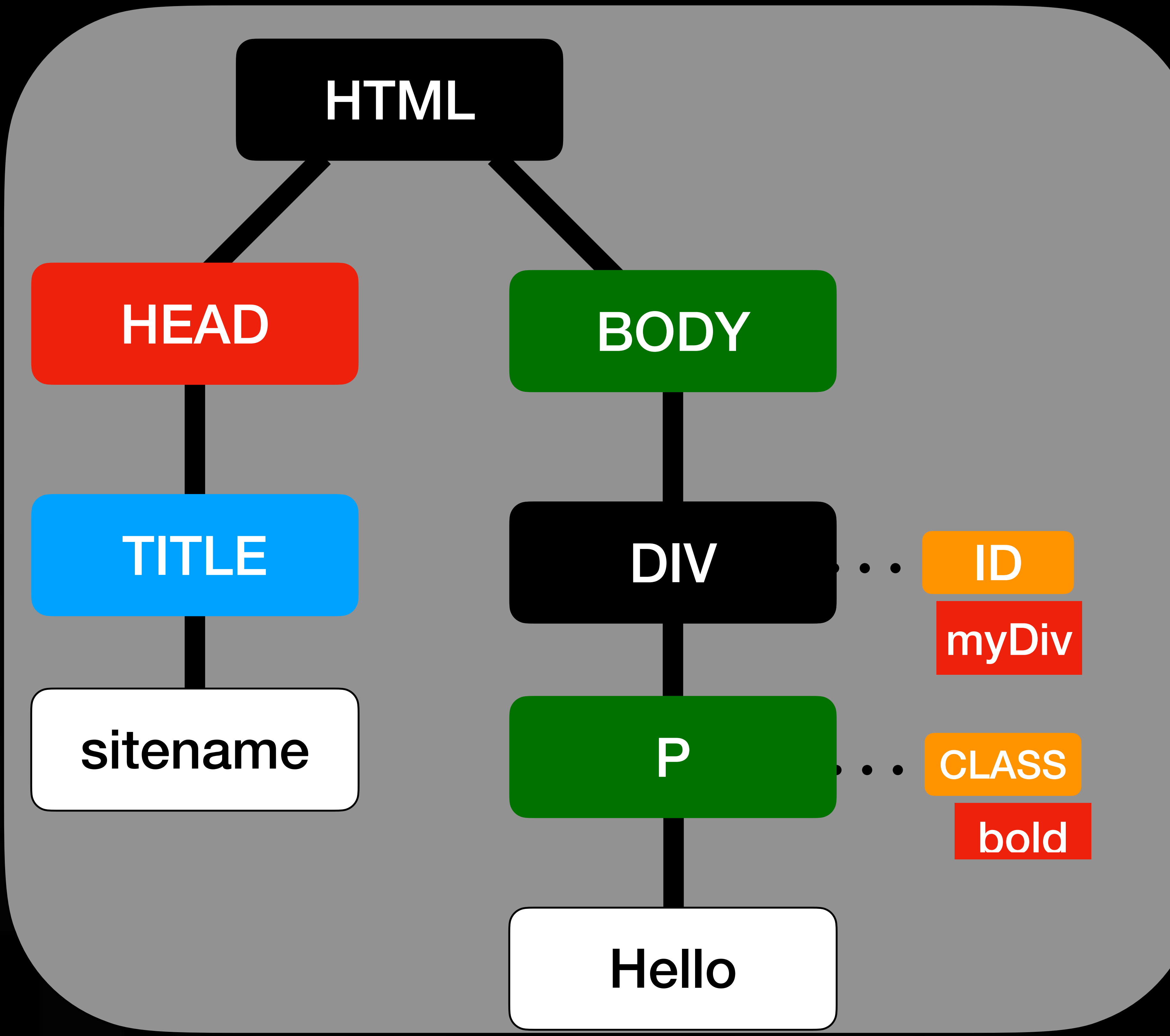
Event Object

```
const el = document .querySelector ( “.bold” )

el.addEventListener( “click”, function(e){

                   } )
```

event Object

```
e.target.innerText
```

target = element



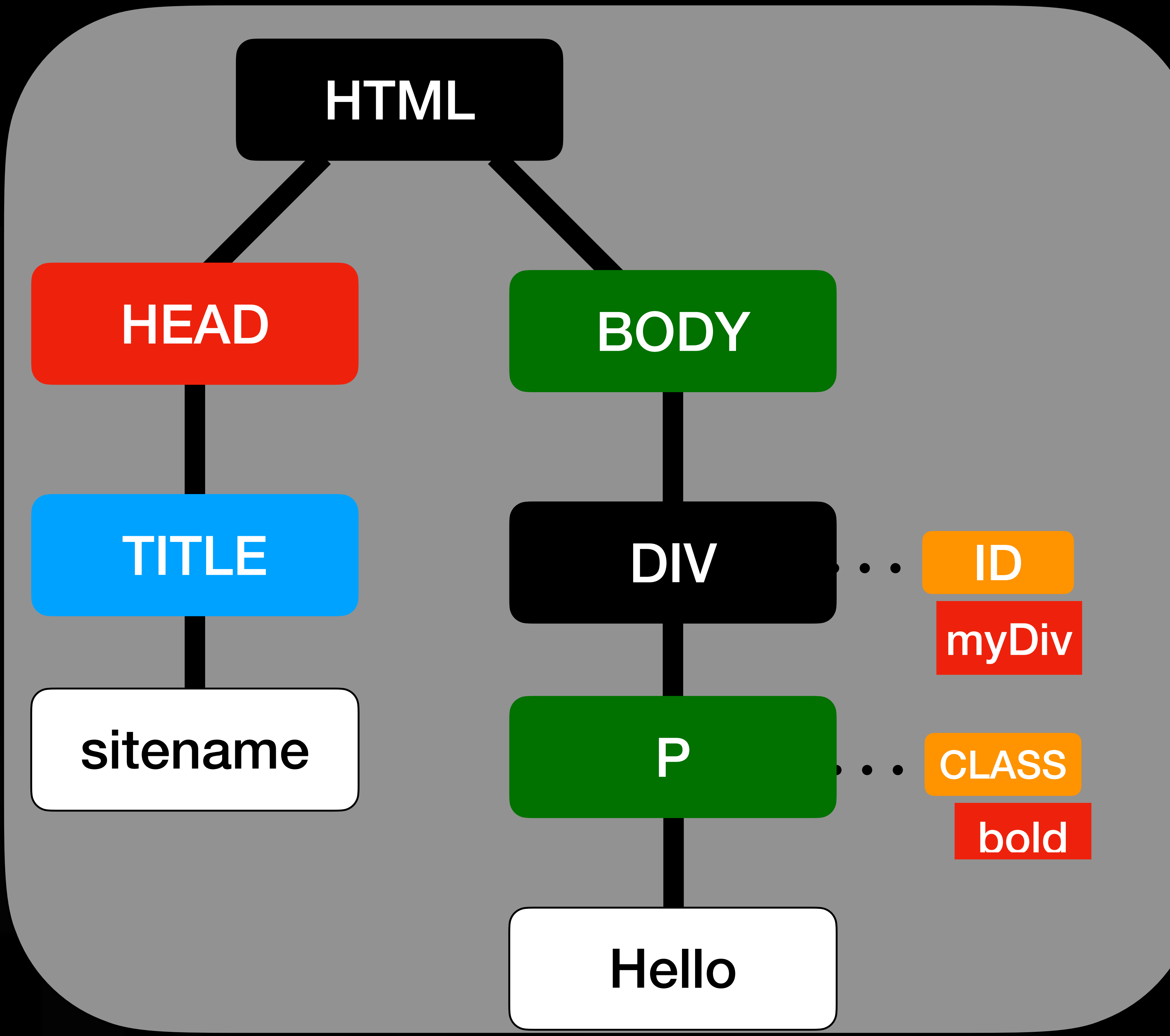
```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
         Hello </p>
      </div>
 </body>
</html>
```

Add Elements

```
const el = document .querySelector ( “.bold” )

const child = document.createChild(‘span’)

el.appendChild(child)

el.prependChild(child)
```

add element after

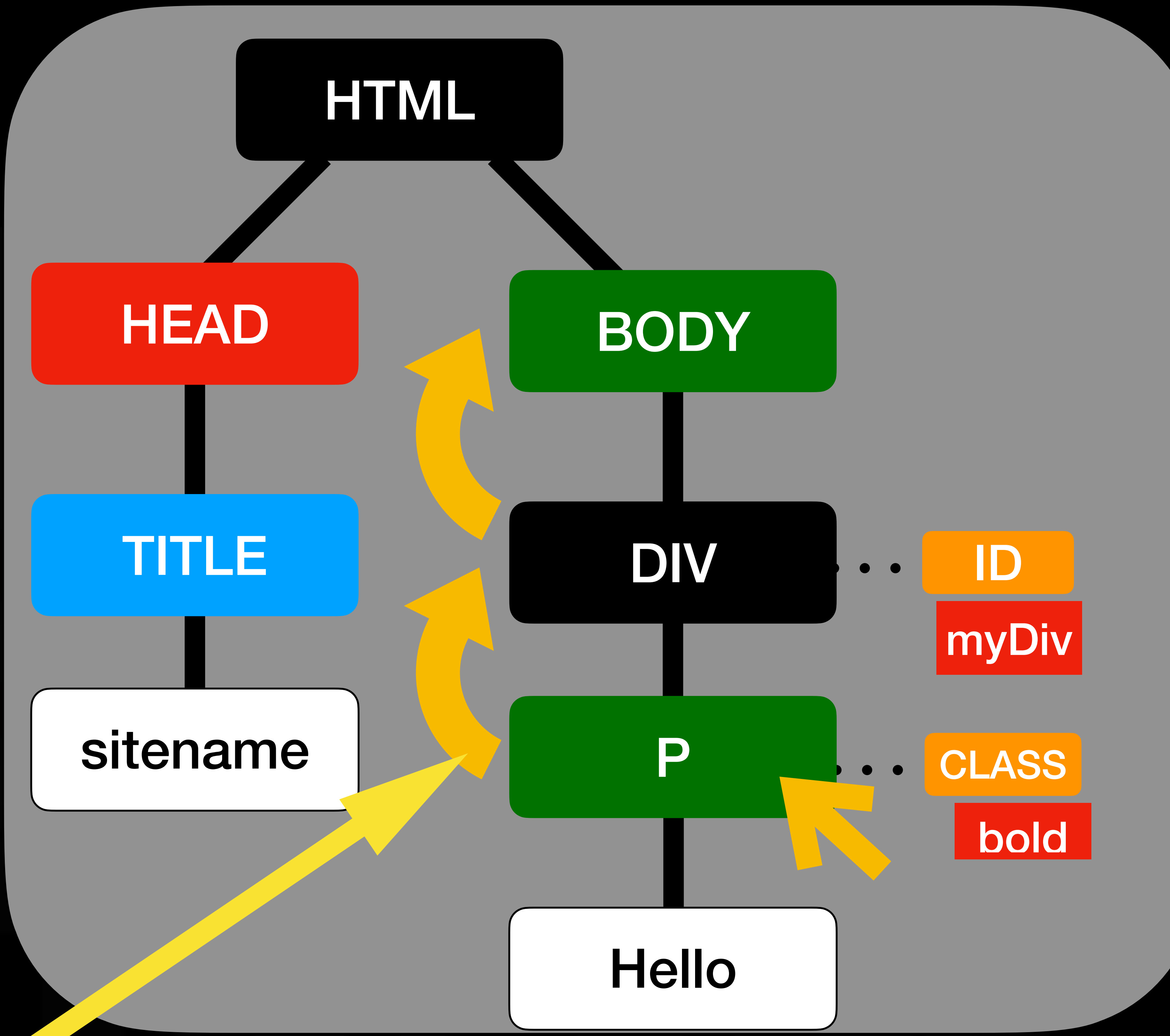
add element before



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
         <p class=“bold”>
        Hello </p>
      </div>
 </body>
</html>
```

Event Bubbling

```
const body = document .querySelector ( “body” )
```



“click” started here, and it bubbles “up”
P => Div => Body

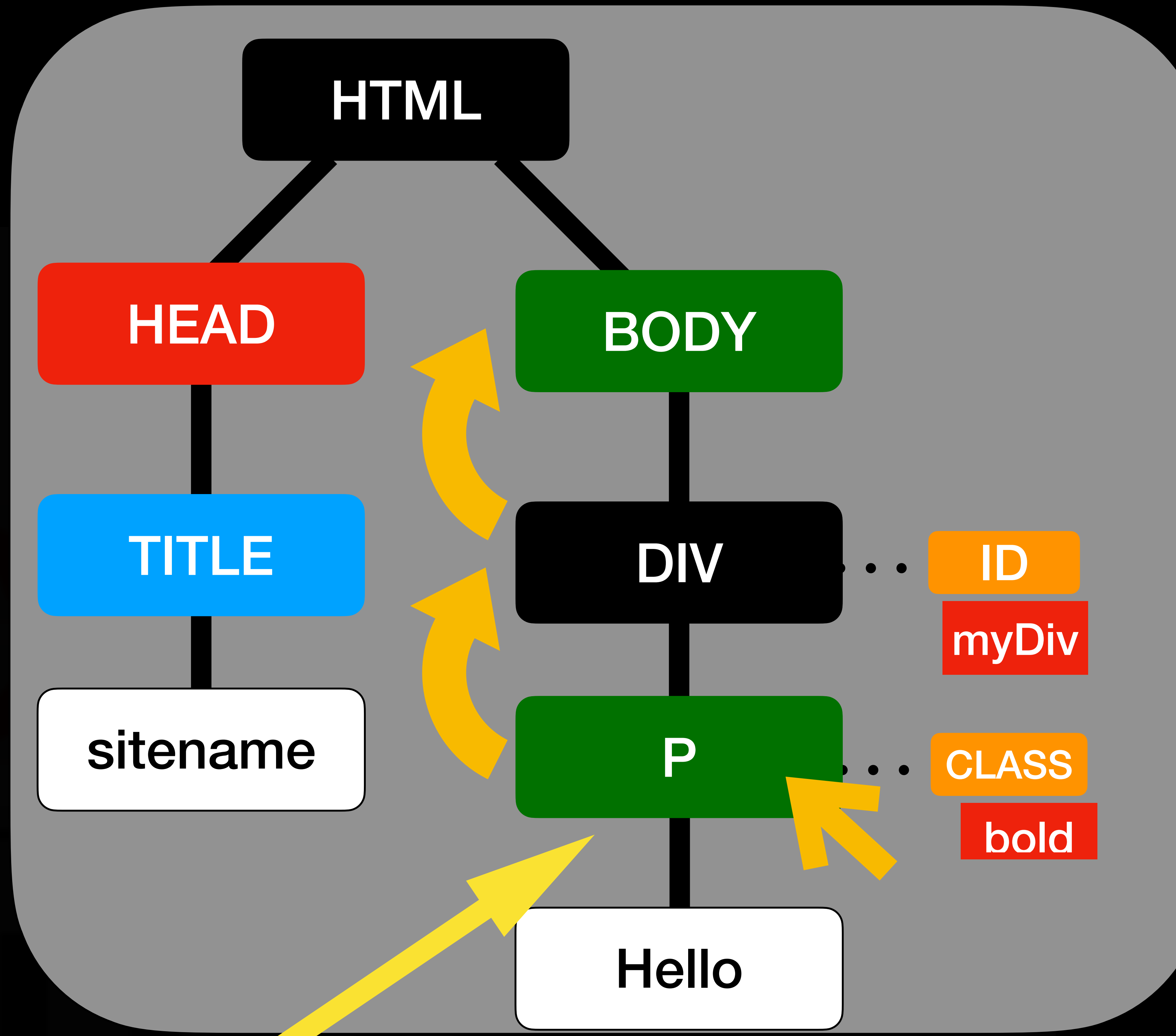
```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
         Hello </p>
      </div>
 </body>
</html>
```

Event Delegation

```
const body = document .querySelector ( “body” )

body.addEventListener( “click”, function(e){

                    } )
```

“body” will also capture “click” / we can delegate “events” to body

“click” started here, and it bubbles “up”
P => Div => Body



```
<html>
<head>
   <title>sitename<title>
</head>
 <body>
      <div id=“myDiv”>
          <p class=“bold”>
         Hello </p>
      </div>
 </body>
</html>
```

Mouse Events

mousedown event

mouseenter event

mouseleave event

mousemove event

mouseout event

mouseover event

mouseup event

click event

dblclick event

Keyboard Events

keyup event

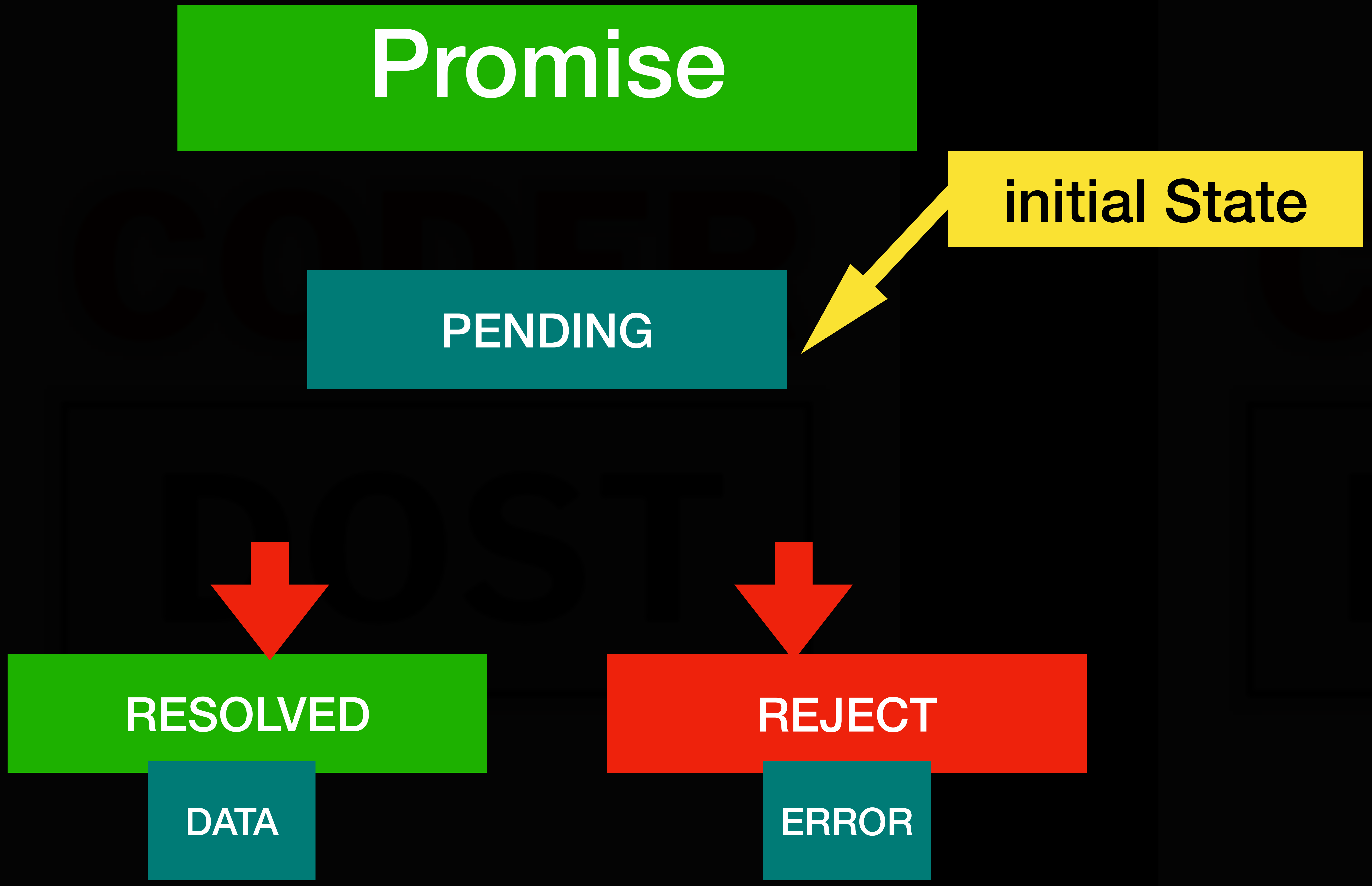
keydown event

keypress event

Document Events

scroll event
copy event
cut event
paste event

Promise : States



Promise

initial State

PENDING

RESOLVED

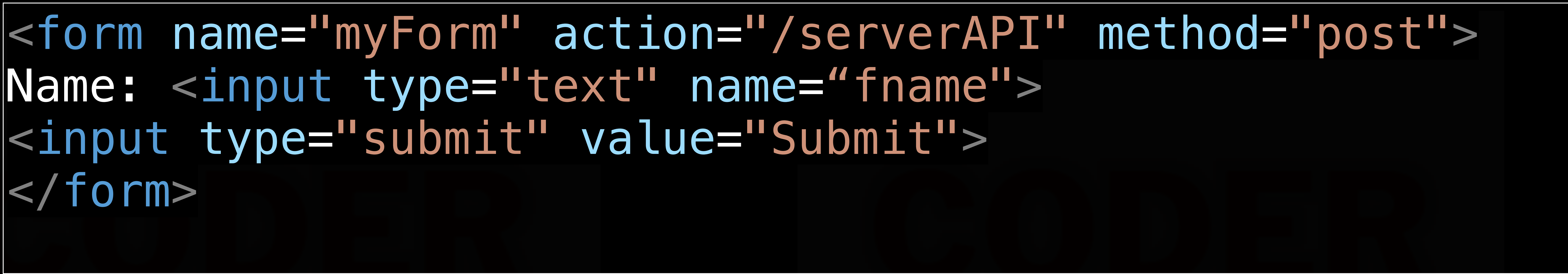
DATA

REJECT

ERROR

6. DOM forms

Forms



```
<form name="myForm" action="/serverAPI" method="post">
Name: <input type="text" name=“fname">
<input type="submit" value="Submit">
</form>
```

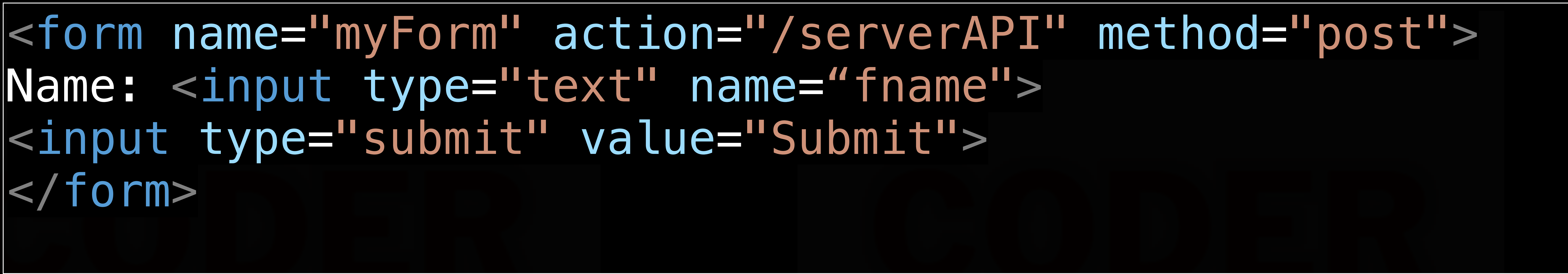
```
const form = document.querySelector ( “form” )

const nameInput = form.fname

el.addEventListener( “submit”, function(e){
        nameInput.value
                    })
```

access inputs by “name” attribute

value of input

event on submit button click

Validate Forms



```
<form name="myForm" action="/serverAPI" method="post">
Name: <input type="text" name=“fname">
<input type="submit" value="Submit">
</form>
```

```
const form = document.querySelector ( “form” )
const nameInput = form.fname
const regexPattern = /^(?:\d{3})([-/.])\d{3}\1\d{4}$/

el.addEventListener( “submit”, function(e){

 const result = regexPattern.test(nameInput.value)


})
```

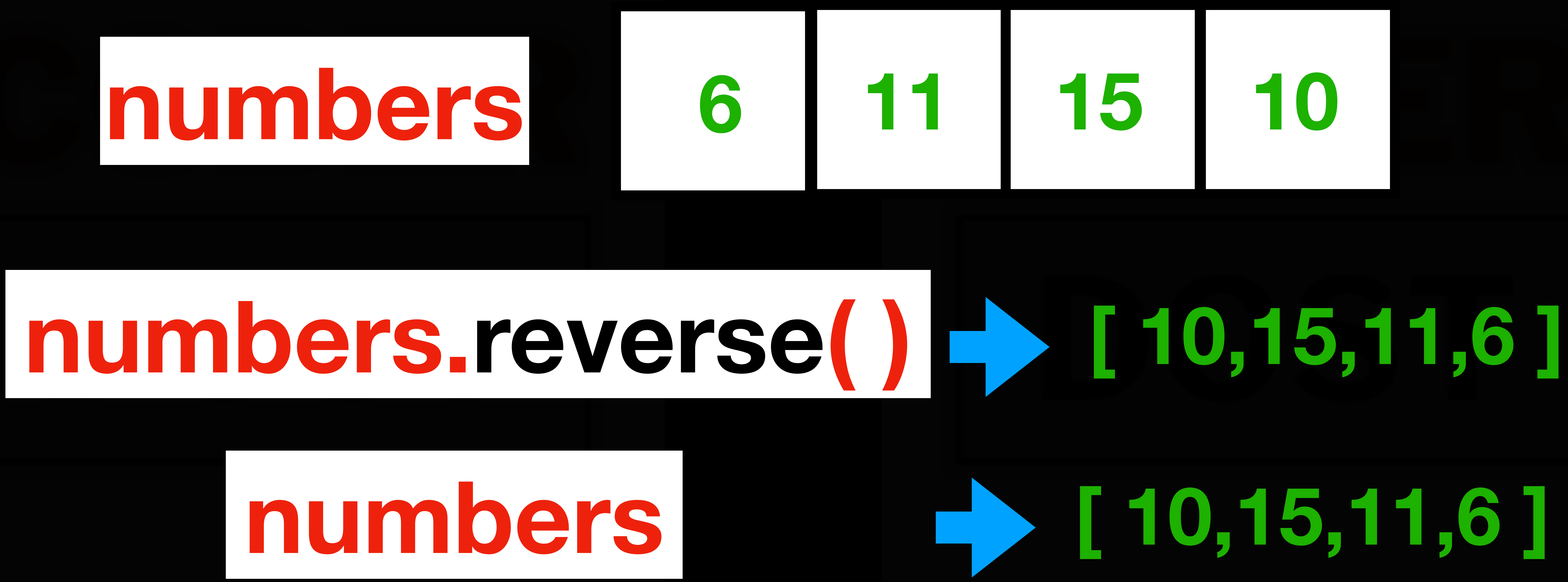
pattern for phone 111-222-333

BOM- Browser Object Model

```
  window = {

          location :Location object ,
          document : DOM Object,
          alert : function(),
          confirm : function(),
          scrollX : 0,
          scrollY : 100px,
          innerWidth : 900px,
          innerHeight : 900px,
          open : function(),
          close : function(),
          scrollTo: function()

          … … 100 more
      }
```

7. Arrays

REVERSE method



numbers

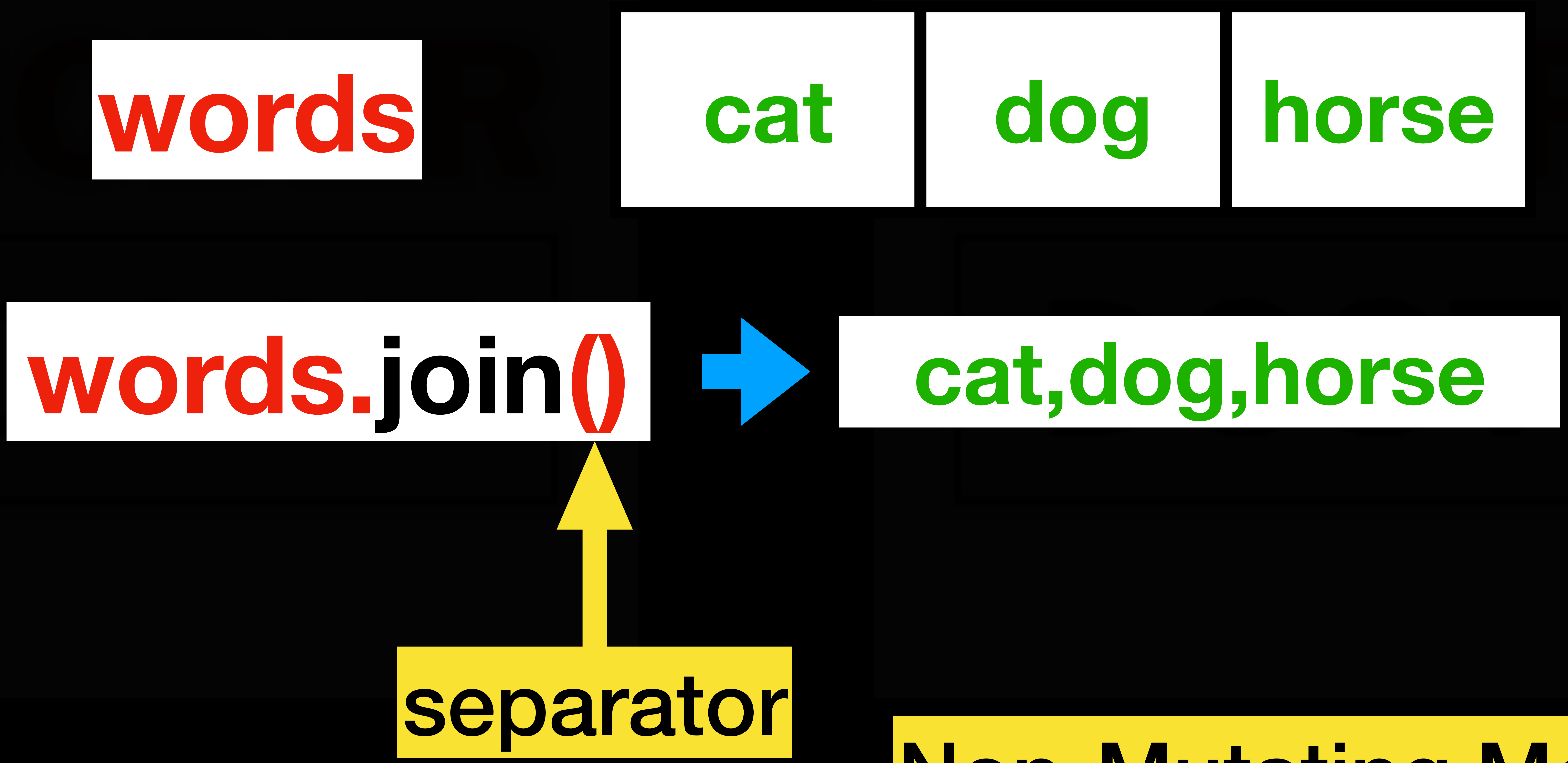
6	11	15	10

numbers.reverse( ) → [10,15,11,6]

numbers → [10,15,11,6]

Mutating Method

JOIN function



words

cat

dog

horse

words.join()

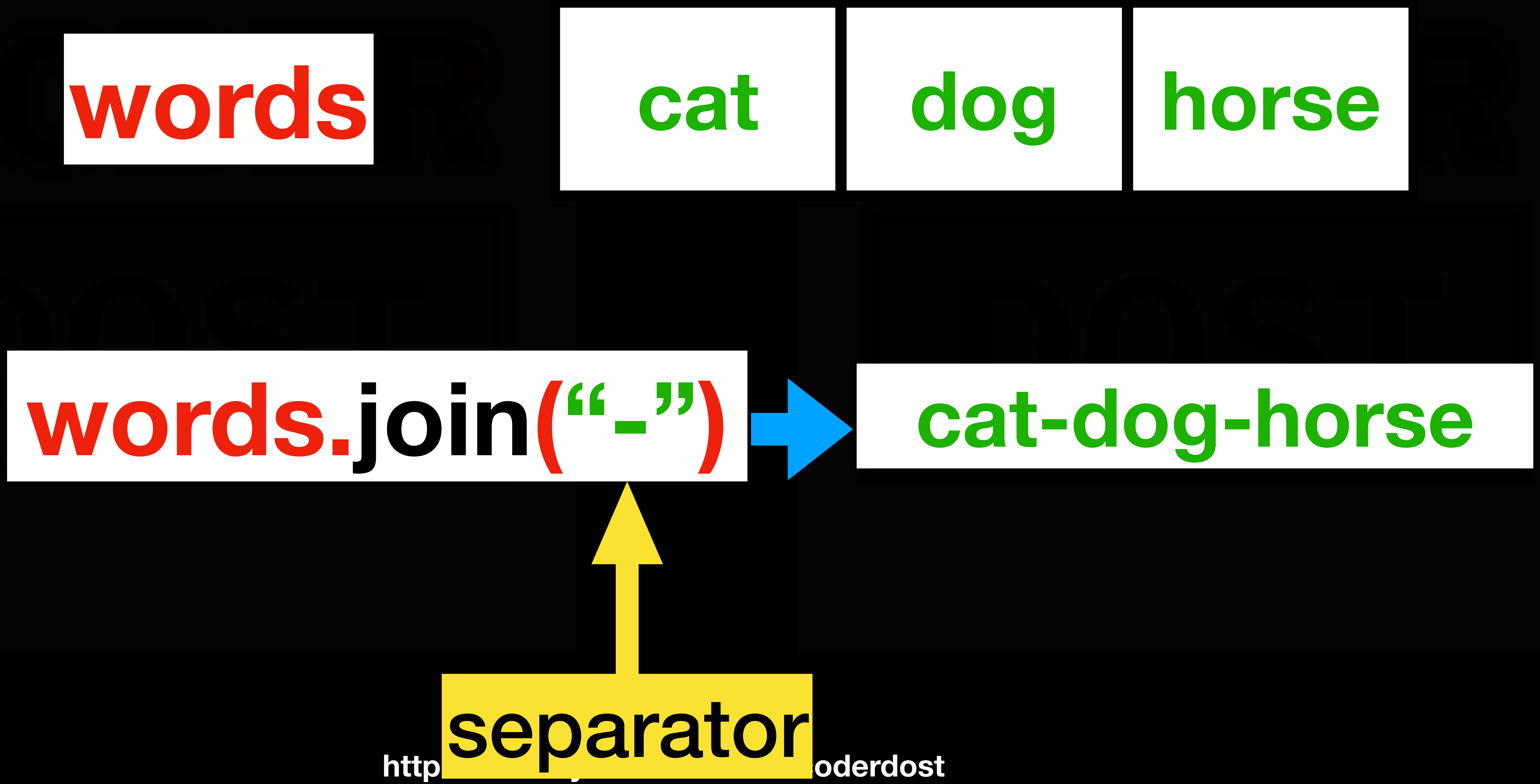
cat,dog,horse

separator

Non-Mutating Method

https://www.youtube.com/@c

JOIN function



words

cat

dog

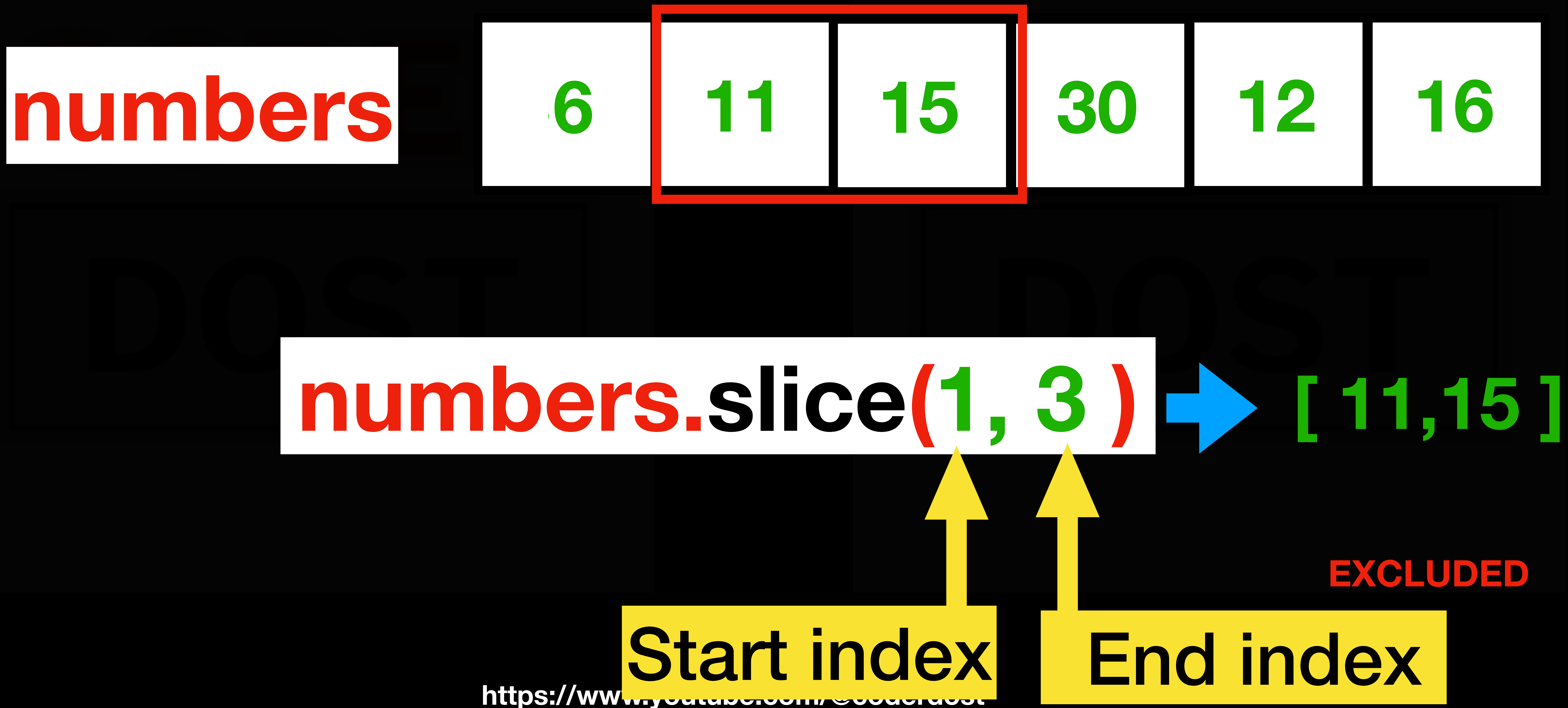
horse

words.join(“-”)

cat-dog-horse

separator

SLICE function



numbers	6	11	15	30	12	16

```
numbers.slice(1, 3 )
```

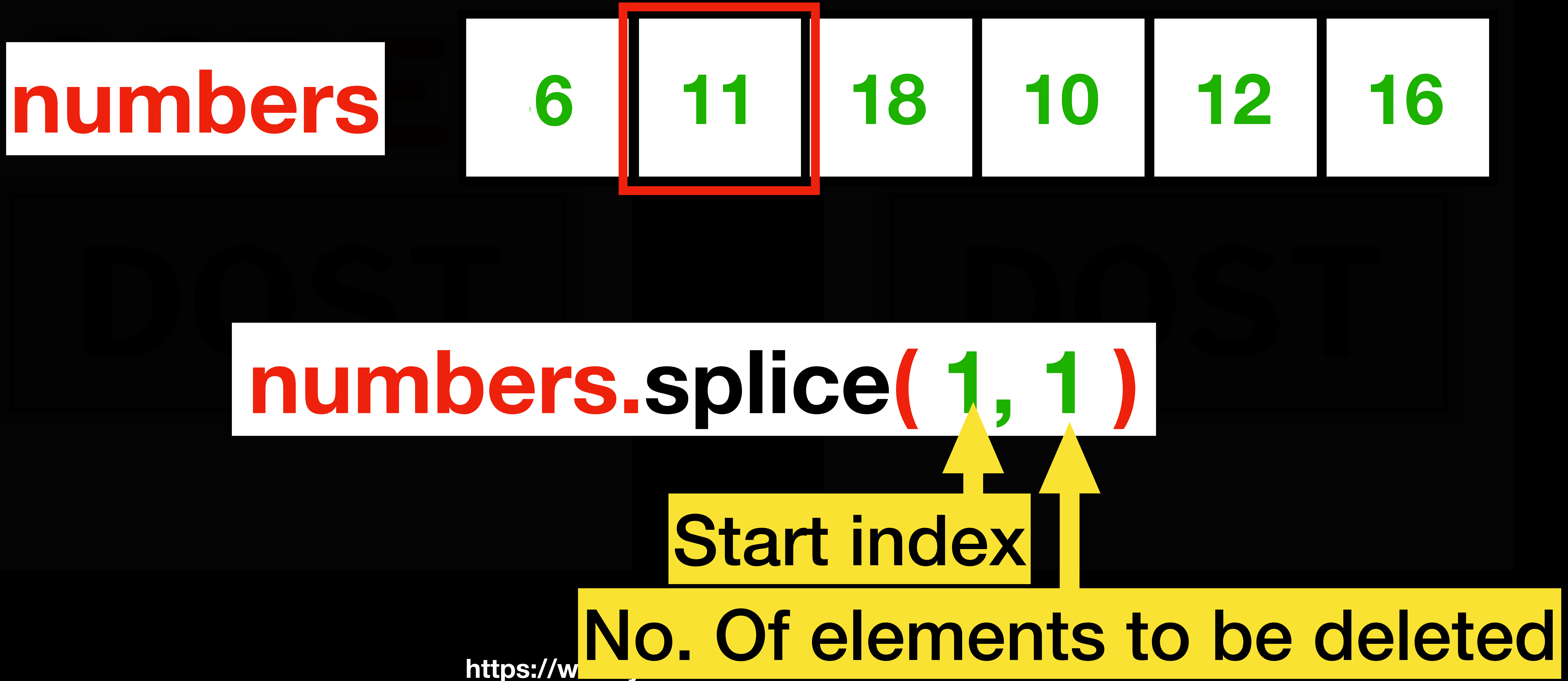
[11,15]

EXCLUDED

Start index

End index

SPLICE function



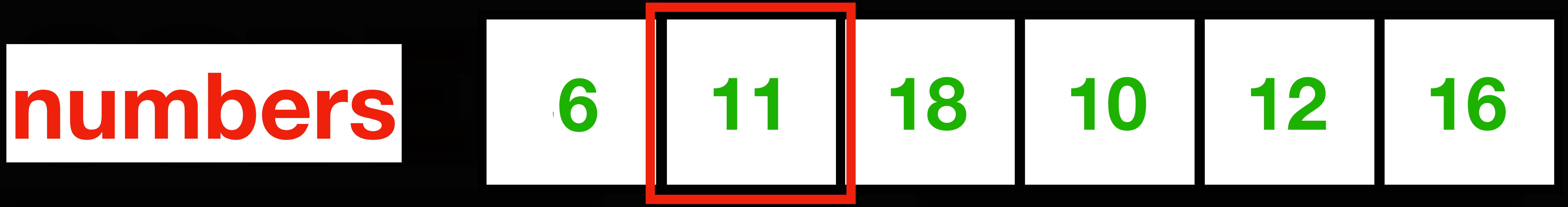
numbers	6	11	18	10	12	16

```
numbers.splice( 1, 1 )
```

Start index

No. Of elements to be deleted

SPLICE function



numbers

6	11	18	10	12	16

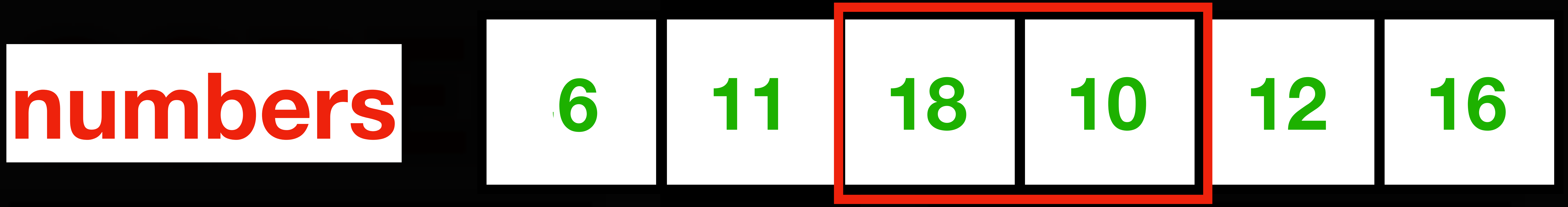
numbers.splice(1, 1) → [11]

numbers → [6,18,10,12,16]

https://www.youtube.com/@coderdost

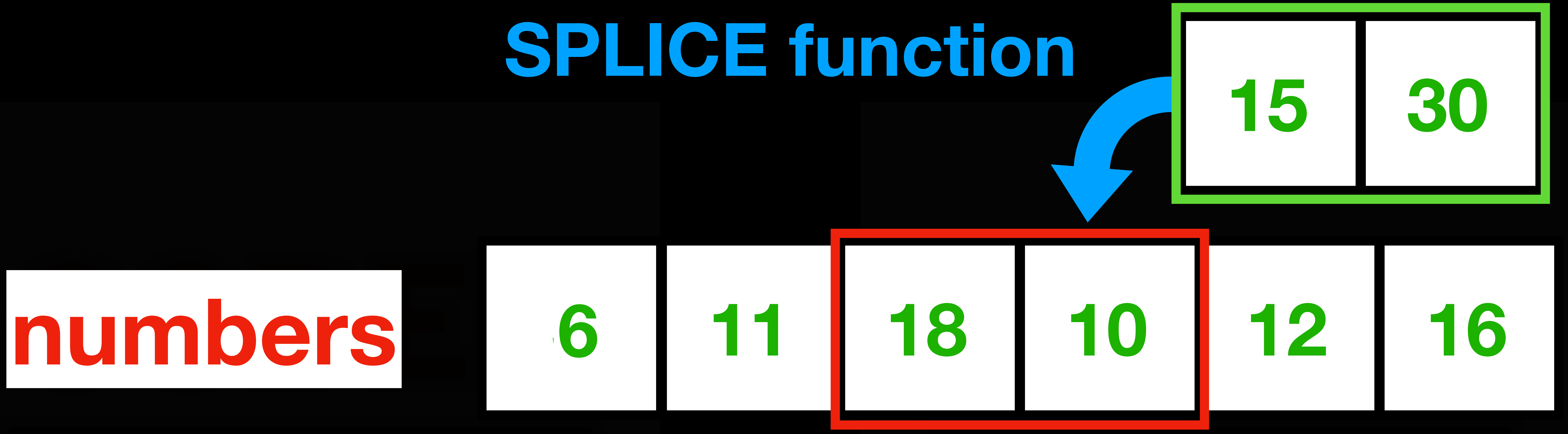
Mutating Method

SPLICE function (return value)



numbers.splice(2, 2) → **[18,10]**

SPLICE function



15	30				

numbers	6	11	18	10	12	16

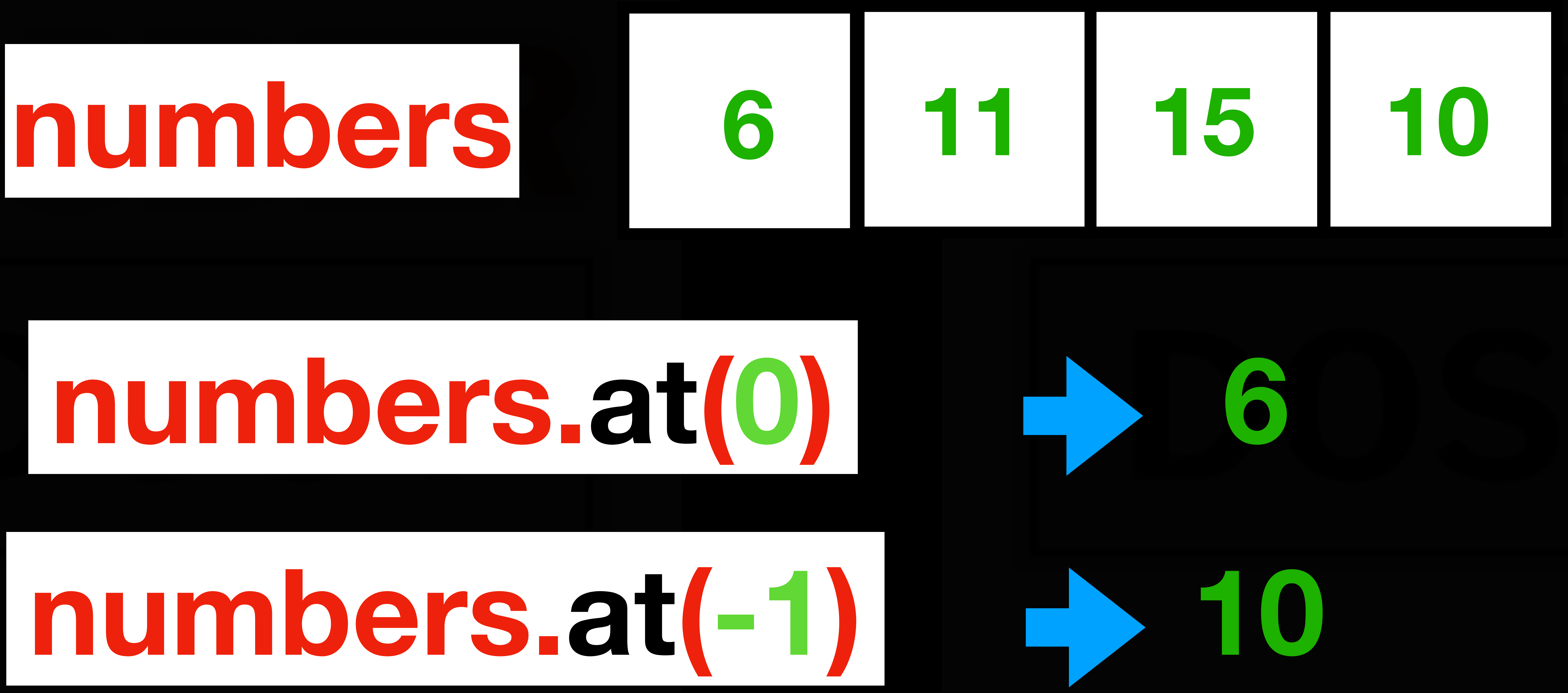
```
numbers.splice( 2, 2,15, 30 )
```

→ [18,10]

numbers → [6,11,15, 30,12,16]

https://www.youtube.com/@coderdost

AT method



numbers

6	11	15	10

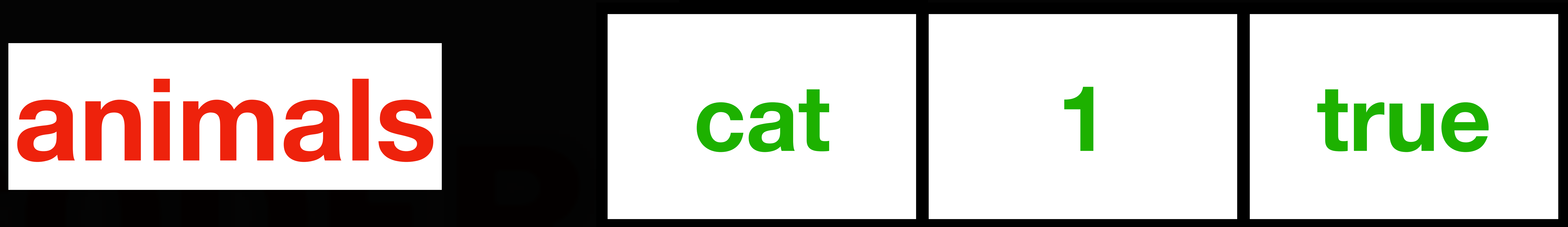
```
numbers.at(0)
```

→ 6

```
numbers.at(-1)
```

→ 10

Mixed Array



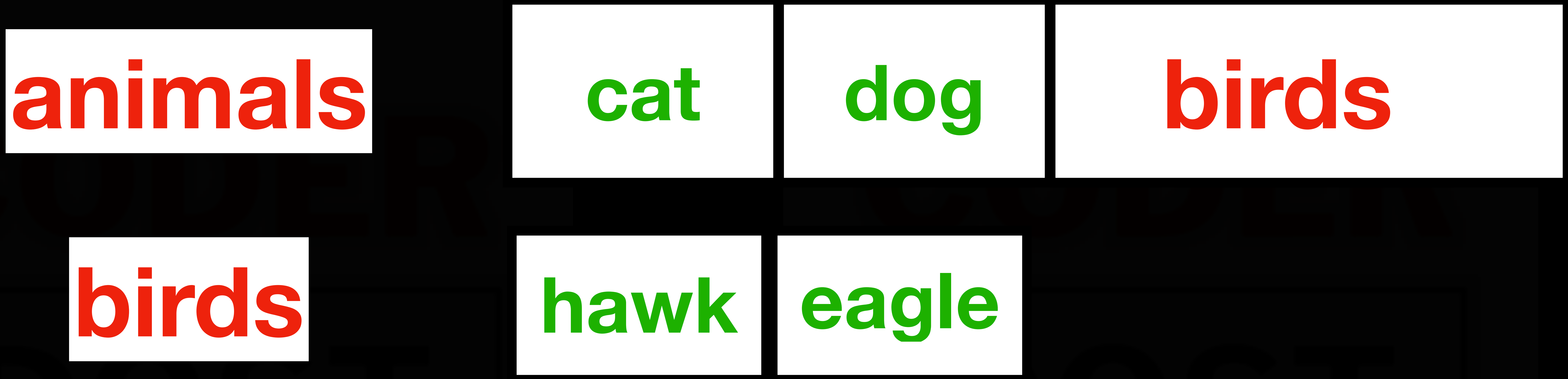
animals

cat	1	true

```
animals = ["cat", 1 , true ]
```

NO ERRORS

Nested Array


animals

cat

dog

birds

birds

hawk

eagle

```
animals = ["cat", "dog" , [ "hawk", "eagle"] ]
```

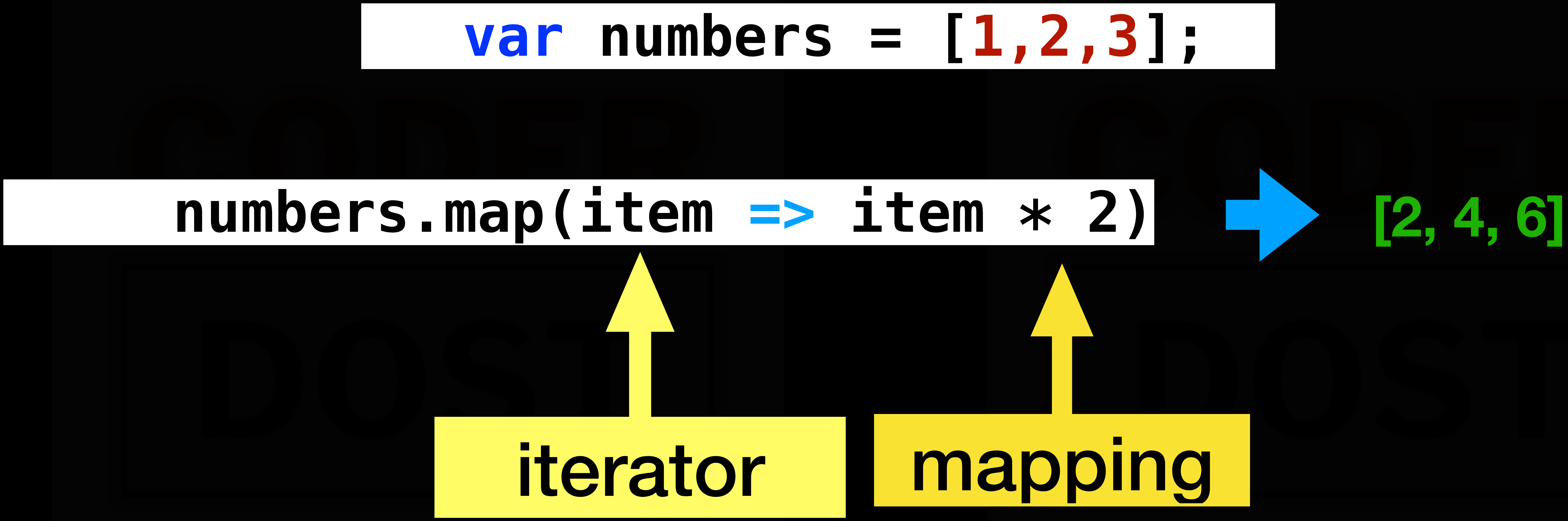
Accessing Nested Array

animals = [“cat”, “dog” , [“hawk”, “eagle”]]

animals[2][1] → “eagle”

animals[2][0] → “hawk”

Higher order functions : map()



```
var numbers = [1,2,3];
```

```
numbers.map(item => item * 2)
```

https://www.youtube.com/@coderdost

Higher order functions : map()

```
var numbers = [1,2,3];
```

```
numbers.map(item => item * 2)
```

ITERATION 1	1	2
ITERATION 2	2	4
ITERATION 3	3	6

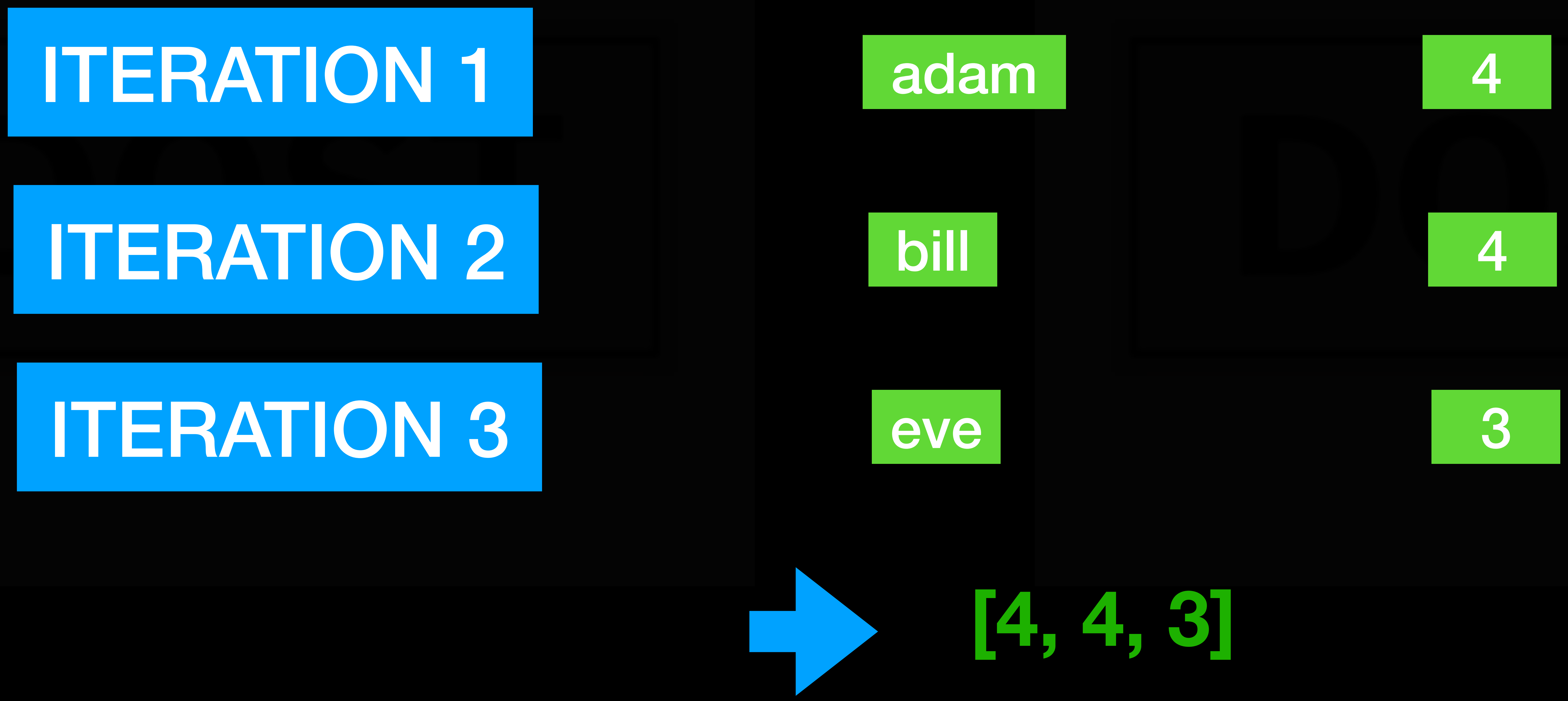
➡ [2, 4, 6]

https://www.youtube.com/@coderdost

Higher order functions : map()

```
var users = [{name:“adam”},{name:“bill”},{name:“eve”}];
```

```
users.map(item => item.name.length)
```



ITERATION 1 — adam — 4

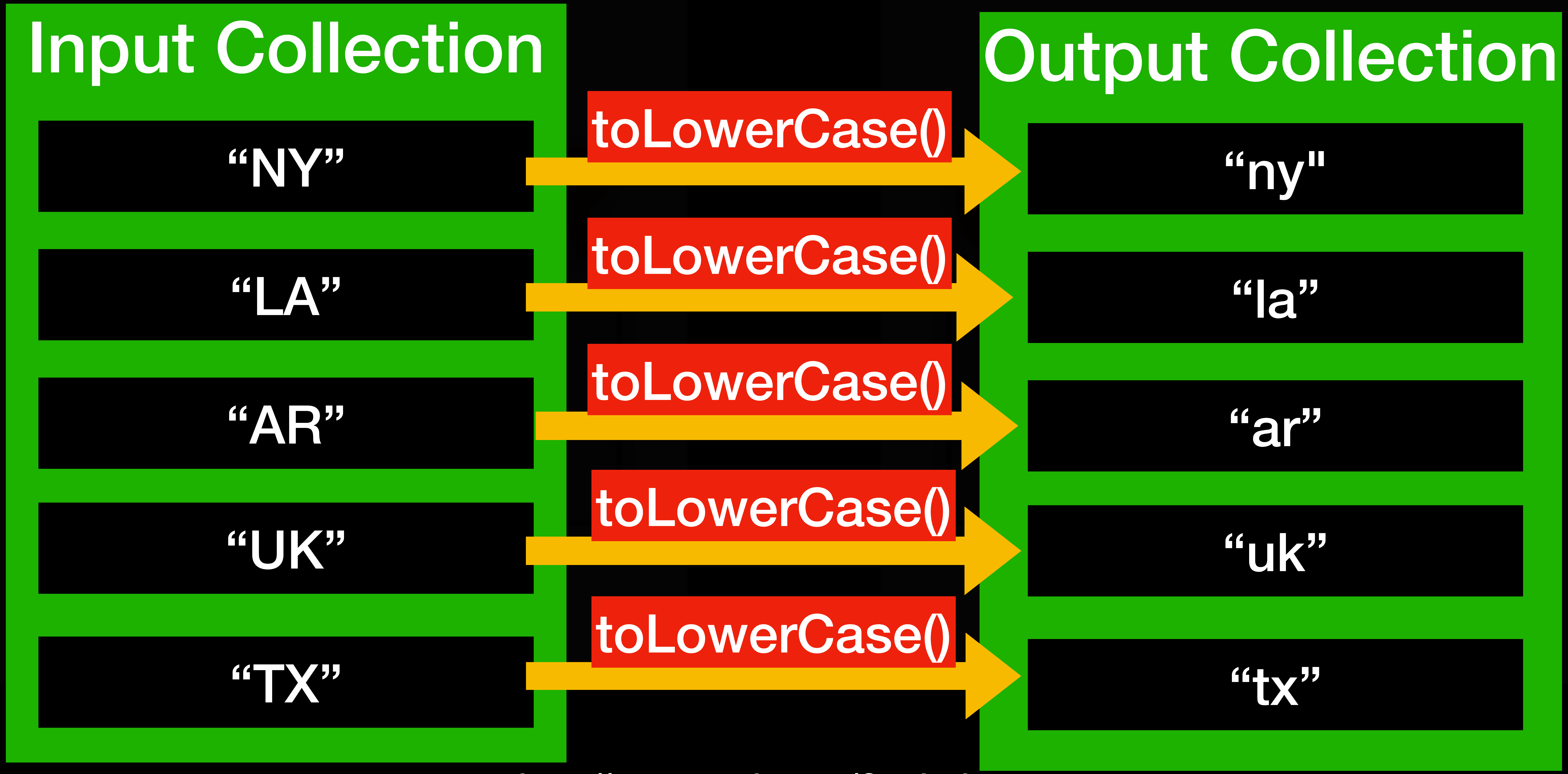
ITERATION 2 — bill — 4

ITERATION 3 — eve — 3

[4, 4, 3]

https://www.youtube.com/@coderdost

Higher order functions : map()



Input Collection

Output Collection

“NY” toLowerCase() “ny"

“LA” toLowerCase() “la”

“AR” toLowerCase() “ar”

“UK” toLowerCase() “uk”

“TX” toLowerCase() “tx”

https://www.youtube.com/@coderdost

map()

```
const cities = [“NY”,”LA”,”TX”];
```

```
cities.map((city) => city.toLowerCase())
```

ITERATION 1	“NY”	“ny”
ITERATION 2	“LA”	“la”
ITERATION 3	“TX”	“tx”

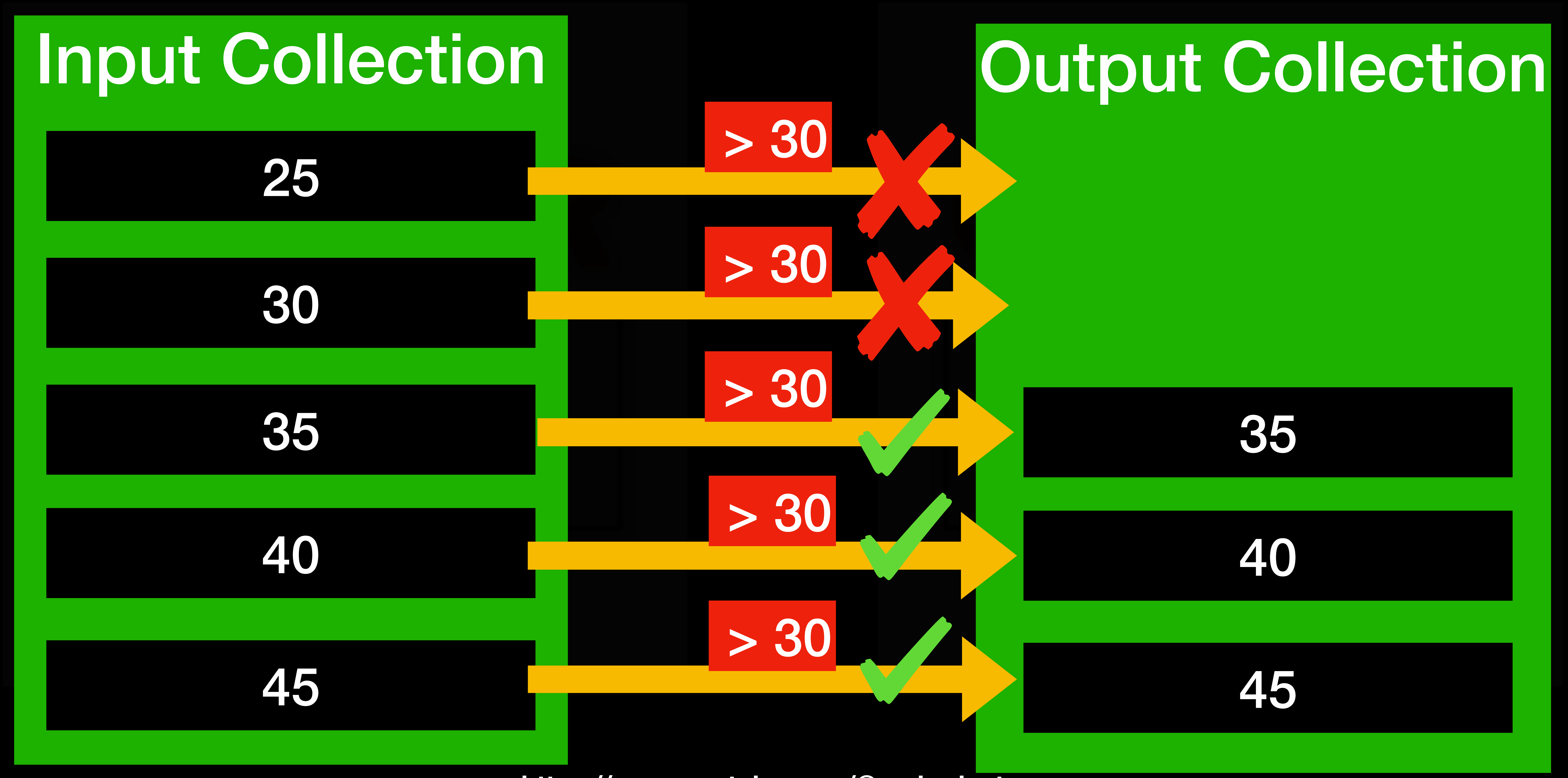
map()

```
const cities = [“NY”,”LA”,”TX”];
```

```
const low = cities.map((city) => city.toLowerCase());
```

low → [“ny”,”la”,”tx”]

Higher order functions : filter()



Input Collection

25

30

35

40

45

> 30

> 30

> 30

> 30

> 30

Output Collection

35

40

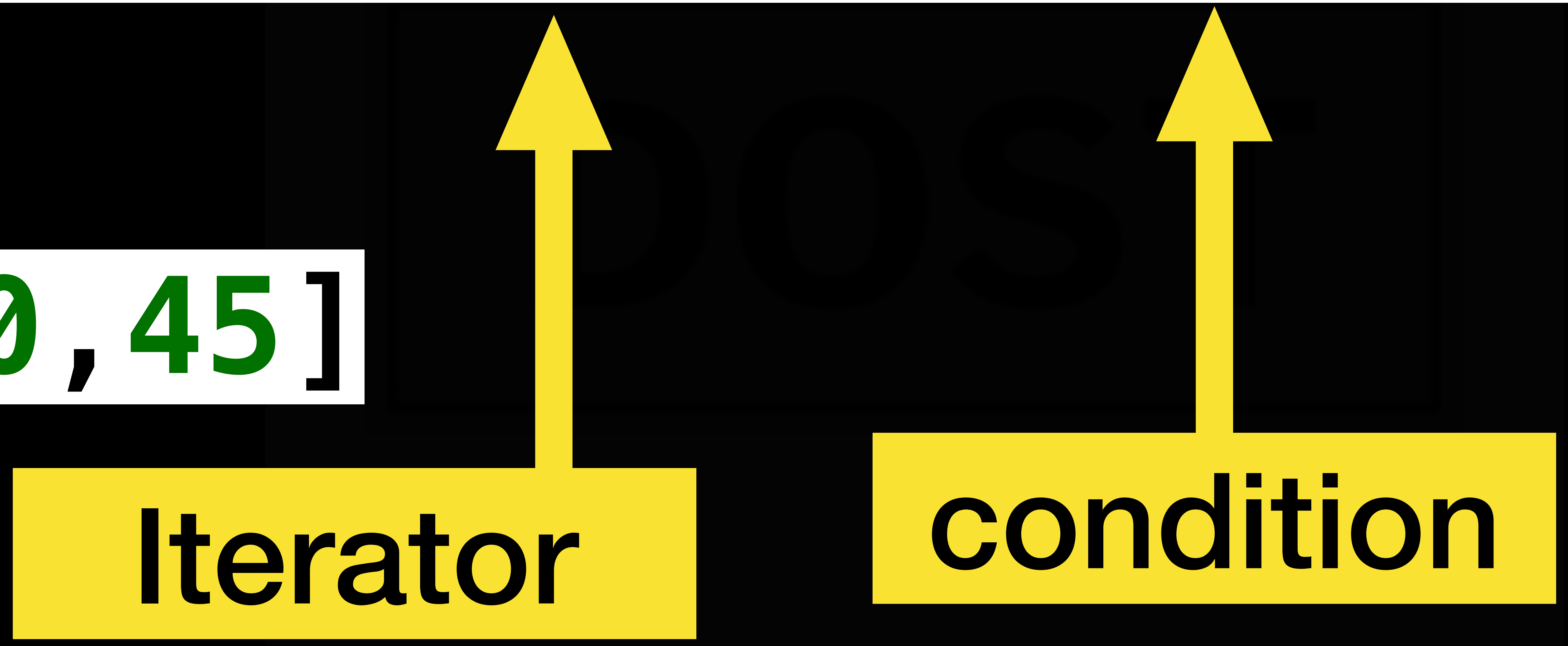
45

https://www.youtube.com/@coderdost

filter()

```
const ages = [25,30,35,40,45];

const ageGreat = ages.filter((age) => (age > 30));
```

ageGreat → [35,40,45]

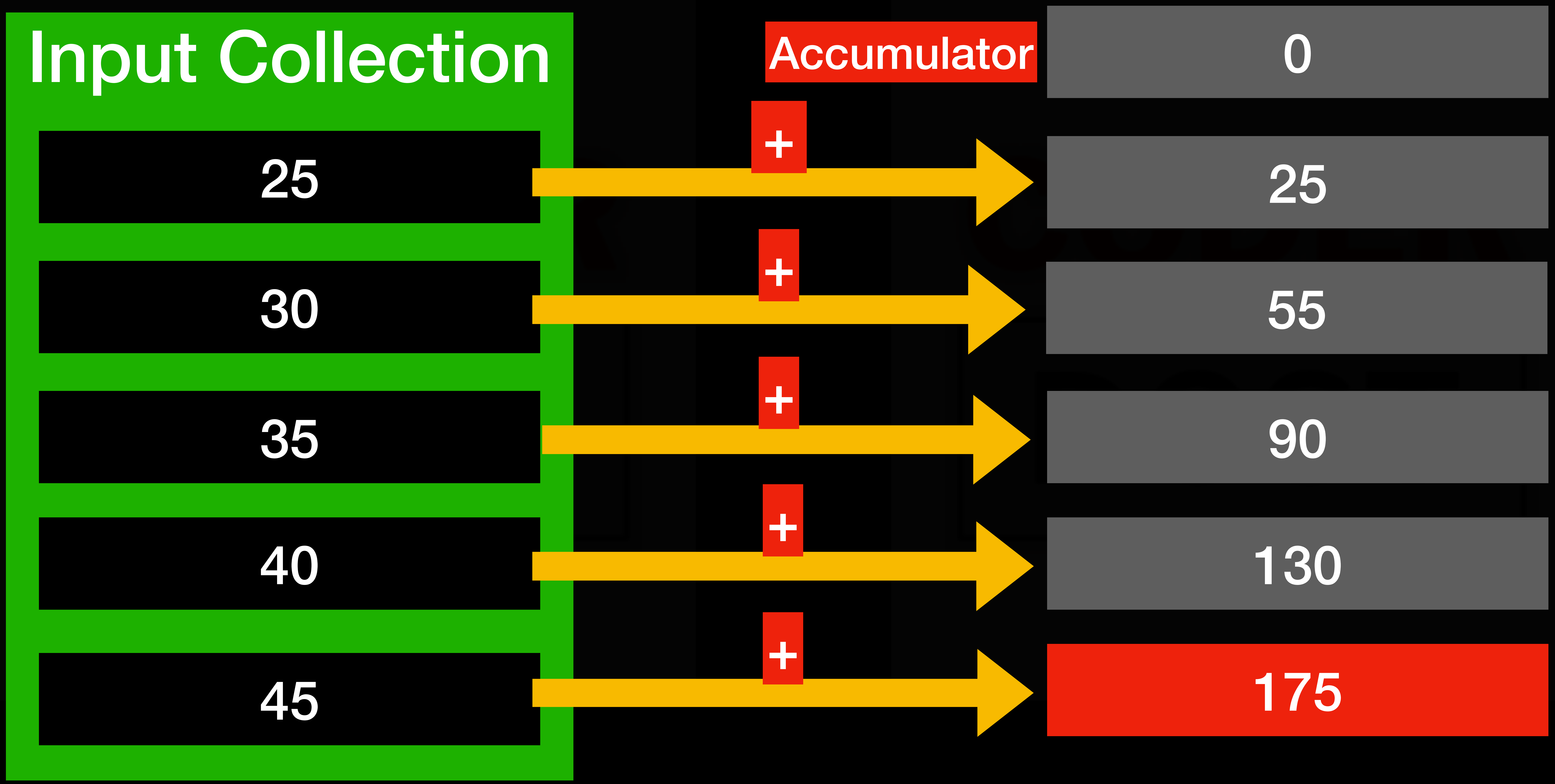


Iterator

condition

https://www.youtube.com/@coderdost

Higher order functions : reduce()



Input Collection

25

30

35

40

45

Accumulator

0

+ 25

+ 55

+ 90

+ 130

+ 175

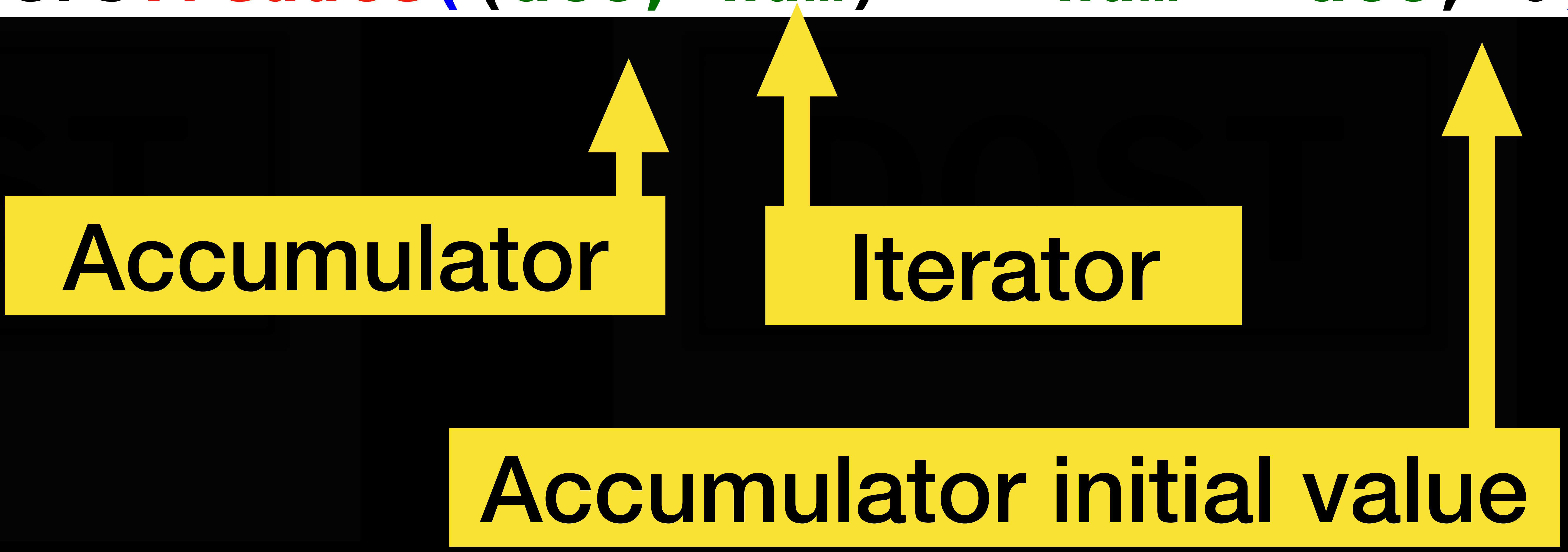
https://www.youtube.com/@coderdost

Reduce

```
const numbers = [25,30,35,40,45];
```

```
const r = numbers.reduce((acc, num) => num + acc, 0)
```

r → 175



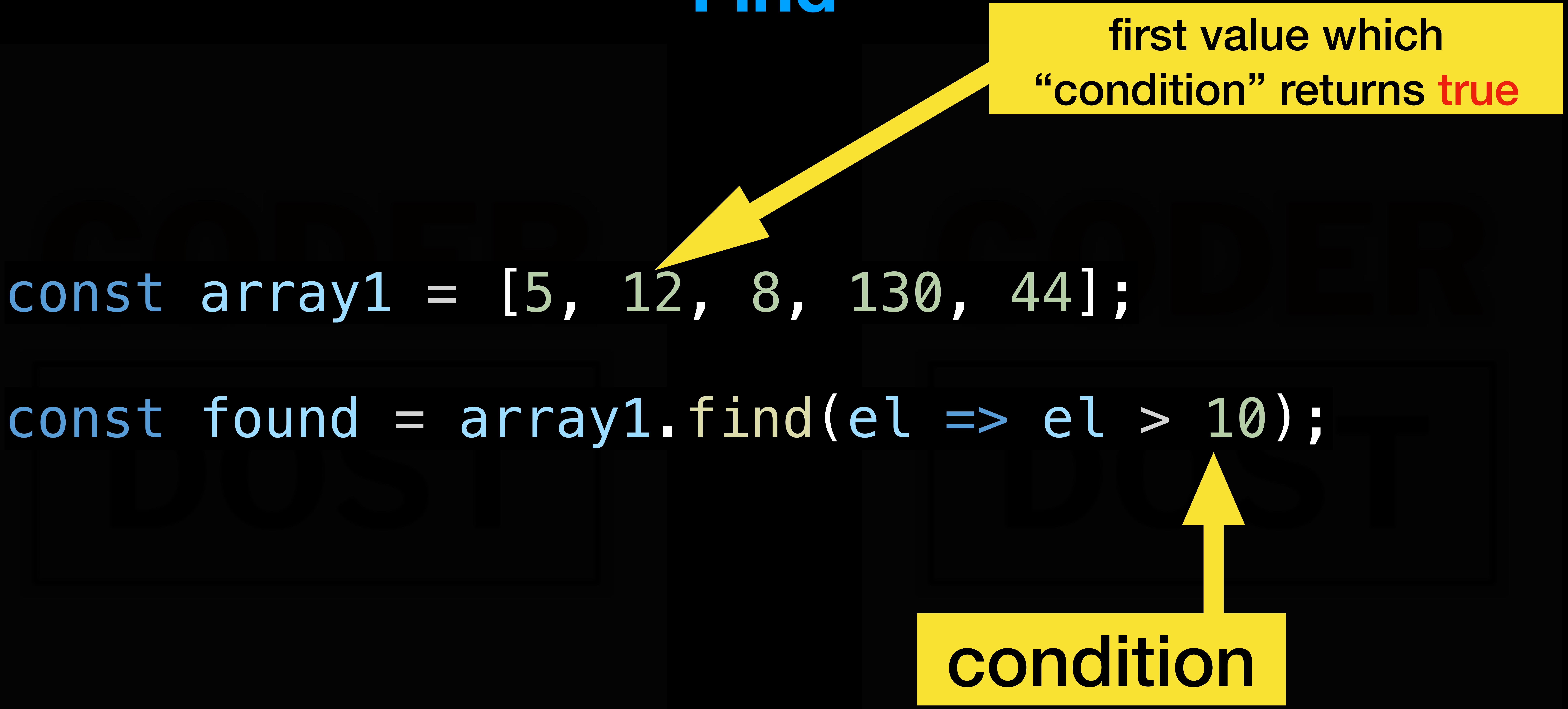
Accumulator

Iterator

Accumulator initial value

https://www.youtube.com/@coderdost

Find

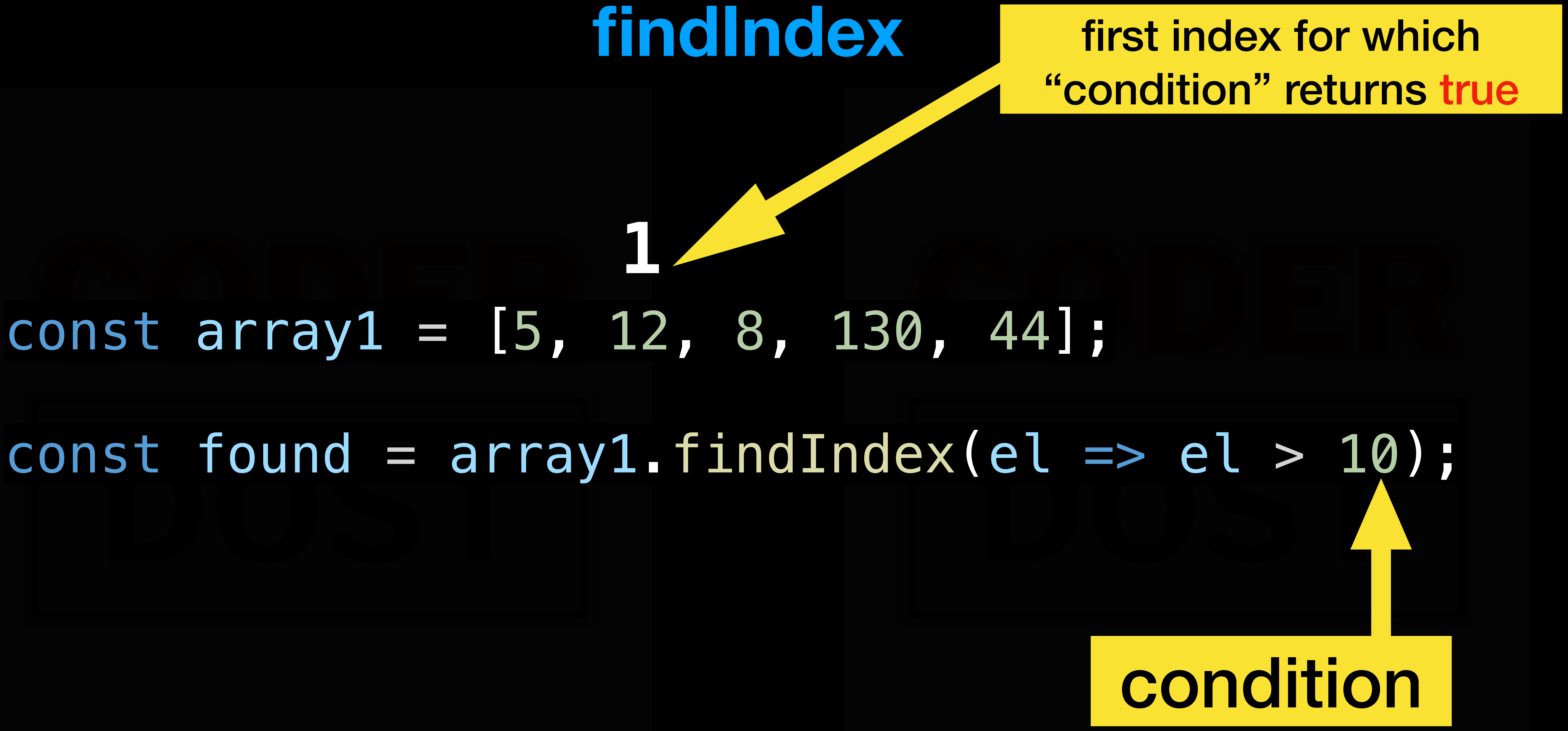


```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(el => el > 10);
```

first value which “condition” returns true

condition

findIndex



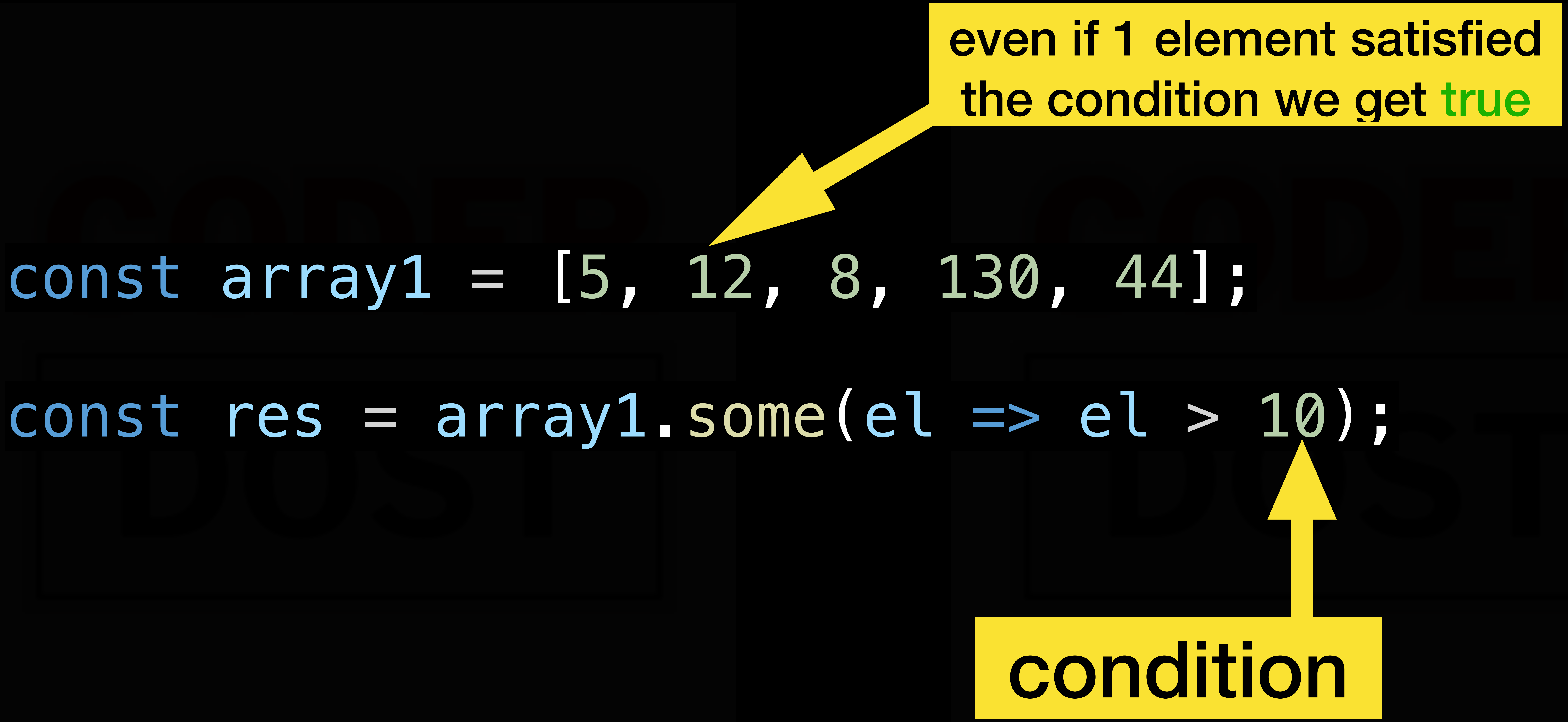
first index for which
“condition” returns true

1

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.findIndex(el => el > 10);
```

condition

some



```
const array1 = [5, 12, 8, 130, 44];

const res = array1.some(el => el > 10);
```

even if 1 element satisfied the condition we get true

condition

every

even if 1 element don’t satisfied the condition we get false

```
const array1 = [5, 12, 8, 130, 44];

const res = array1.every(el => el < 100);
```

condition

flat

```
const arr1 = [0, 1, 2, [3, 4]];

console.log(arr1.flat());  →  [0, 1, 2, 3, 4]


const arr2 = [0, 1, 2, [[[3, 4]]]];

console.log(arr2.flat(2));  →  [0, 1, 2, [3, 4]]
```

depth of flattening

https://www.youtube.com/@coderdost

flatMap

flat() + map()

```
const arr1 = [1, 2, [3], [4, 5], 6, []];

const flattened = arr1.flatMap(num => num);
```

[1, 2, 3, 4, 5, 6]

Sorting Array

```
const arr = ['March', 'Jan', 'Feb', 'Dec'];
```

```
arr.sort( compareFn )
```

```
function compareFn(a, b) {
    if (a < b) {
      return -1;
    }
    if (a > b) {
      return 1;
    }
    // a must be equal to b
    return 0;
  }
```

Function chaining

var word = “Hello”

word.split(“”) → [“H”,”e”,”l”,”l”,”o”]

[“H”,”e”,”l”,”l”,”o”].reverse() → [“o”,”l”,”l”,”e”,”H”]

[“o”,”l”,”l”,”e”,”h”].join(“”) → “olleH”

Function chaining

```
word.gx( ).fx( ).hx( )
```

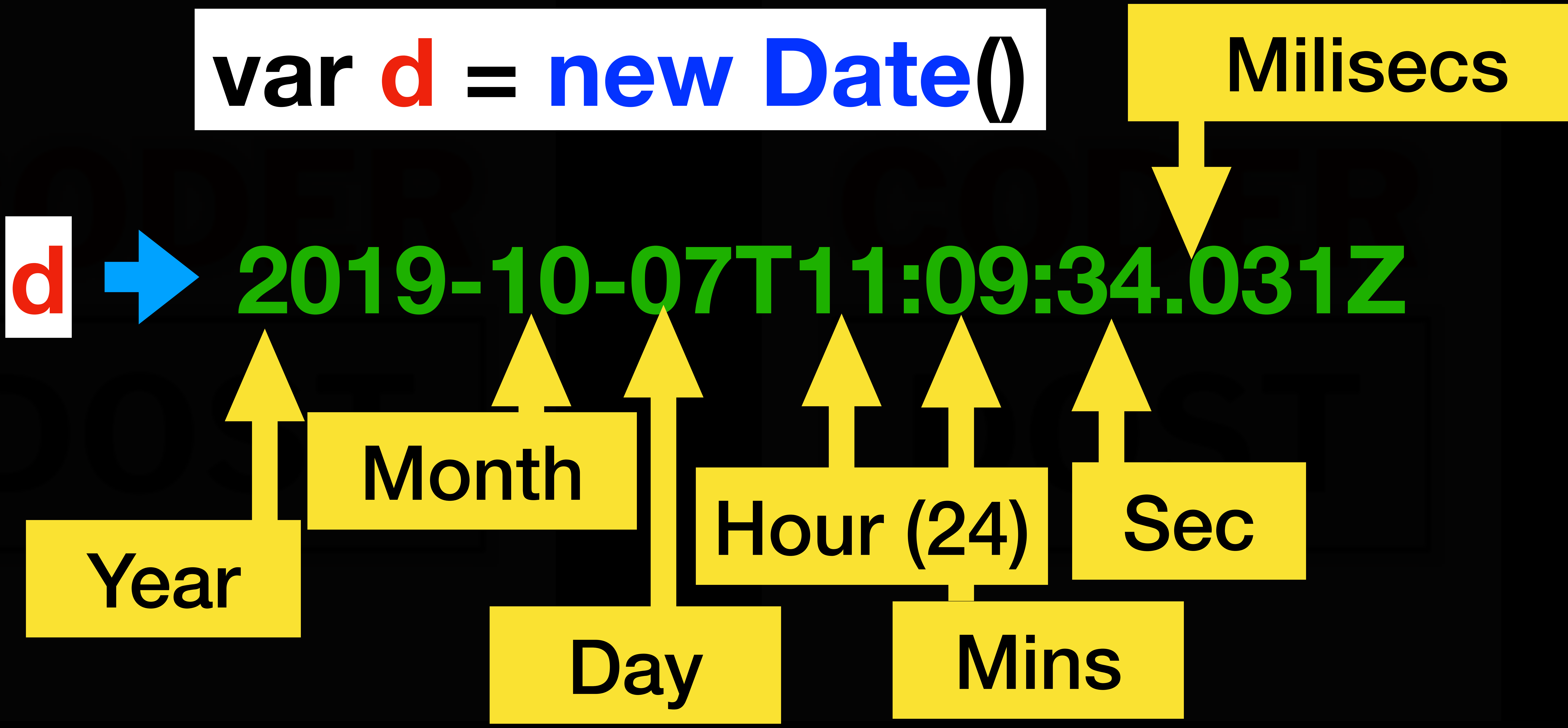
gx( ) → return value compatible type with fx()

fx( ) → return value compatible type with hx()

8. Date Library

Date Library

```
var d = new Date()
```



d → 2019-10-07T11:09:34.031Z

Milisecs

Year

Month

Day

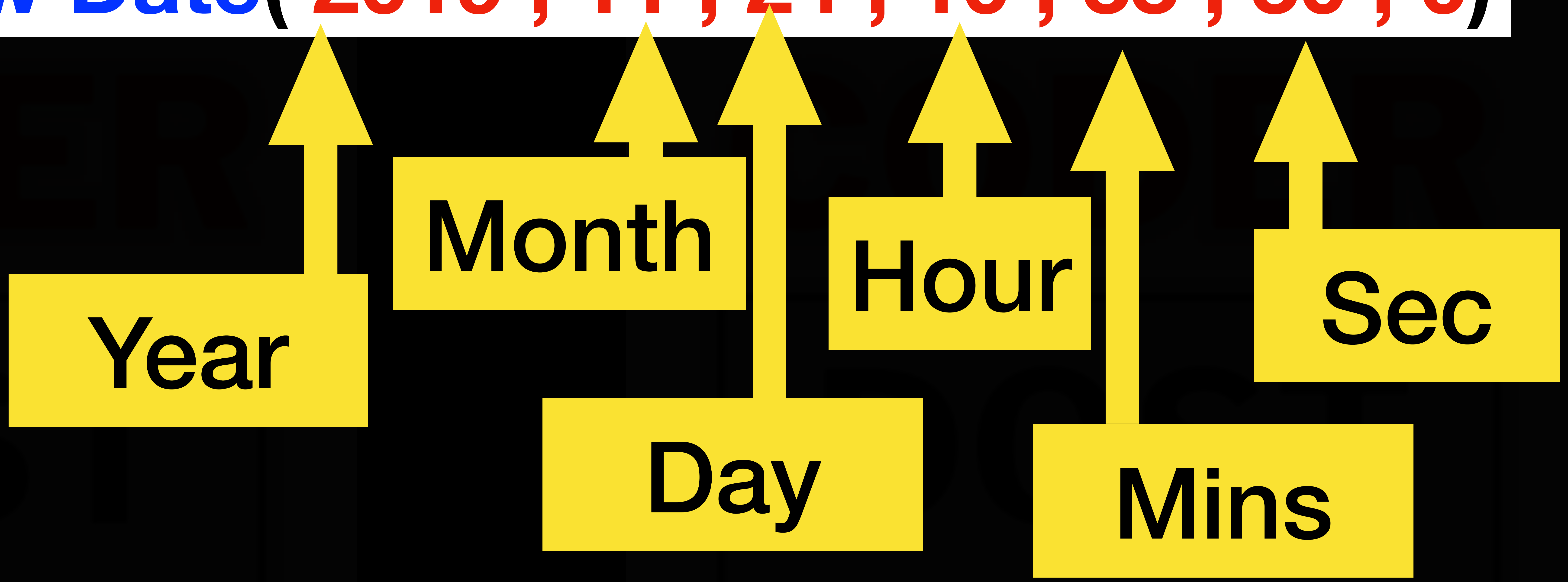
Hour (24)

Mins

Sec

Date Library

```
var d = new Date( 2019 , 11 , 24 , 10 , 33 , 30 , 0)
```



Date Library

JAN =0
FEB =1
MAR = 2
...

DEC = 11

d.getDay() → 2

d.getDate() → 24

d.getMonth() → 11

d.getFullYear() → 2019

Date Library

```
var d = new Date(2019, 11, 24, 10, 33, 30, 0)
```

`d.getTime()` → 1577163810000

Millisecs from 1st Jan 1970

`d.toUTCString()` → "Tue, 24 Dec 2019 05:03:30 GMT”

`d.toISOString()` → "2019-12-24T05:03:30.000Z"

9. LocalStorage

LocalStorage

its a API of Browser (window Object) to Store data Locally

Every Origin separately stores data

e.g.

google.com will have different database, and facebook.com have different storage on your browser

LocalStorage

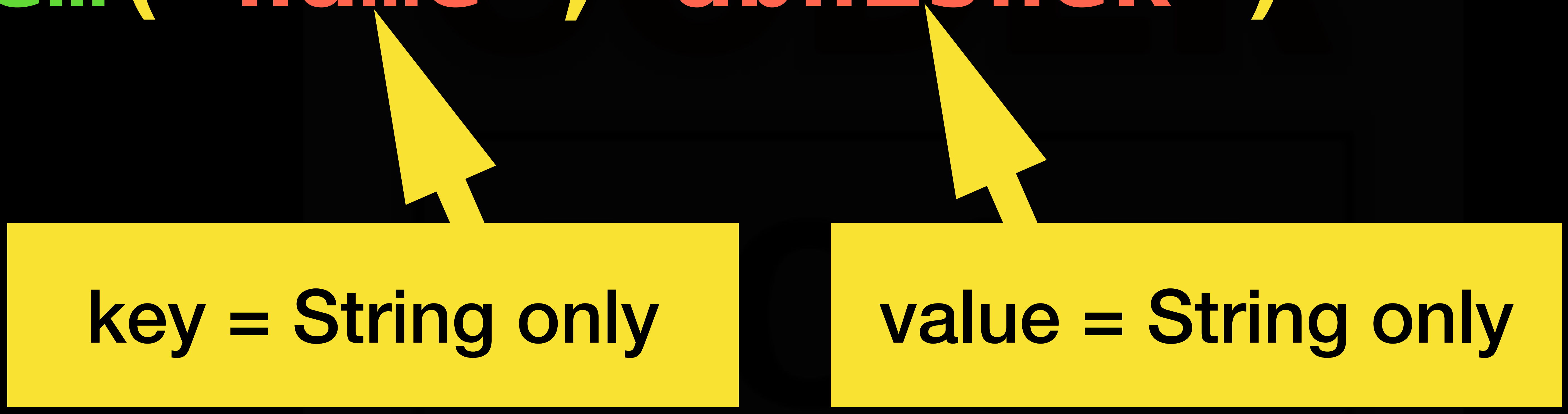
```
window.localStorage
```

OR

```
localStorage
```

LocalStorage: Adding Data

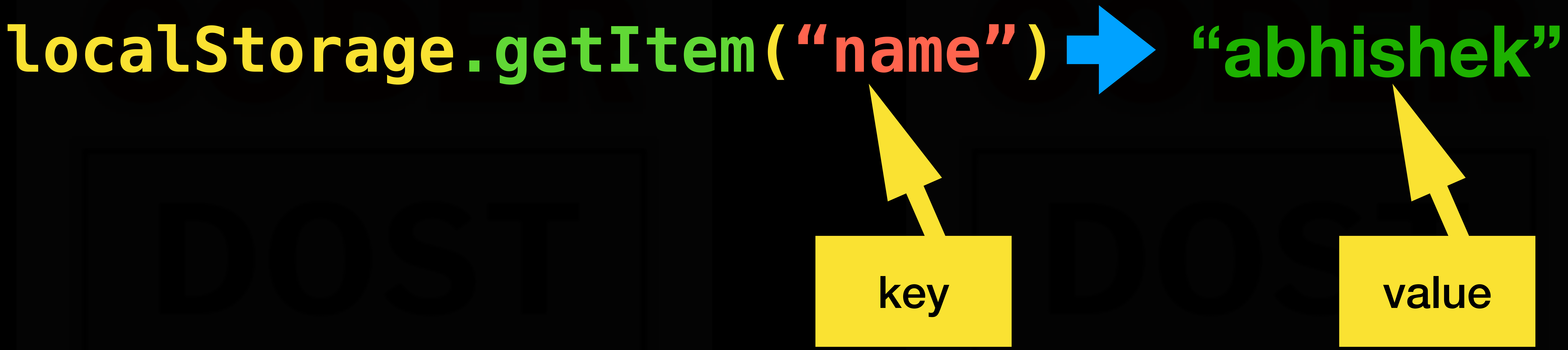
```
localStorage.setItem(“name”,”abhishek”)
```



key = String only

value = String only

LocalStorage: Adding Data



```
localStorage.getItem(“name”)
```

→ “abhishek”

key

value

if “key” is not found , returns “null”

LocalStorage: removing Data

```
localStorage.removeItem(“name”)
```

key

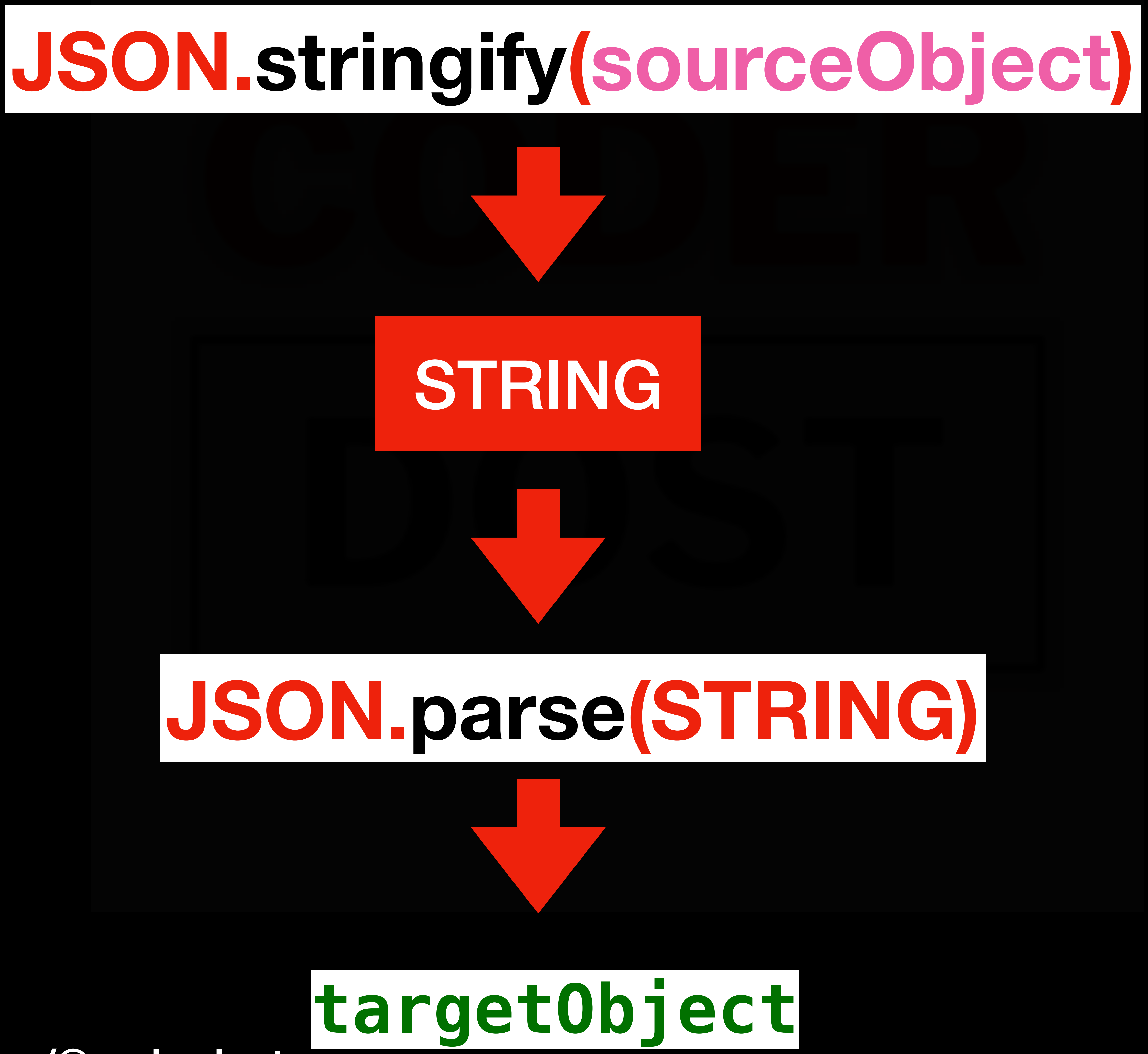
LocalStorage: Clear All Data

```
localStorage.clear()
```

removes all keys for that origin

JSON to String/ String to JSON

```
 var sourceObject = {
          name : “abhishek”,
          age: 30,
          course : {
                  name : “nodejs”
                  }
}
```



JSON.stringify(sourceObject)

STRING

JSON.parse(STRING)

targetObject

https://www.youtube.com/@coderdost

https://www.youtube.com/@coderdost

10. Object Oriented JS

Constructor

this is shortcut

```
const person = {
                name : “p1”,
           }
```

this is full form

```
const person = new Object({
                name : “p1”,
           })
```

constructor function

Constructor

```
function Person(name){
          this.name = name;
        )
```

```
const person = new Person(‘p1’)
```

Every function in JavaScript is also a Constructor

constructor function

prototype - property

```
 function Person(name){
                this.name = name;
          )
```

```
Person.prototype.printName = function(){
   console.log(this.name)
}
```

```
const person = new Person(‘p1’)
```

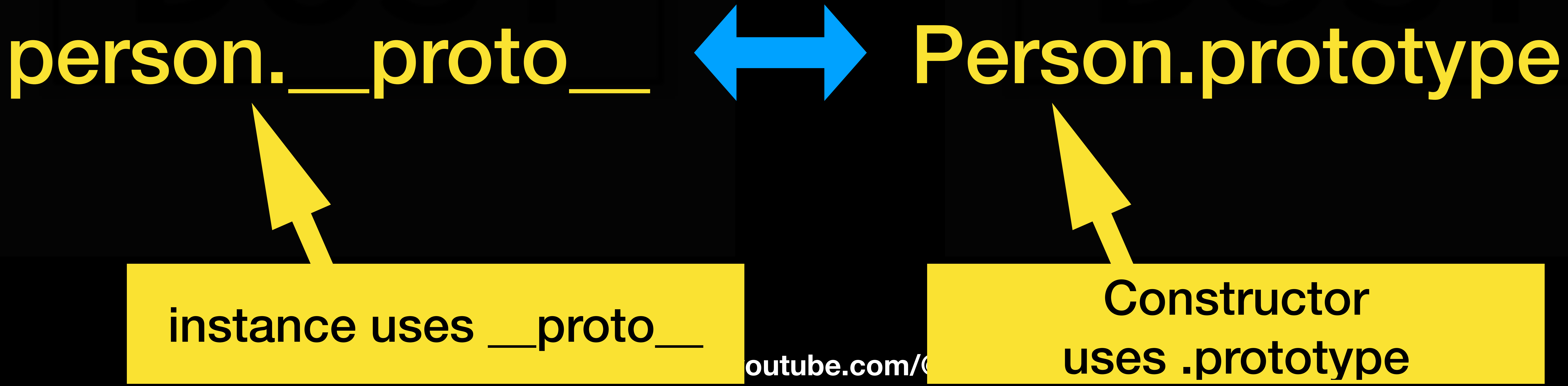
```
person.printName()
```

https://www.youtube.com/@coderdost

prototype - property

```
function Person(name){
                this.name = name;
        )
```

```
const person = new Person('p1')
```



person.__proto__

Person.prototype

instance uses __proto__

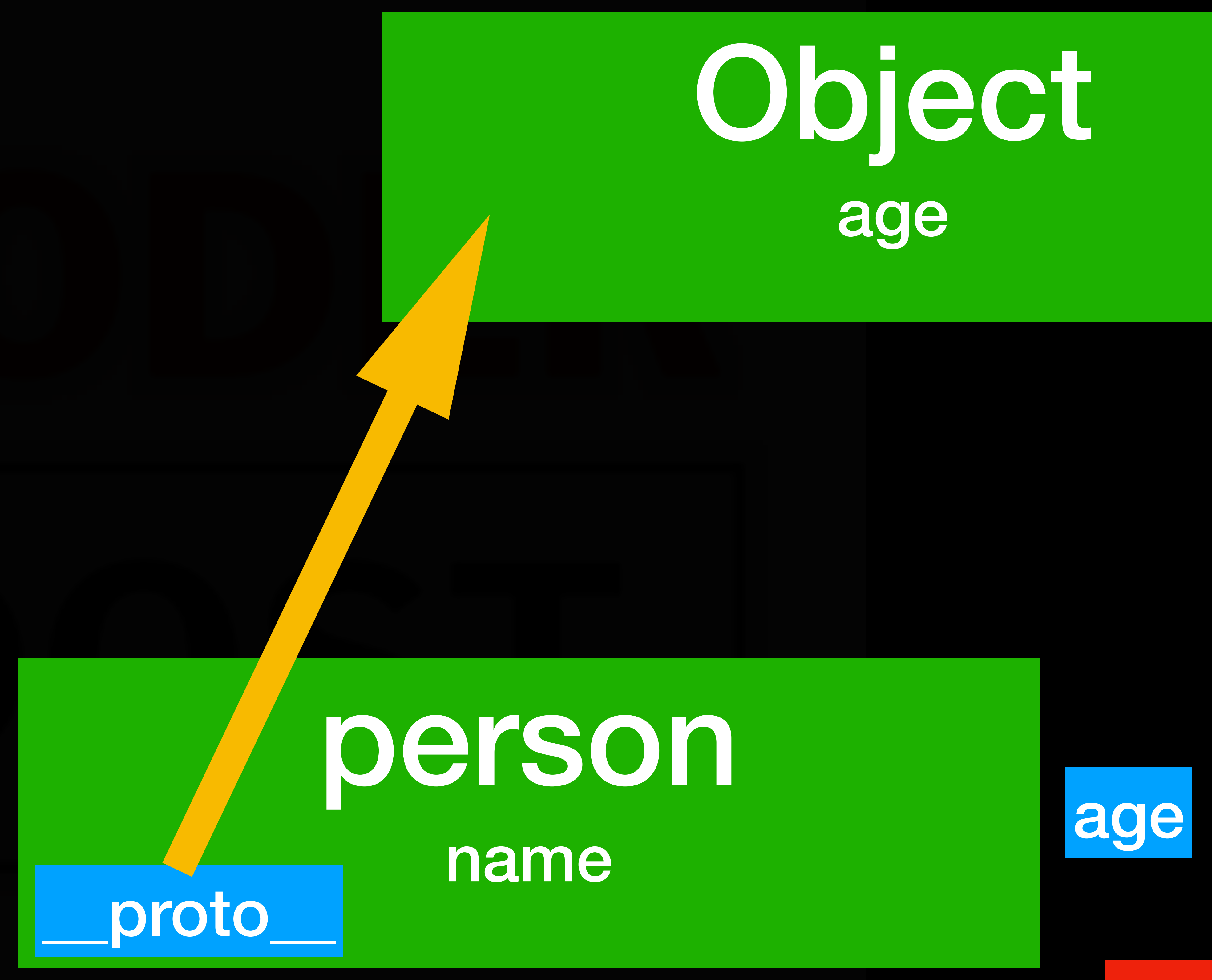
Constructor
uses .prototype

prototype

```
Array.prototype.push = function(){

}

Array.prototype.pop = function(){

}
```

You can also over-write existing methods

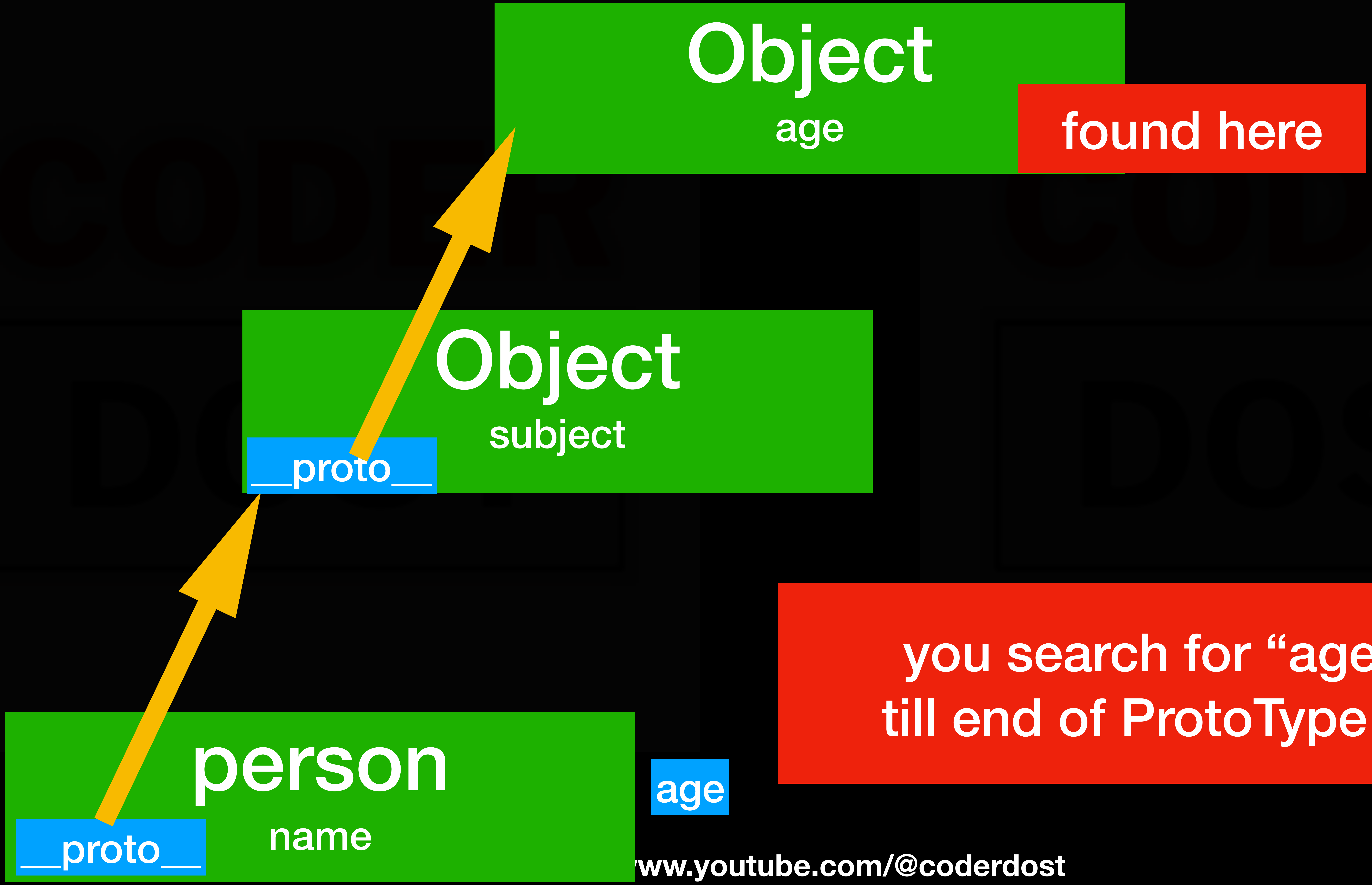
Prototype Inheritance



searching for ‘age’ in “person”

__proto__ property tells about the ProtoType of this instance. You can start searching from there

Prototype Inheritance



Object

age

found here

Object

subject

__proto__

person

name

__proto__

age

you search for “age” property
till end of ProtoType chain (null)

Built-in Prototypes

Object.prototype

Array.prototype

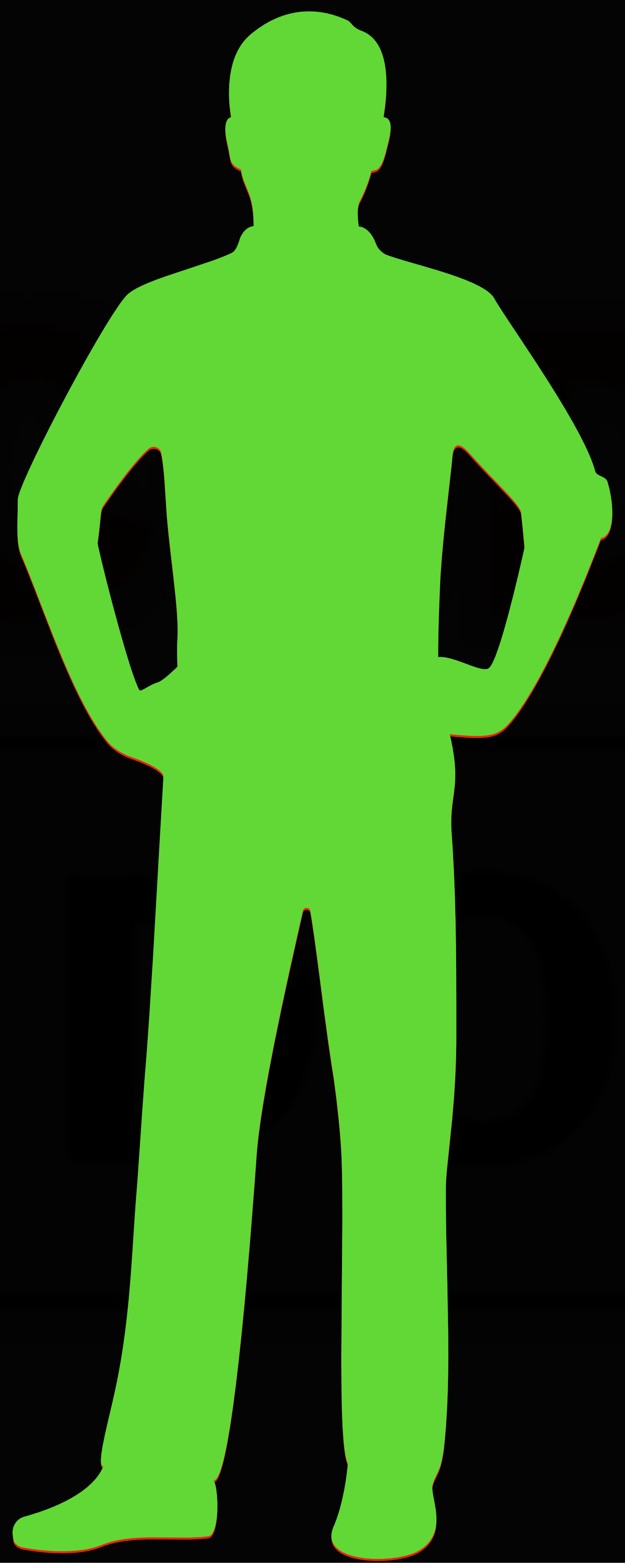
Function.prototype

ES6 - Class

Name	abhishek
Age	30
Address	Street 10, mumbai, india
Mobile	888888888

we want to store all this info of person

Objects



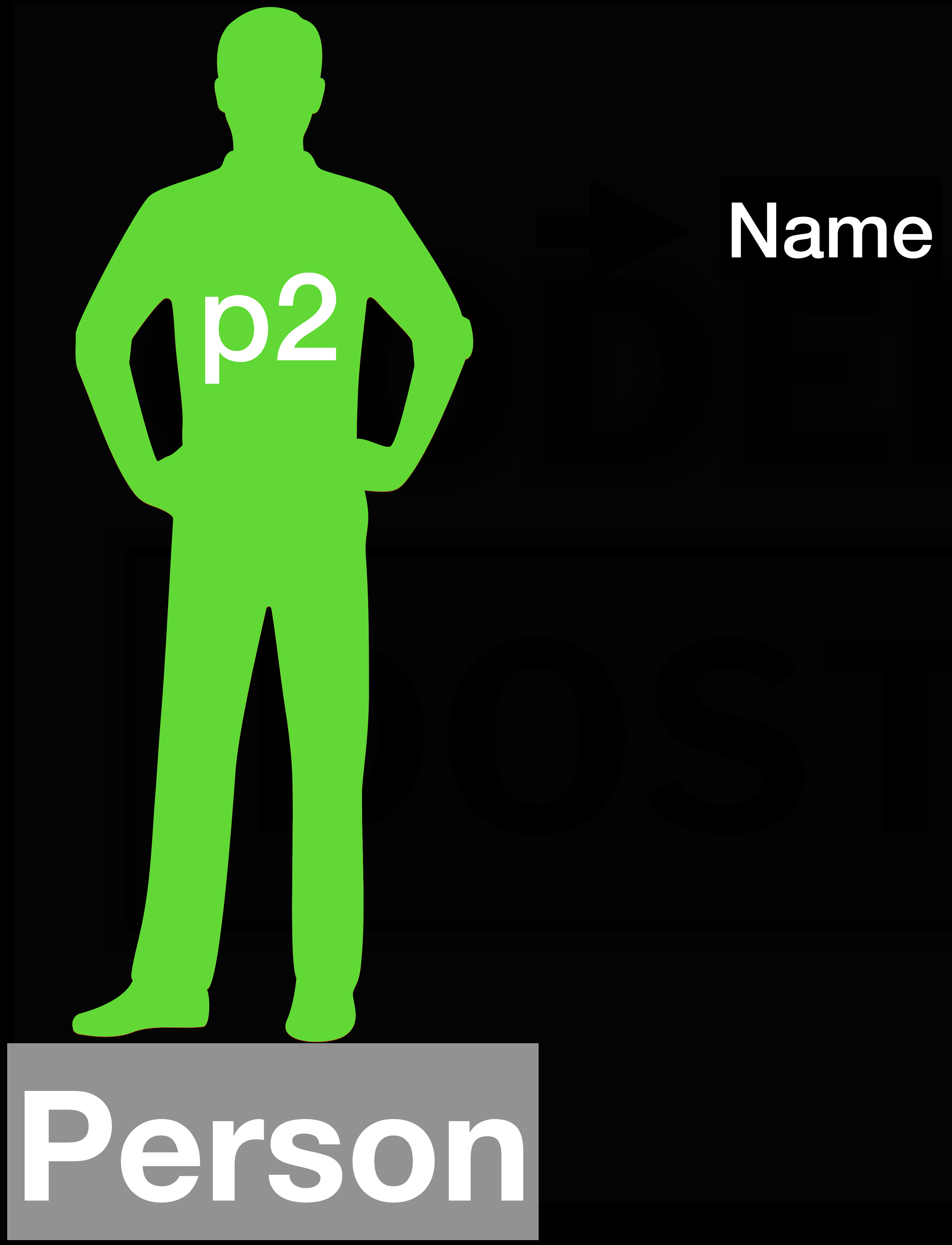
person

```
var person = {
         name  : "abhishek",
         age :30 ,
         address : "street 10, Mumbai, India",
         phone:8888888888
      }
```

```
var person1 = {
         fullname : "ajay",
         age :30 ,
         address : "street 10, Mumbai, India",
         mobile:8888888888
      }
```

But issue can be there if do it manually - mismatching keys

Class

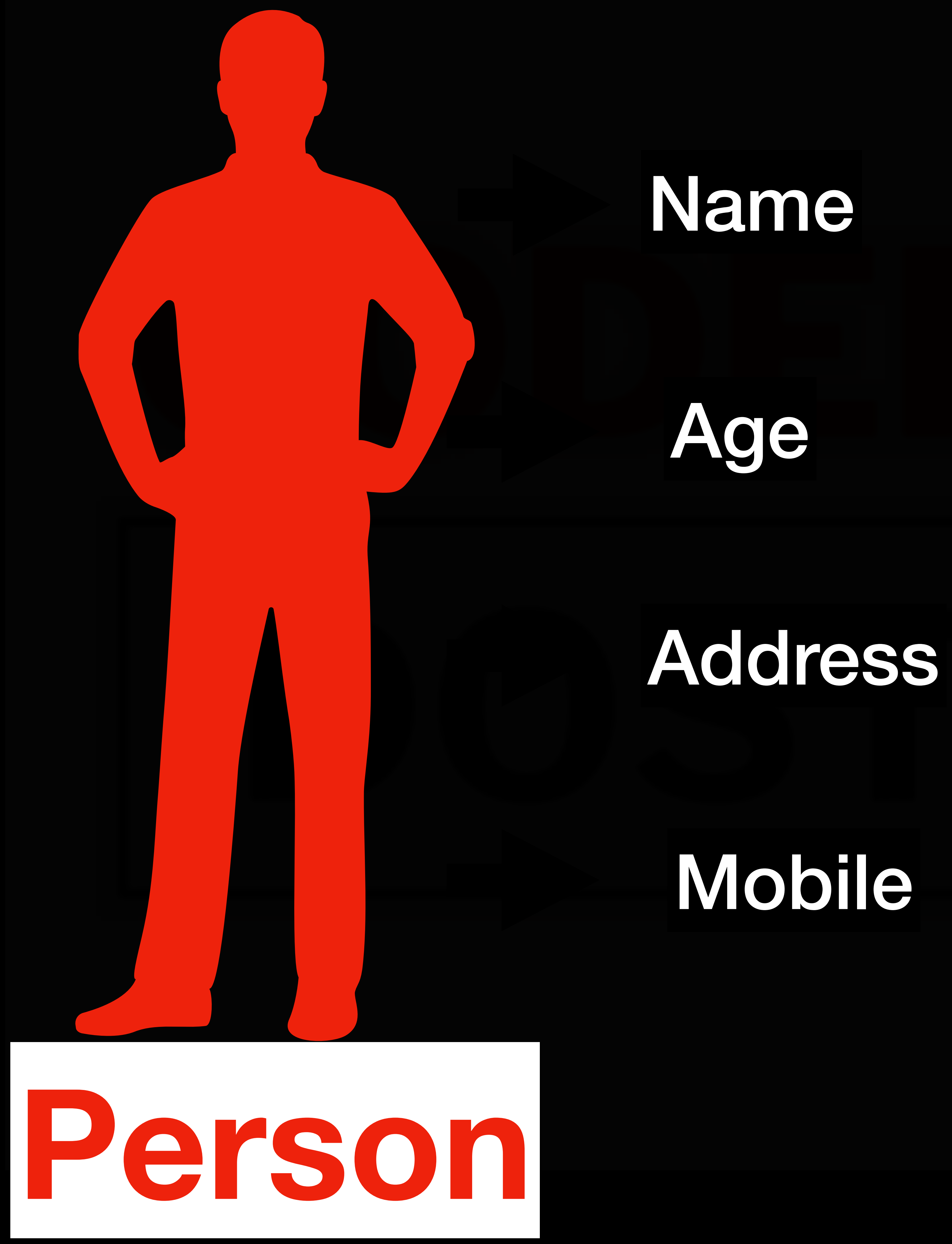


```
class Person {
    constructor(name) {
        this.name = name;
    }
}
let p1 = new Person("jay");
let p2 = new Person("jack");
```

Constructor Call

https://www.youtube.com/

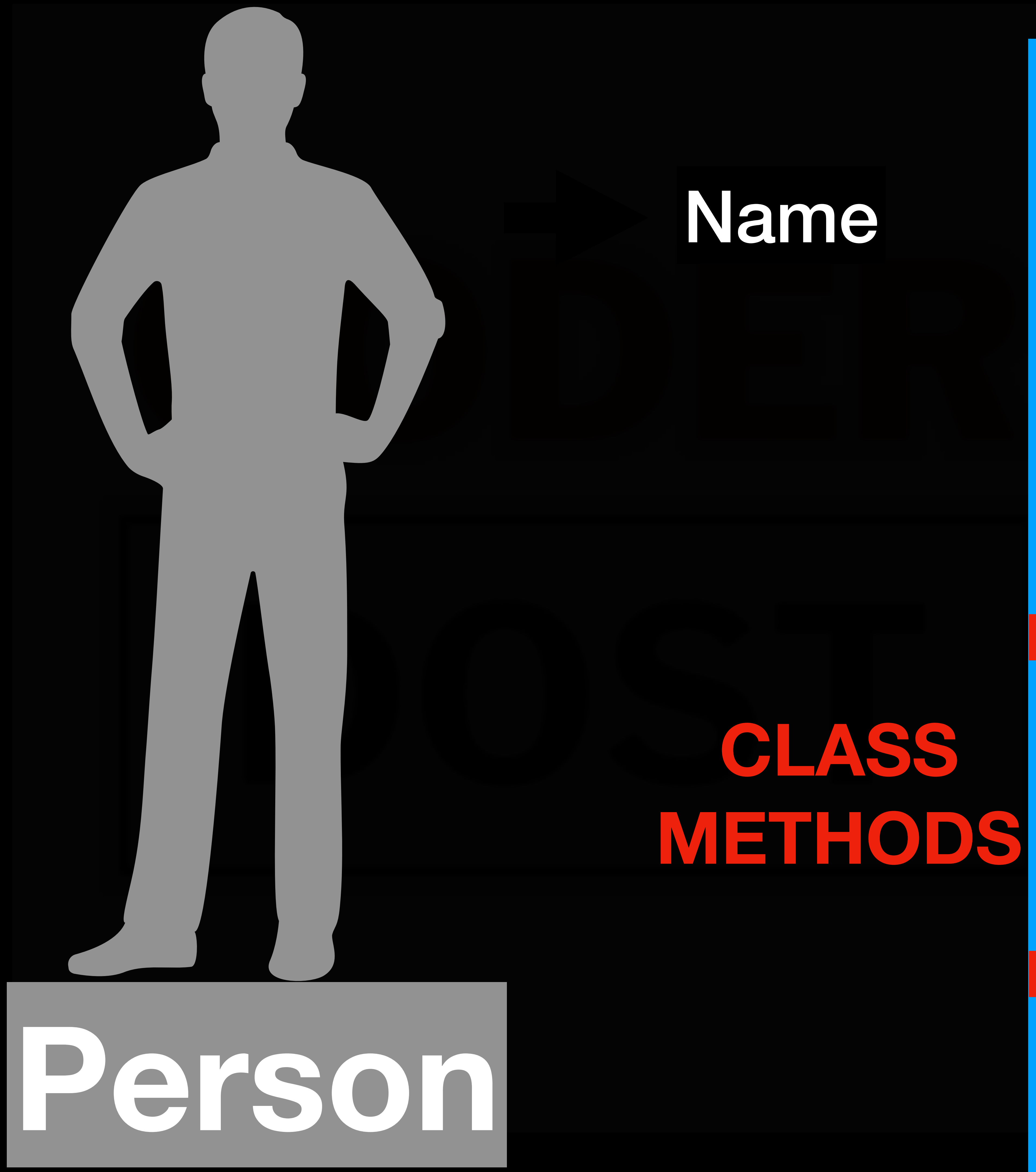
Class Properties



```
let p1 = new Person(
“ajay”,30,”street 10”,88888
);
```

https://www.youtube.com/@coderdost

Class Methods



```
class Person {
        constructor(name) {
          this.name = name;
                    }
        getName() {
          return this.name;
              }
        setName(name) {
          this.name = name;
              }
}
```

Accessor Properties

```
 const person = {
           firstName : “john”,
           lastName : “smith”,
           get fullName(){
             return this.firstName +” ”+ this.lastName
           }
}
```

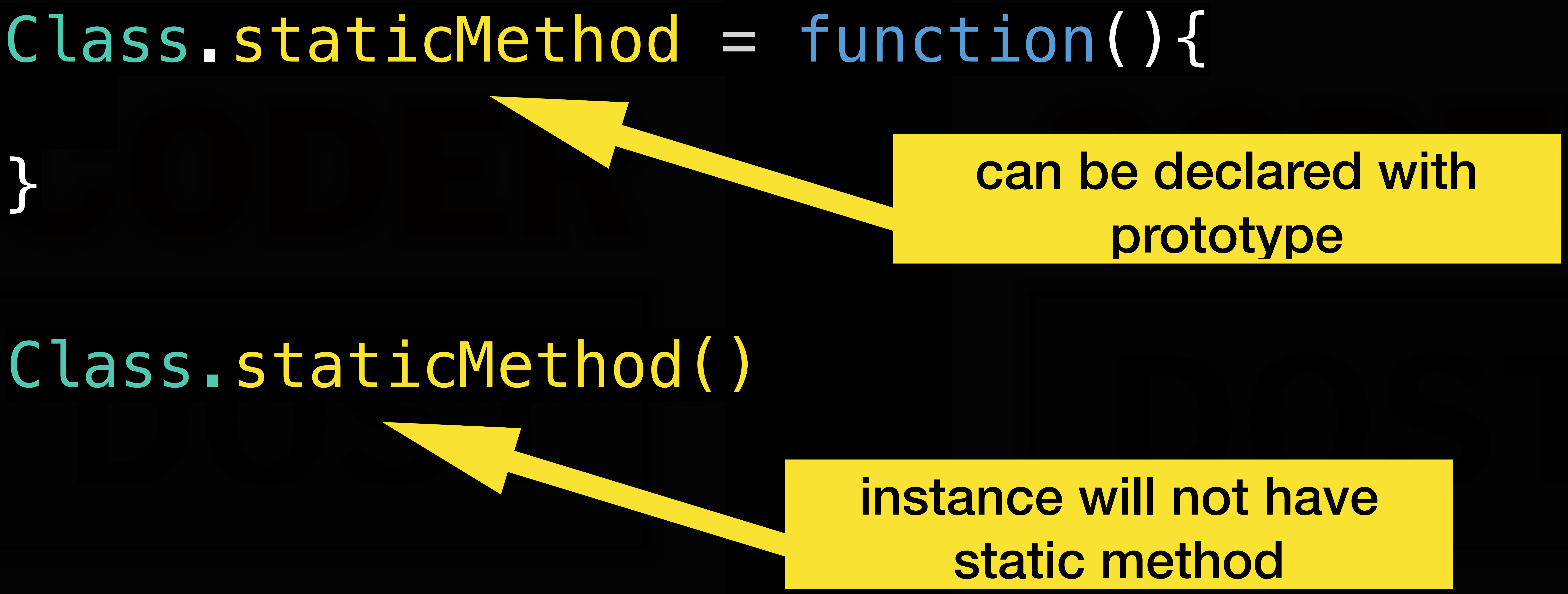
`person.fullName` → `“john smith”`

Accessor Properties

```
const person = {
          firstName : “john”,
          lastName : “smith”,
          get fullName(){
            return this.firstName +” ”+ this.lastName
          }
          set fullName(name){
            this.firstName = name.split(‘’)[0]
            this.lastName = name.split(‘’)[1]
          }
}
```

```
person.fullName  =“Jon Snow”
```

Static Methods



```
Class.staticMethod = function(){

}
```

can be declared with prototype

```
Class.staticMethod()
```

instance will not have static method

Static Methods

```
class Class {
    static staticMethod() {
      // …
    }
  }

Class.staticMethod()
```

can be declared in Class also with “static”

Inheritance

```
class ParentClass {
   constructor(name) {
     this.name = name
   }
 }
class ChildClass extends ParentClass {

 }
const child = new ChildClass(“john”)
```

instance will be having “name” = “john”

Inheritance

```
class ParentClass {
   constructor(name) {
     this.name = name
   }
 }

class ChildClass extends ParentClass {
   constructor(name, age) {
     super(name)
     this.age = age;
   }
 }
```

call parent’s constructor and passes values

```
const child = new ChildClass(“john”, 30)
```

https://www.youtube.com/@coderdost

11. Asynchronous JavaScript

Async JavaScript

```
function double(x){ return x*2};


setTimeout(()=>{
      double(4);
},4000)
```

how can I use this value

Callbacks ?

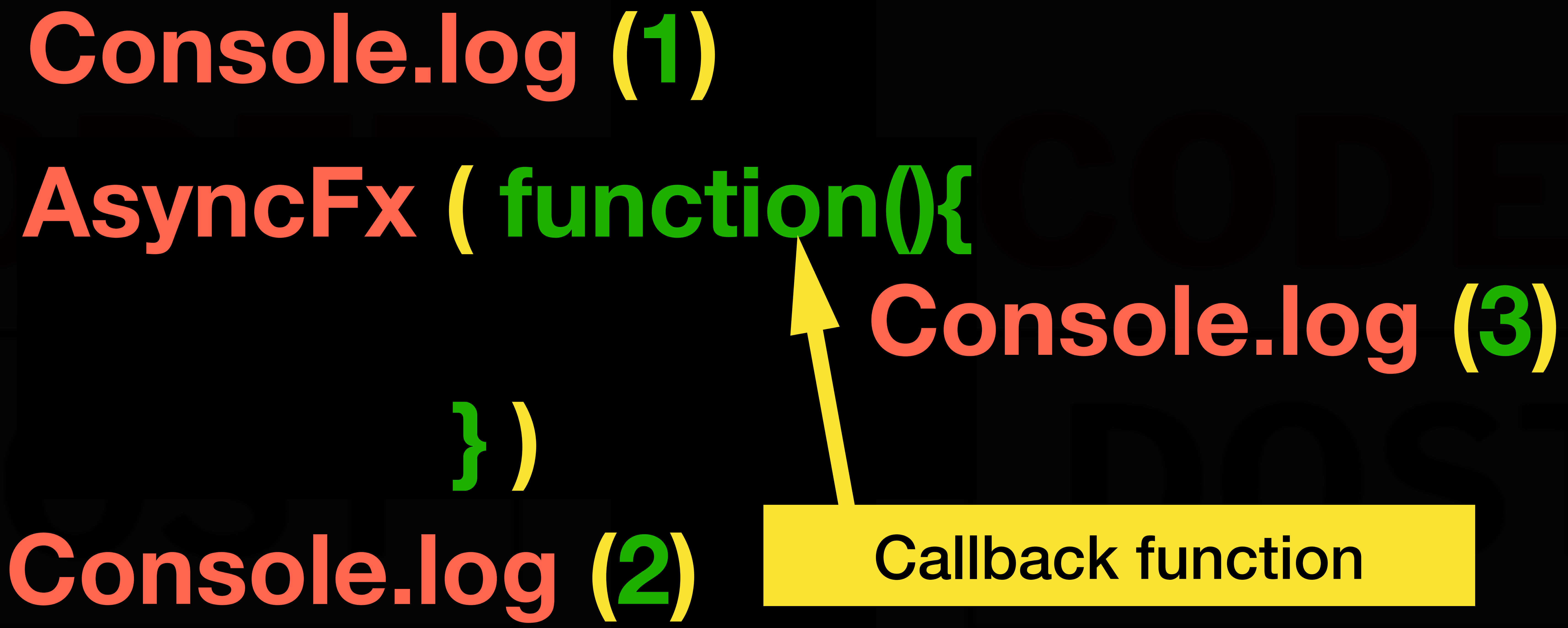
Async JavaScript

```
function double(x){ return x*2};

setTimeout((cb)=>{
      cb(double(4));
},
4000,
print)

function print(result){
    console.log(result)
}
```

3rd argument

callback function

Async JavaScript



```
Console.log (1)
AsyncFx ( function(){
                              Console.log (3)

              } )
Console.log (2)
```

Callback function

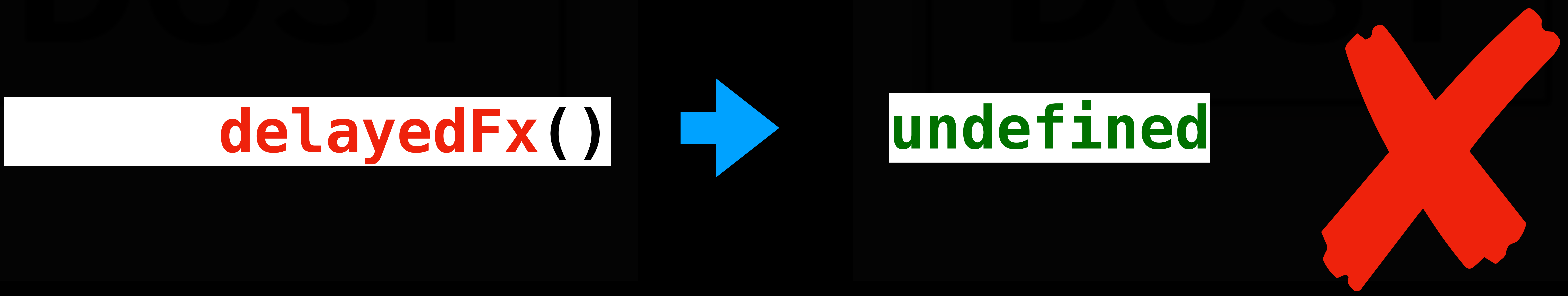
Async JavaScript

```
AsyncFx ( function(){

       AsyncFx ( function(){

              AsyncFx ( function(){
                      } )

               } )

        } )
```

Callback Hell

Promise

```
function delayedFx(){
    setTimeout(function(){
        someAction();
    }, 3000);
}
```



delayedFx() → undefined

As “someAction” will run late - its output can’t be returned

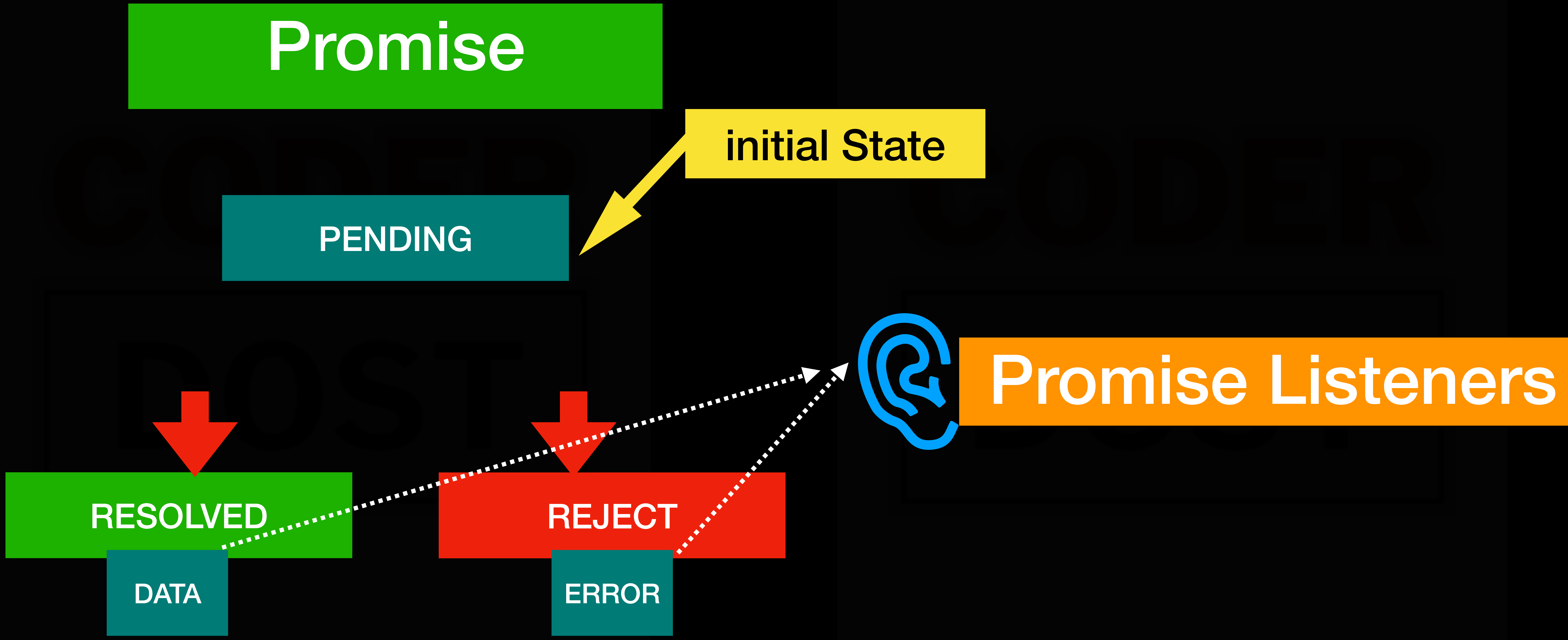
Promise

```
function delayedFx(){
    setTimeout(function(){
        someAction();
    }, 3000);
    return promise;
}
```

delayedFx() → promise ✓

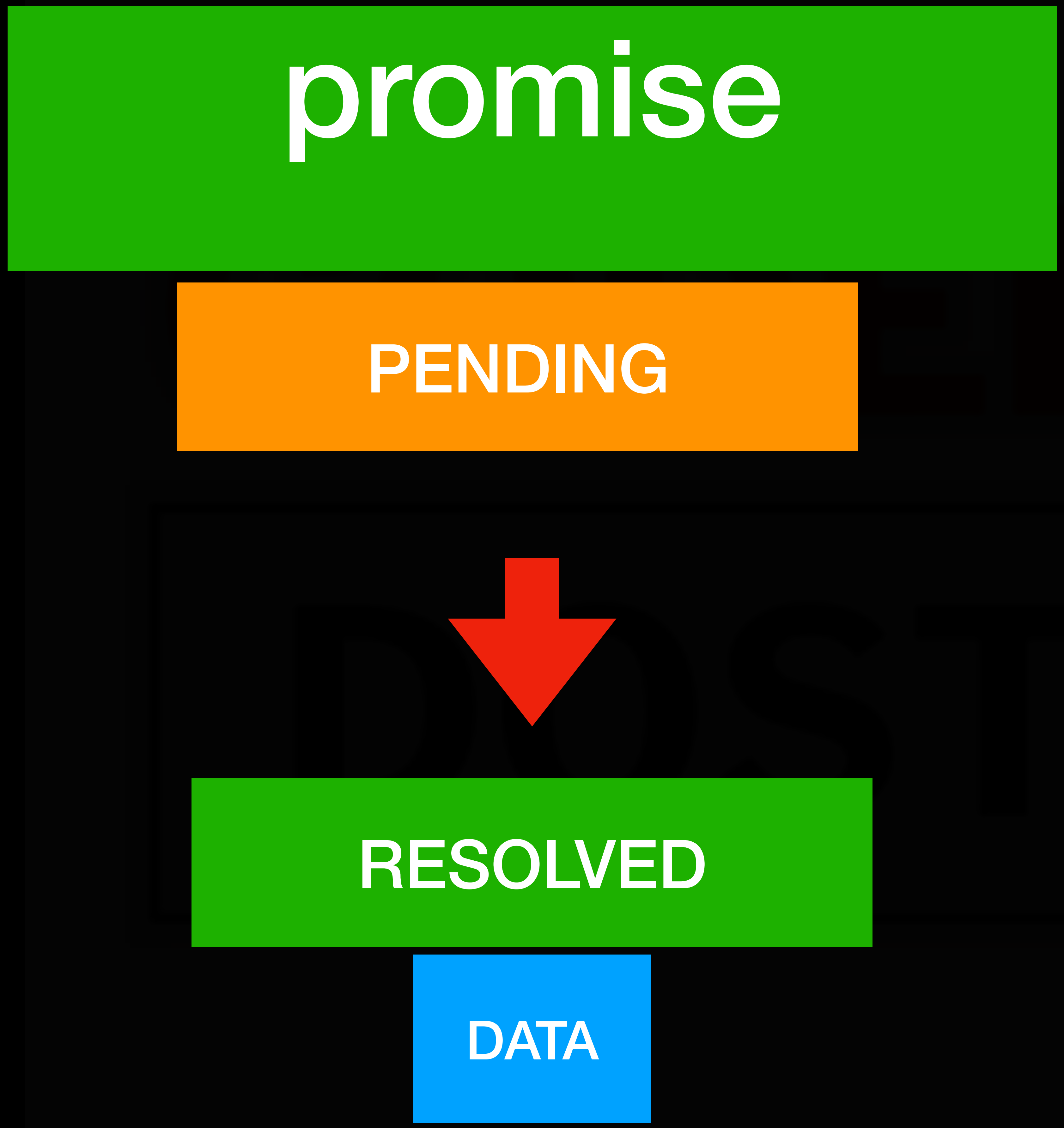
A ‘promise’ is something which can return value in future

Promise : States



https://www.youtube.com/@coderdost

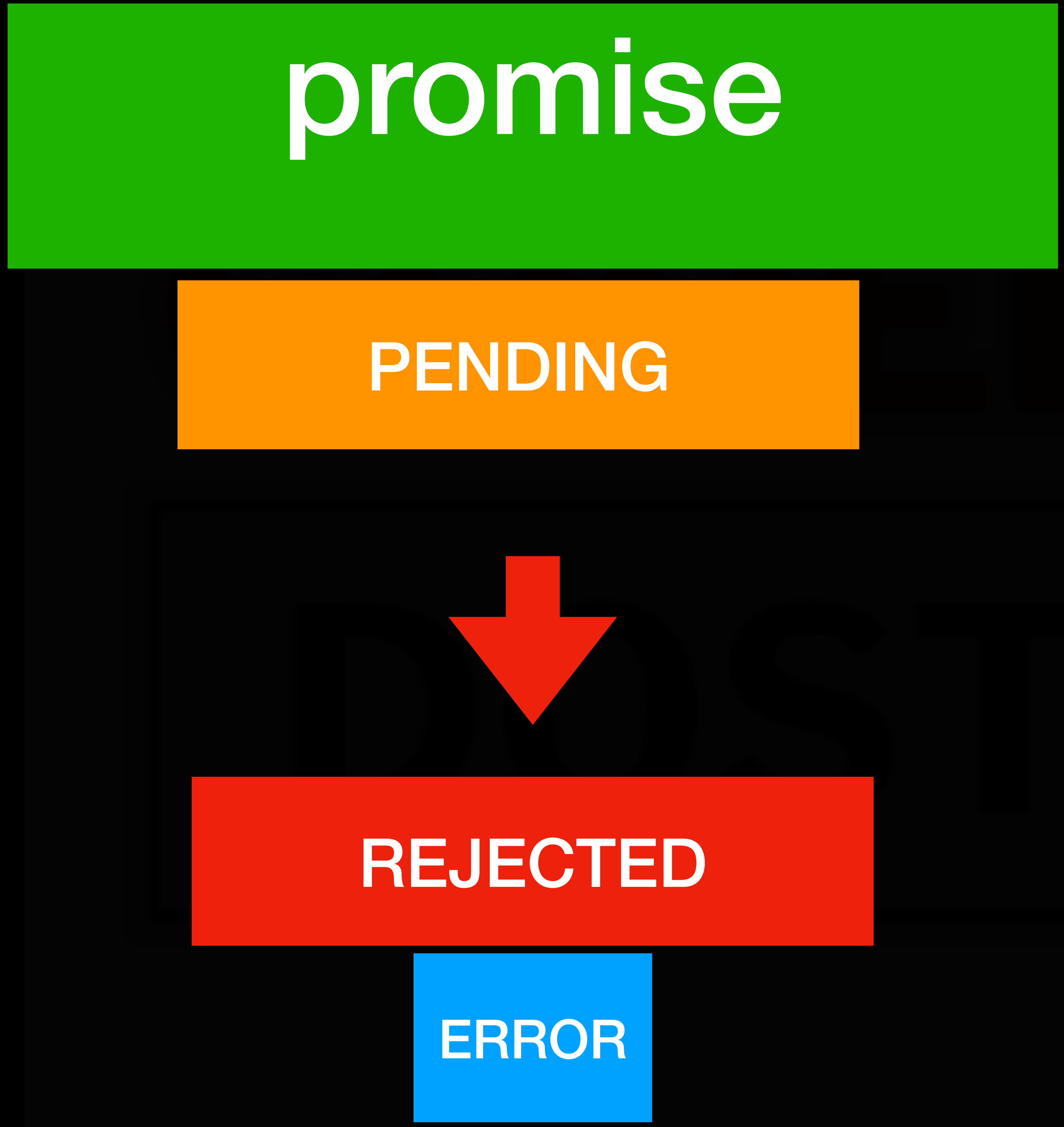
Promise : States



```
new Promise(function(resolve,reject){
      resolve(data);

})
```

runs after a long operation

Promise : States



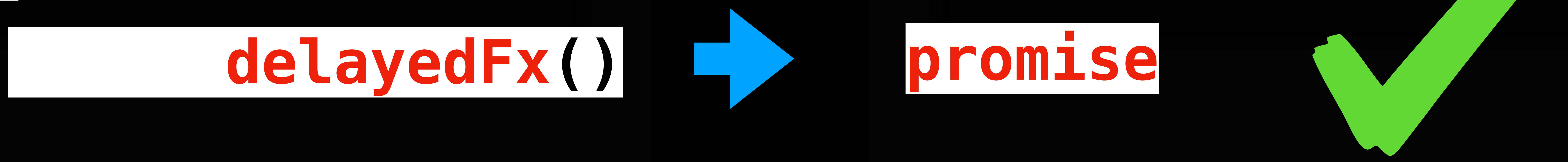
```
new Promise(function(resolve,reject){
      reject(error);

})
```

runs after a long operation

Promise

```
function delayedFx(){
  let promise = new Promise((resolve, reject)=>{
       setTimeout(function(){
            resolve(someAction());
       }, 3000);
  }
       return promise;
}
```

resolve will send data to Promise listeners (.then)



A ‘promise’ is returned but it will “resolve” later

Promise : .then() & .catch()

```
const p  = new Promise(function(resolve,reject){

})

p.then(function(data){

}).catch(function(data){

})
```

then will run but
Callback waits

catch will run but
Callback waits

Promise : Resolved

```
const p  = new Promise(function(resolve,reject){
    resolve(data);

})

p.then(function(data){


}).catch(function(data){


})
```

callback runs after resolve()

Promise : Rejected

```
const p  = new Promise(function(resolve,reject){
            reject(data);
})

p.then(function(data){

}).catch(function(data){

})
```

callback runs after reject()

Fetch

```
fetch(URL, optionsObject)
```

```
fetch(“http://google.com/)
```

```
fetch(“http://cricket.com/, {
     “method” : “POST”,
     “body” : “match score”
})
```

→ `promise`

Fetch

```
fetch(“http://cricket.com/, {
     “method” : “POST”,
     “body” : “match score”
})

.then(function(HTTPResponse){

}).catch(function(ErrorResponse){

})
```

Await

```
const P  = new Promise(function(resolve,reject){
       resolve(data);
})
```

```
const data = await P
```

await will sleep at this line

```
console.log(data)
```

This line runs after resolve

Async

```
const P  = new Promise(function(resolve,reject){
         resolve(data);
})

  async  function(){
     const data = await P
     console.log(data)
  }
```

✓

We always use “async” for await based functions

JSON (JavaScript Object Notation)

```
{
     “name” : "abhishek",
     “age” : ”30” ,
     “address” : "street 10, Mumbai, India",
     “phone”: “888888888”
 }
```

quotes on properties

HashMaps

Dictionary

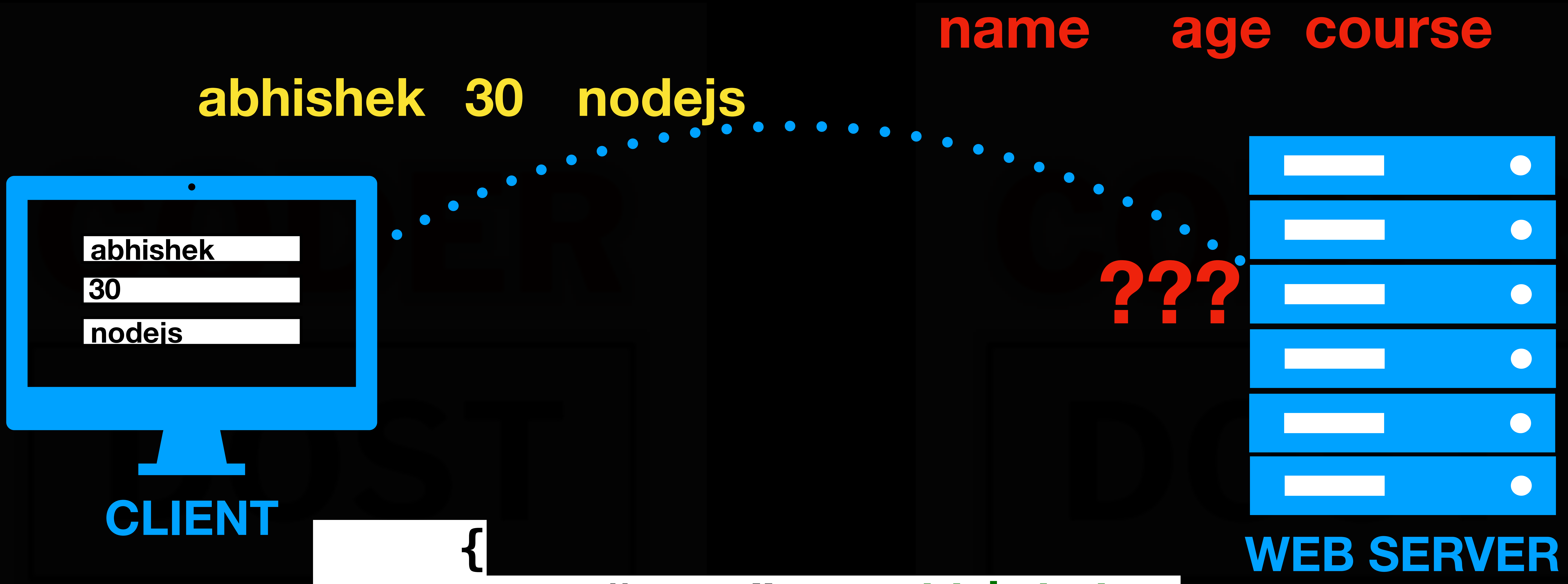
https://www.youtube.com/@coderdost

Universal Data Transfer

```
     {
          “name” : "abhishek",
          “age” : ”30” ,
          “course” : "nodejs",
      }
```

Can be understood by any Programming Language

Universal Data Transfer format (JSON)



```
       {
              “name” : "abhishek",
              “age” : ”30” ,
              “course” : "nodejs",
        }
```

https://www.youtube.com/@coderdost

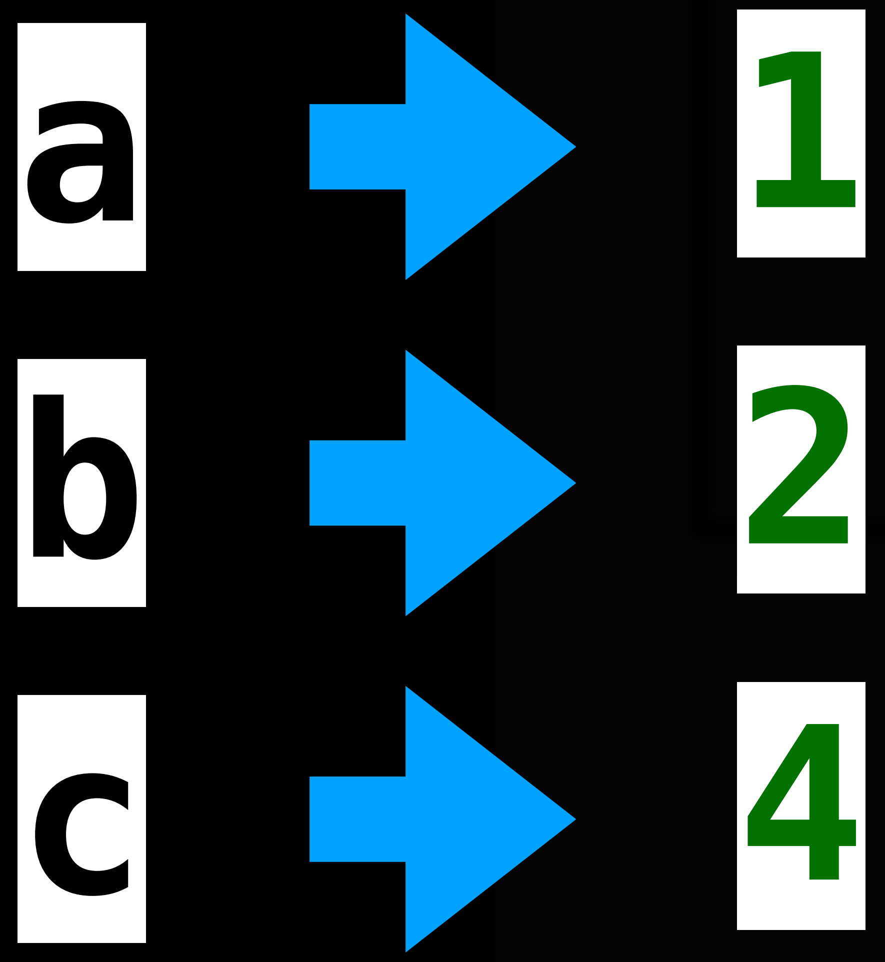
12. ES 6 JavaScript

https://www.youtube.com/@coderdost

De-structuring Assignment Array

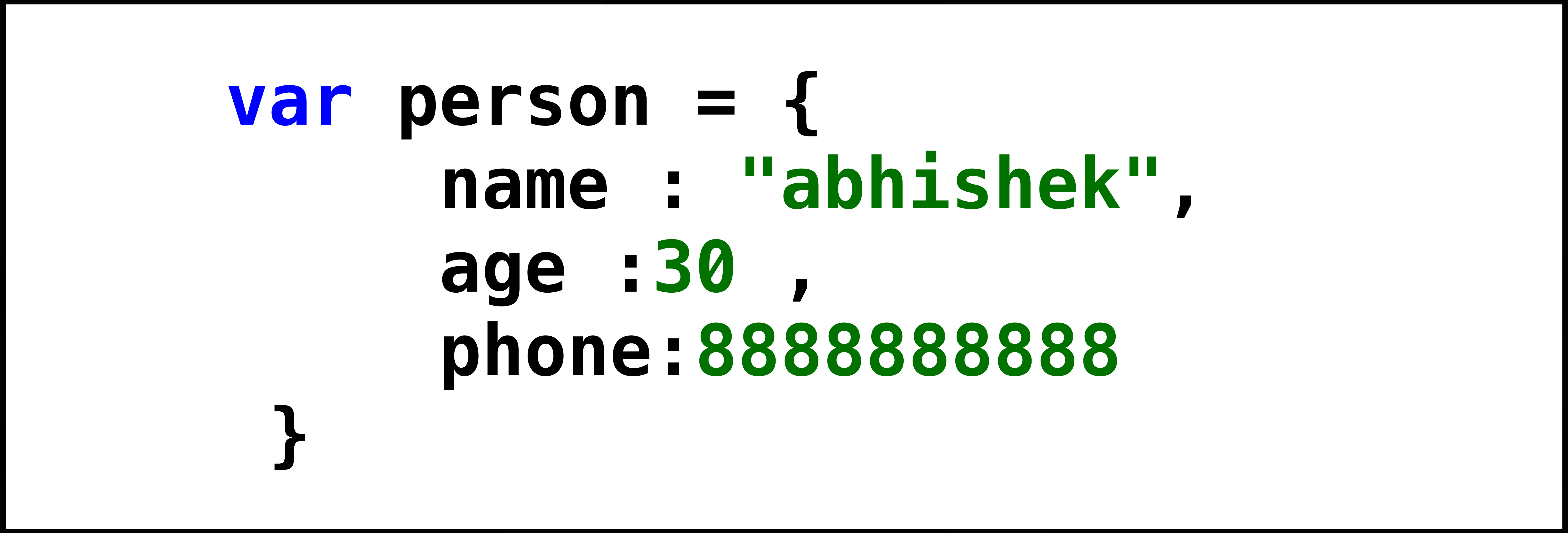
```
nums = [1, 2, 4];
```

```
const [a,b,c] = nums;
```

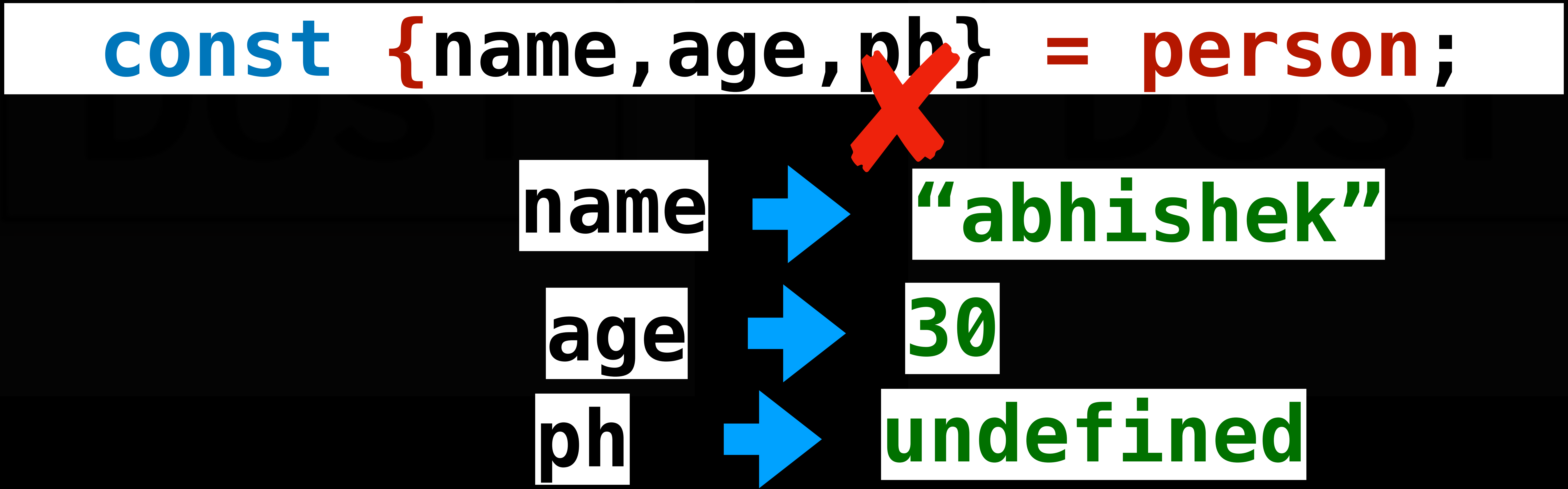


https://www.youtube.com/@coderdost

De-structuring Assignment Objects



```
var person = {
        name : "abhishek",
        age :30 ,
        phone:8888888888
 }
```



```
const {name,age,ph} = person;
```

name → “abhishek”

age → 30

ph → undefined

Spread Operator

```
Math.max(4, 5, 100, 0, 1)
```

→ 100

```
const numbers = [4, 5, 100, 0, 1];
```

```
Math.max(numbers)
```

✗

Spread Operator

```
const numbers = [4, 5, 100, 0, 1];
```

```
…numbers
```

→

```
4, 5, 100, 0, 1
```

Math.max(...numbers) ✓

Math.max(4, 5, 100, 0, 1)

Rest Parameters

```
let max = function(...nums){
    // function body
}
```

```
max(1,2,4,5)
```

```
nums = [1, 2, 4, 5];
```

Rest Parameters

```
let max = function(...nums){
    // function body
}
```

```
max(1,2)
```

```
max(1,2,3)
```

```
nums = [1, 2];
```

```
nums = [1, 2, 3];
```

Short Circuiting

```
var person = {
    name: 'Jack',
    age: 34
  }
console.log(person.job || 'unemployed');
```

will stop at first value which is "truthy"

```
console.log(person.job && 'unemployed');
```

will stop at first value which is "falsy"

Nullish Coalescing (??)

```
const foo = null ?? 'default string’;
```

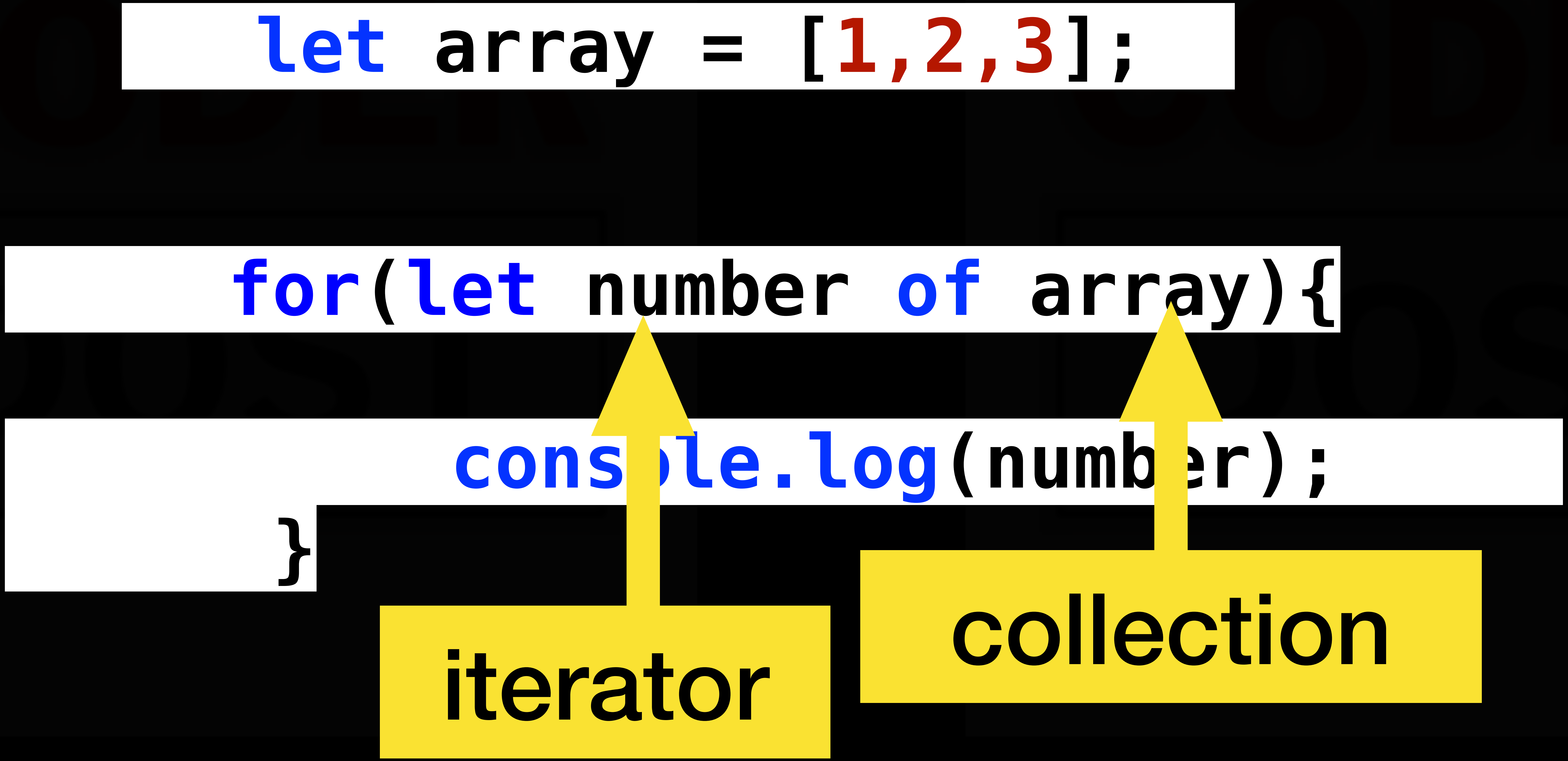
default string

```
const baz = 0 ?? 42;
```

0

https://www.youtube.com/@coderdost

FOR OF Loop



```
let array = [1,2,3];

for(let number of array){
    console.log(number);
}
```

iterator

collection

FOR OF Loop

ITERATION 1

```
let array = [1,2,3];

for(let number of array){
                  1
    console.log(number);      >   1
}
```

FOR OF Loop

ITERATION 2

```
let array = [1,2,3];

for(let number of array){
        console.log(number);
    }
```

2

```
>    1
>    2
```

FOR OF Loop

ITERATION 3

```
let array = [1,2,3];

for(let number of array){
                3
    console.log(number);
}
```

```
>    1
>    2
>    3
```

FOR V/S FOR OF Loop

FOR

Difficult to Write

Error chances high

More flexible

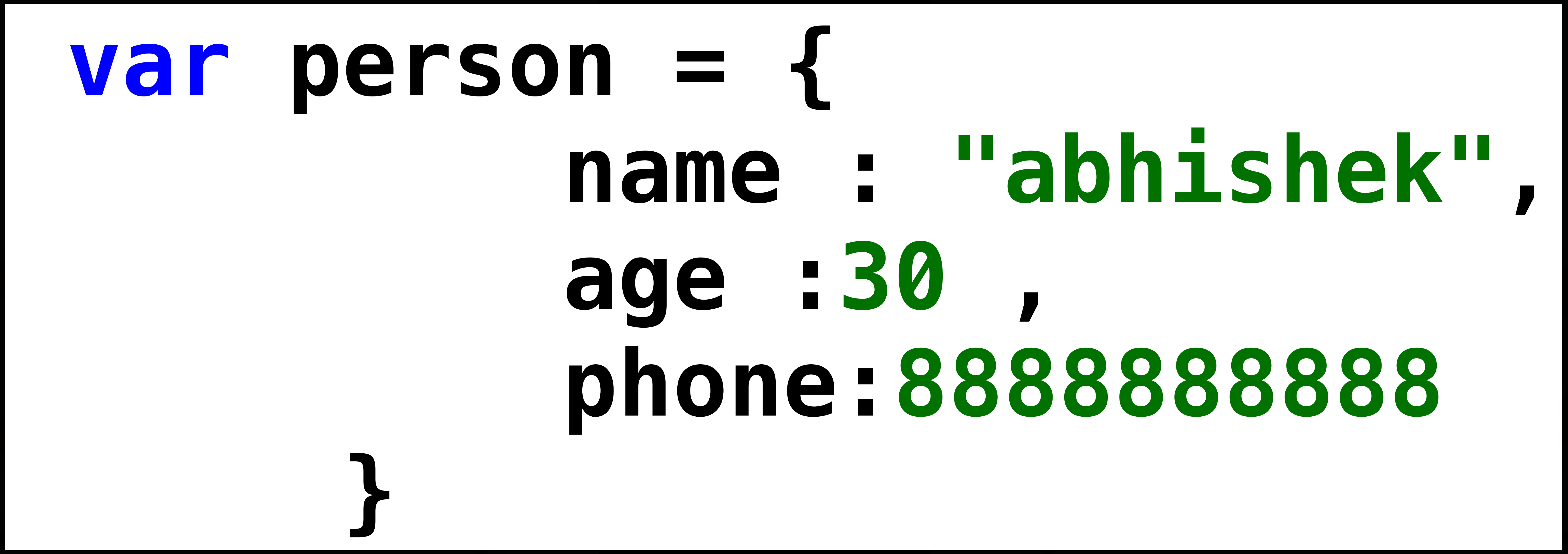
FOR OF

Easy to Write

Error chances low

Good for Loops which iterate each element

Object Literals: properties



```
var person = {
         name : "abhishek",
         age :30 ,
         phone:8888888888
     }
```

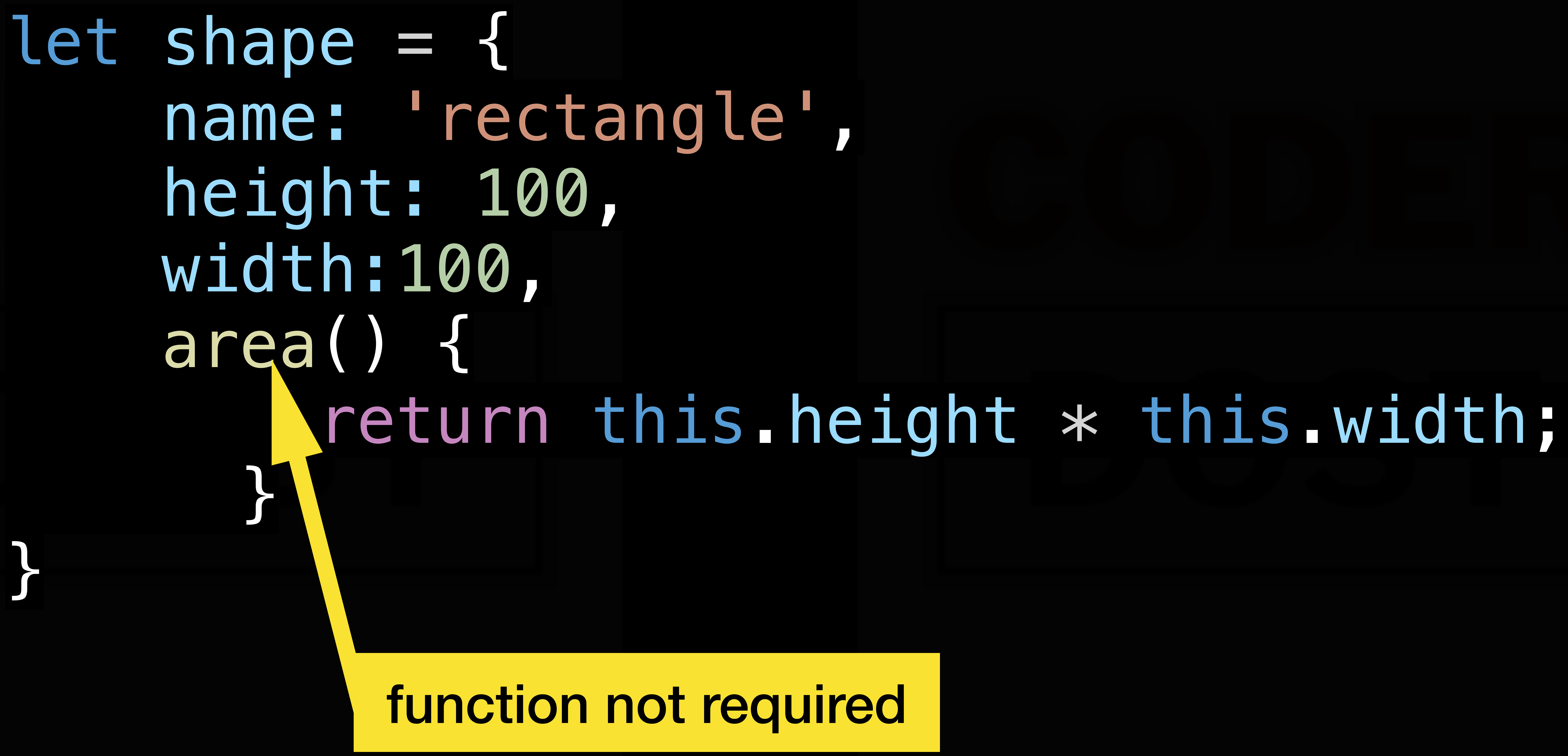
Object Literals: properties

```
var name = “abhishek";
var age = 30 ;
var phone = 8888888888 ;
```

```
 var person = {
                name ,
                age ,
                phone
          }
```

Shorthand Object Literals

Object Literals : methods



```
let shape = {
  name: 'rectangle',
  height: 100,
  width:100,
  area() {
    return this.height * this.width;
   }
}
```

function not required

Object Literals : Computed keys

```
let keyName = “nameofShape"
```

```
let shape = {
    [keyName]: 'rectangle',
    height: 100,
    width:100,
}
```

key is variable here

```
shape.nameofShape
```

→ **“rectangle”**

https://www.youtube.com/@coderdost

Optional Chaining (?.)

```
let person = {
    username: 'Jack',
    age: 34
  }
```

```
const fName = person?.name?.firstName;
```

undefined

checks if a property exists then only moves ahead.
Avoids error

Object Methods

new Object ()

Object Constructor

Used for creating Objects.

But we generally use { } for easier writing

```
var person = {
            name : "abhishek",
            age :30 ,
            address : "street 10",
            phone:8888888888
        }
```

Object.keys(person) → [“name”,”age”,”address”,”phone”]

Object.values(person) → [“abhishek”,30,”street 10”,8888888888]

Object.entries(person) → [[“name:“abhishek”], [“age”,30] , [“address”: “street 10”], [“phone”: 8888888888]]

https://www.youtube.com/@coderdost

Set : Add

```
let set = new Set()

set.add(1)          Set(1) {1}
set.add(5)          Set(2) {1,5}
set.add(4)          Set(3) {1,5,4}
set.add(5)          Set(3) {1,5,4}
set.add(1)          Set(3) {1,5,4}
set.add(10)         Set(4) {1,5,4,10}
```

A Set only keeps unique value

Set : size

Set(4) {1, 2, 3, 4}

```
set.size
```

→

```
4
```

Set : Delete

```
Set(4) {1, 2, 3, 4}
```

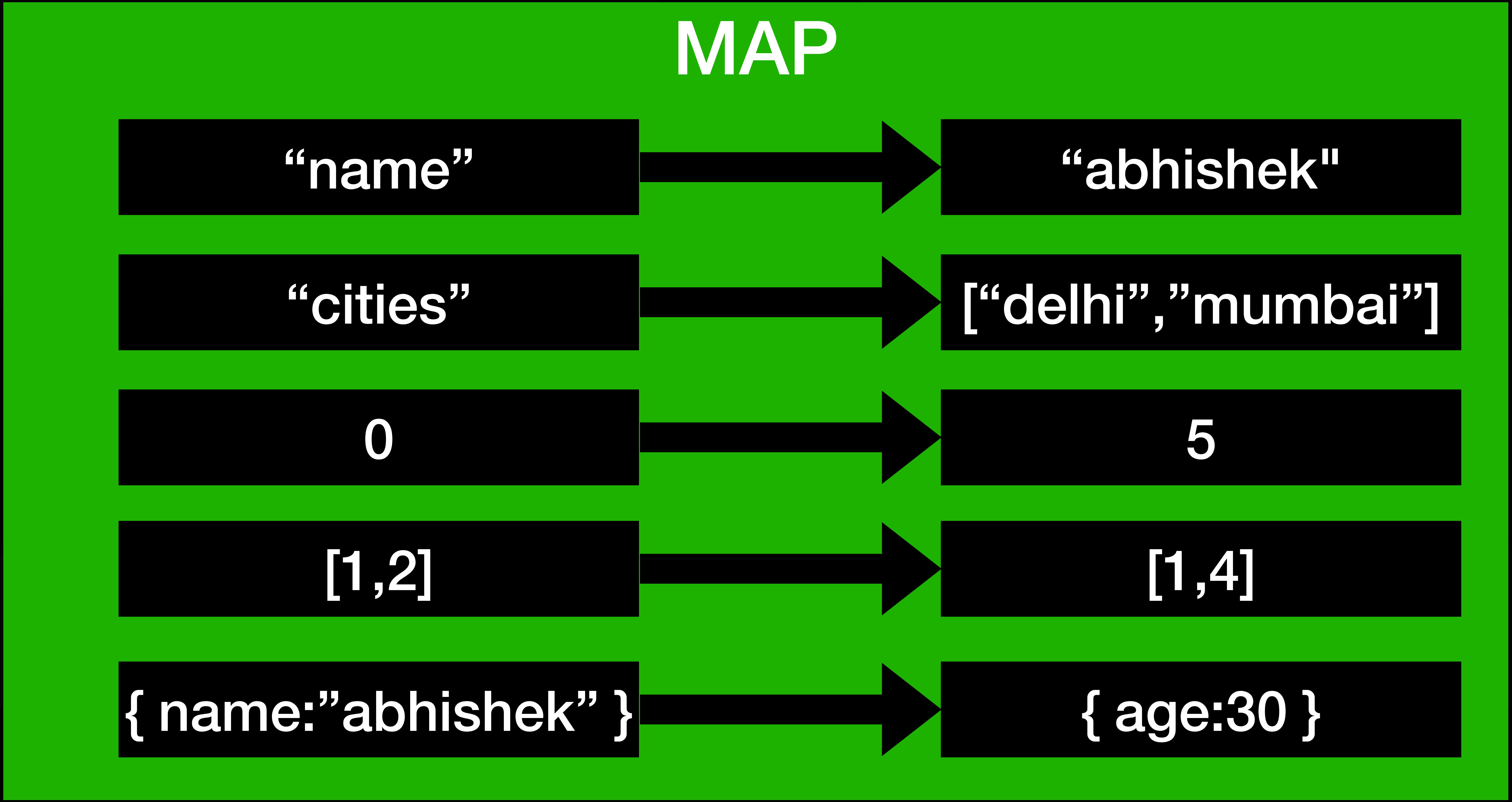
```
set.delete(1)        Set(3) {2, 3, 4}

set.delete(2)        Set(2) {3, 4}

set.delete(4)        Set(1) {3}
```

Set : has & clear

Set(4) {1, 2, 3, 4}

```
set.has(3)   →   true

set.has(5)   →   false

set.clear()      Set(0) { }
```

Map Data Type



MAP

“name” → “abhishek"

“cities” → [“delhi”,”mumbai”]

0 → 5

[1,2] → [1,4]

{ name:”abhishek” } → { age:30 }

Map : Write and Read

```
let map = new Map()

map.set(“name”,”abhishek”)

map.set([1, 2],[1, 4])

map.get([1, 2])  →  [1, 4]  ✗

map.get(“name”)  →  ”abhishek”
```

https://www.youtube.com/@coderdost

Map : Check Exists

```
let map = new Map()

map.set(“name”,”abhishek”)

map.set([1, 2],[1, 4])

map.has(“age”)
```

→ false

Map : Delete

```
let map = new Map()

map.set(“name”,”abhishek”)

map.set([1, 2],[1, 4])

map.delete(“name”)
```

Map : Clear all

```
let map = new Map()

map.set(“name”,”abhishek”)

map.set([1, 2],[1, 4])

map.clear()
```

Clear all values

Map : Length

```
let map = new Map()

map.set(“name”,”abhishek”)

map.set([1, 2],[1, 4])
```

`map.size` → `2`

13. Misc Tools

Import in JS

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
<script src=“index.js" type=“module”>
</script>
</head>
<body>

</body>
```

ES module enable

index.html

https://www.youtube.com/@coderdost

Named Export / Import

```
const a = 5;
const b = 6;

function sum(a,b){
    return a+b;
}

export {a,b,sum};
```

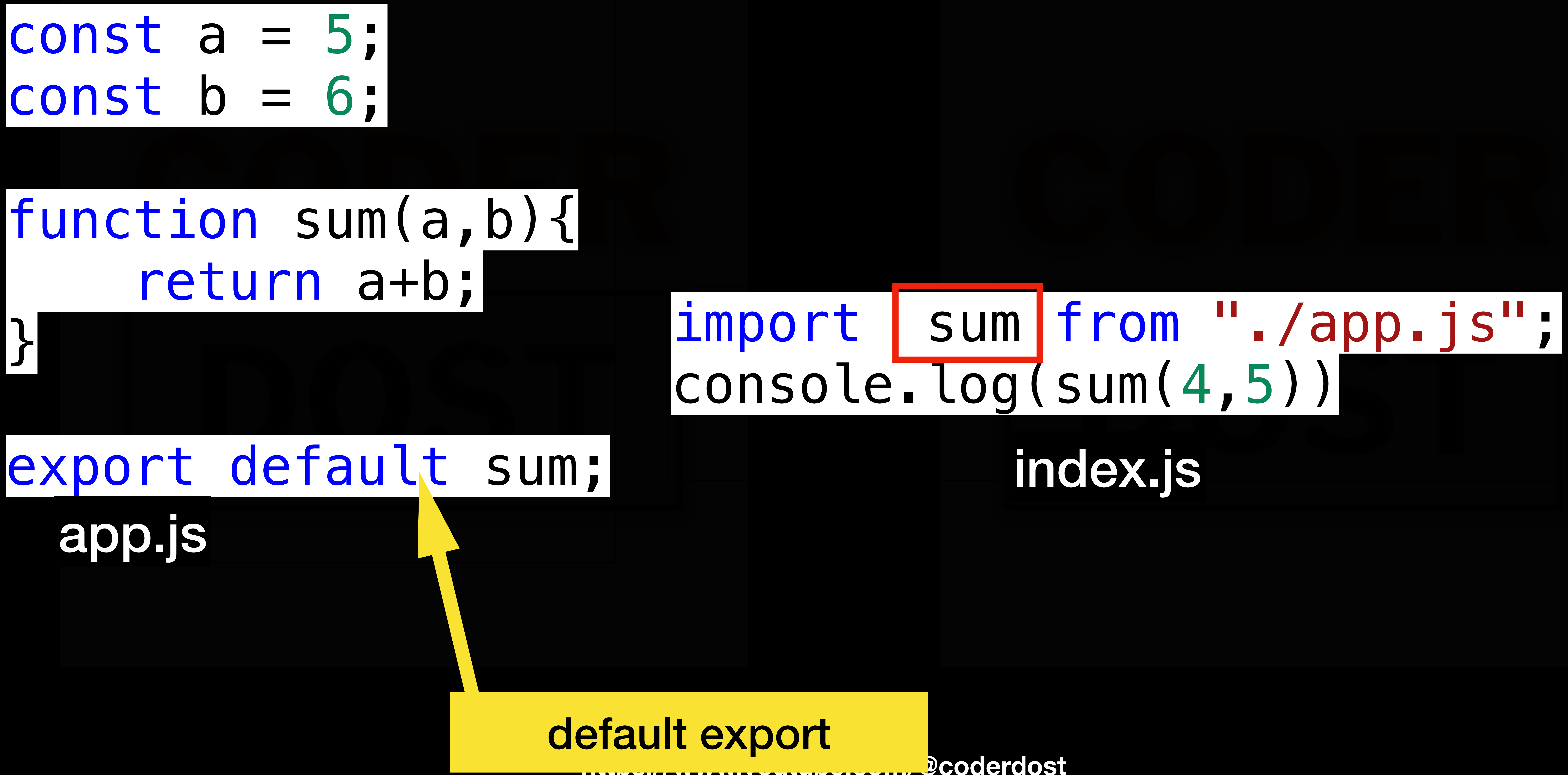
app.js

named exports

```
import {a,b,sum} from "./app.js"
console.log(sum(a,b))
```

index.js

Default Export / Import



```
const a = 5;
const b = 6;

function sum(a,b){
    return a+b;
}

export default sum;
```

app.js

```
import  sum from "./app.js";
console.log(sum(4,5))
```

index.js

default export

Alias

```
function sum(a,b){
    return a+b;
}

export {sum};
```

app.js

named exports

```
import {sum as add} from "./app.js
console.log(add(a,b))
```

index.js

Top Level Await

Now its allowed to use Await without Async at top-level of file/module

```
const x = await resolveAfter2Seconds(10);
console.log(x)
```

blocks the code

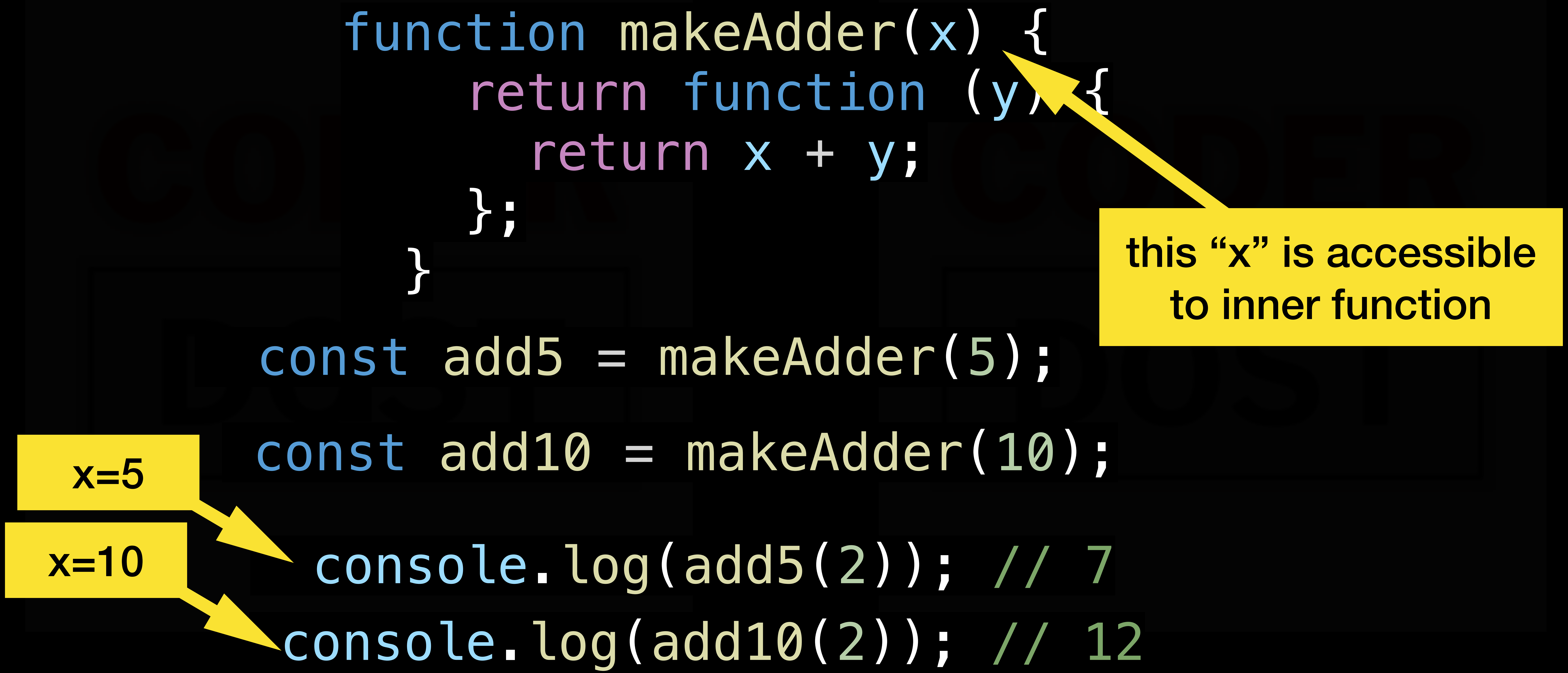
Modular code - IIFE

```
let sum = (function (a,b) {
     return a + b

})();
```

protects inner variables

14. Advanced Concepts

Closure



```
   function makeAdder(x) {
       return function (y) {
         return x + y;
       };
     }
const add5 = makeAdder(5);

const add10 = makeAdder(10);

  console.log(add5(2)); // 7
 console.log(add10(2)); // 12
```

this “x” is accessible to inner function

x=5

x=10

👍

THANKS

https://www.youtube.com/@coderdost