JAVASCRIPT - DAY 1

WHAT IS JAVASCRIPT?

THE LANGUAGE OF THE WEB

- Programming Language
- Used alongside HTML/CSS to create interactive websites
- Backbone of the modern web
- If a webpage does anything dynamic, it’s probably using JavaScript
- Email, Chat, Video, Images, Forums, Games, etc.
- So versatile, we’re now building servers with it!

JAVASCRIPT - DAY 1

FUNDAMENTALS

JAVASCRIPT - DAY 1

VARIABLES

VARIABLES

VARIABLES

Variables are used to store values.

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var‘ keyword.

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var’ keyword.

EXAMPLE:

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var‘ keyword.

EXAMPLE:

```
var myName = “Bruce Wayne”;
```

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var’ keyword.

EXAMPLE:

Using the equals sign assigns values on the right to the variable on the left.

```
var myName = “Bruce Wayne”;
```

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var‘ keyword.

EXAMPLE:

Using the equals sign assigns values on the right

to the variable on the left.

```
var myName = “Bruce Wayne”;
```

VARIABLES

Variables are used to store values.

Variables are declared using the 'var' keyword.

EXAMPLE:

Using the equals sign assigns values on the right

to the variable on the left.

```
var myName = "Bruce Wayne";
```

The string value "Bruce Wayne" is now stored

in the variable 'myName'.

VARIABLES

Variables are used to store values.

Variables are declared using the ‘var‘ keyword.

EXAMPLE:

Using the equals sign assigns values on the right

to the variable on the left.

```
var myName = “Bruce Wayne”;
```

The string value “Bruce Wayne” is now stored

in the variable ‘myName‘.

VARIABLES

VARIABLES

You can think of a variable as a box that we can store data inside of.

VARIABLES

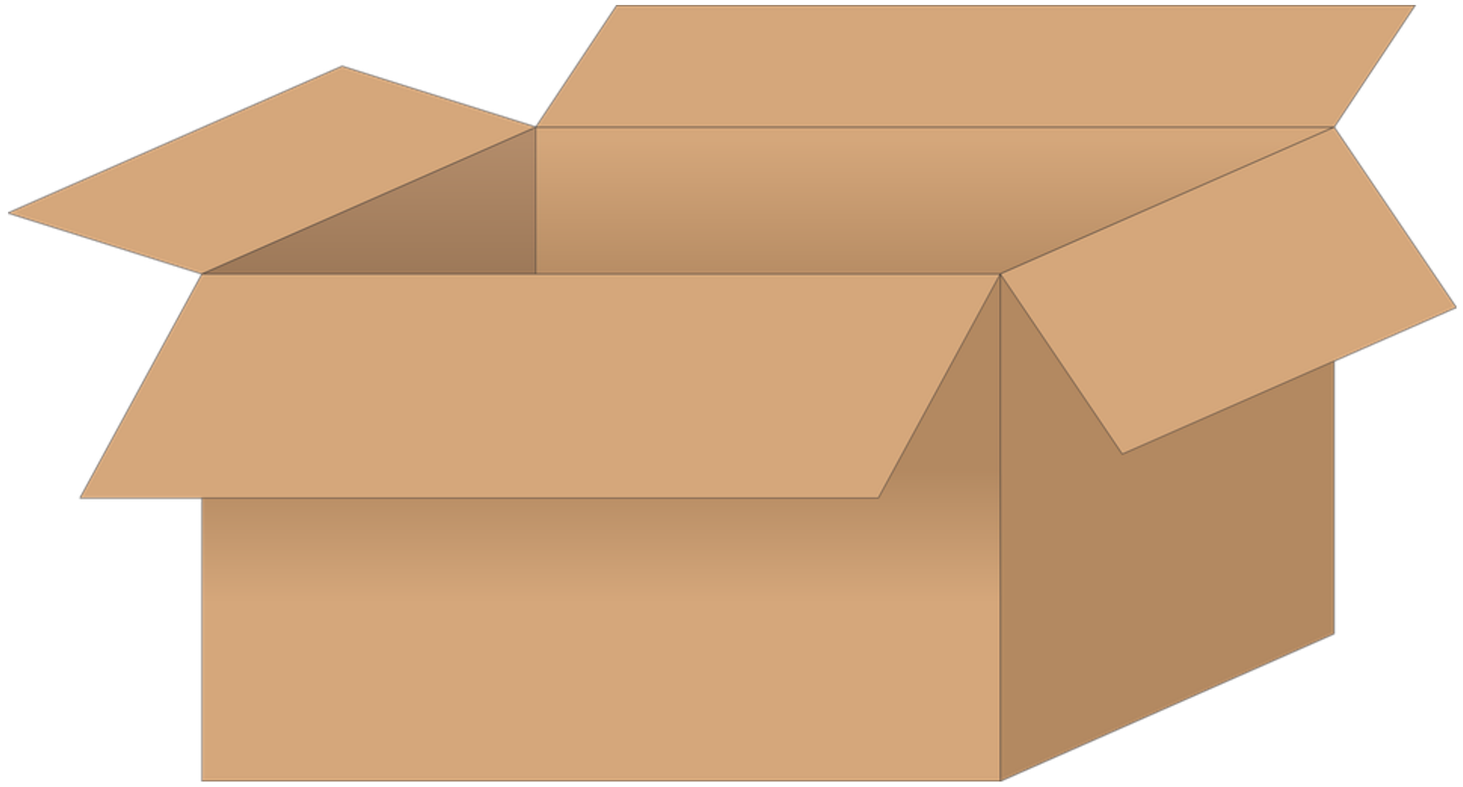
You can think of a variable as a box that we can store data inside of.

```
var age = 52;
```

VARIABLES

You can think of a variable as a box that we can store data inside of.

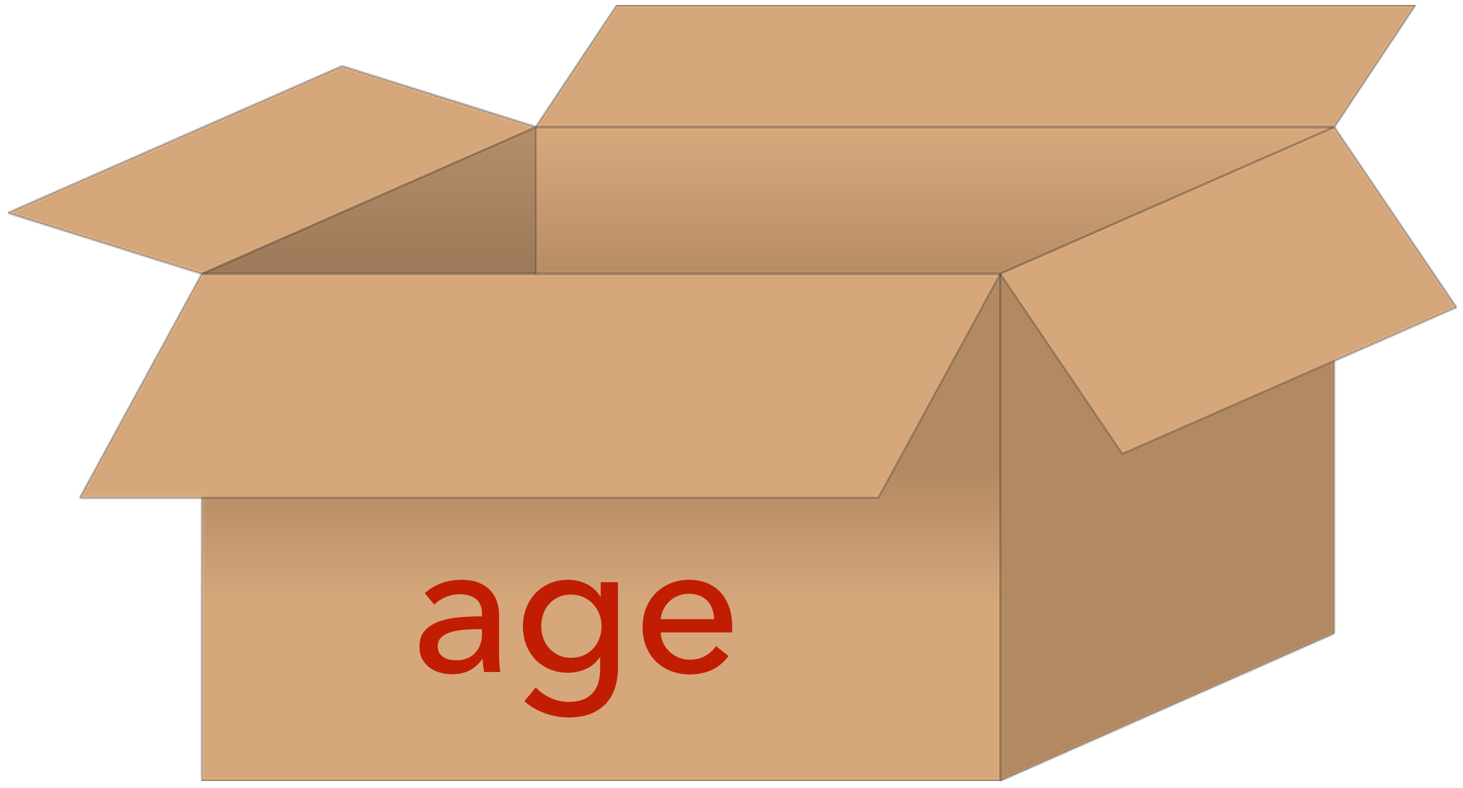
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

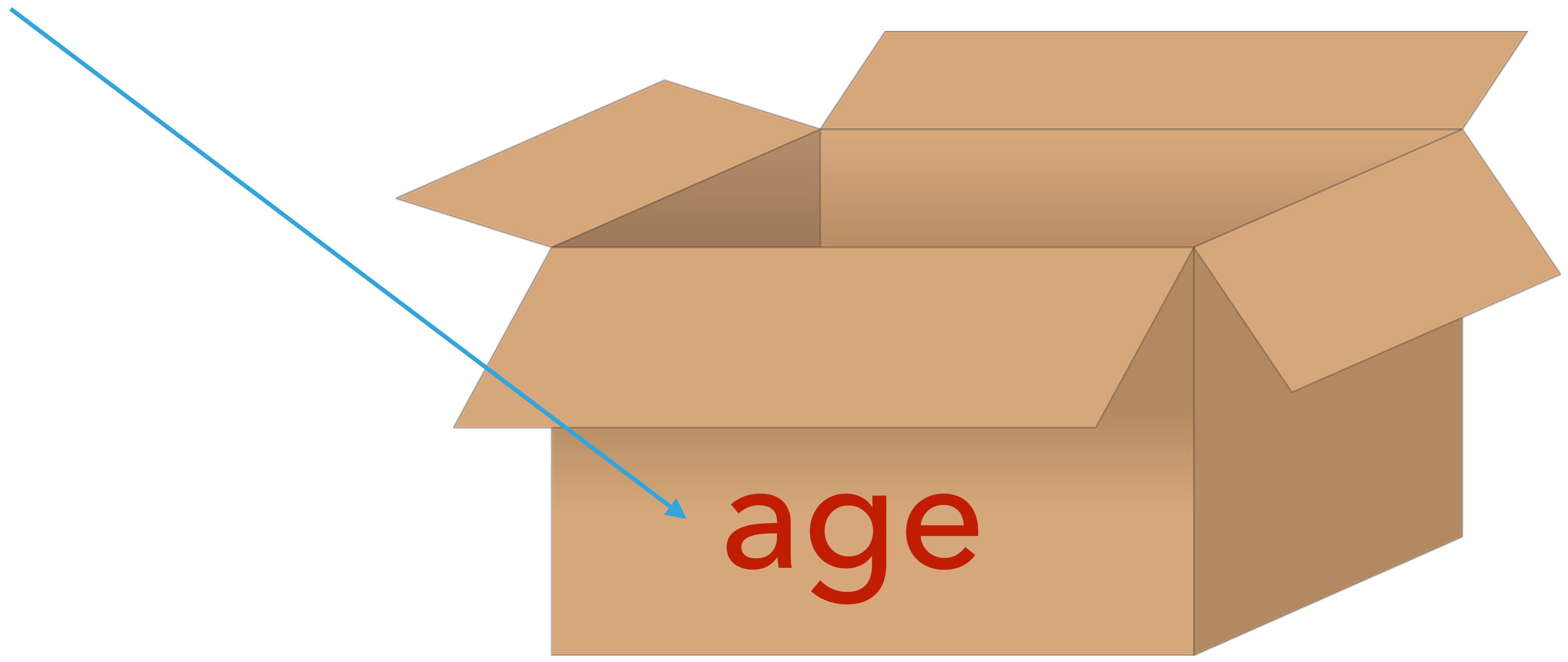
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

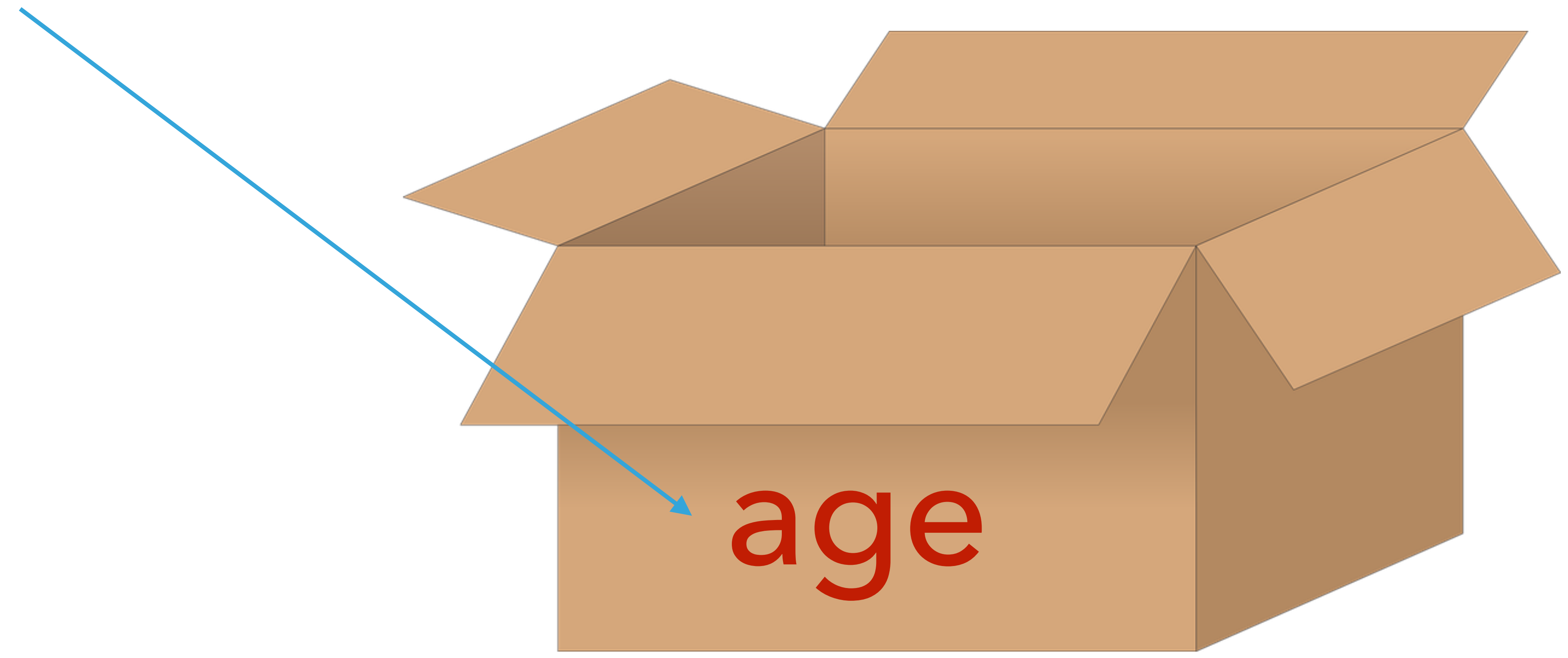
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

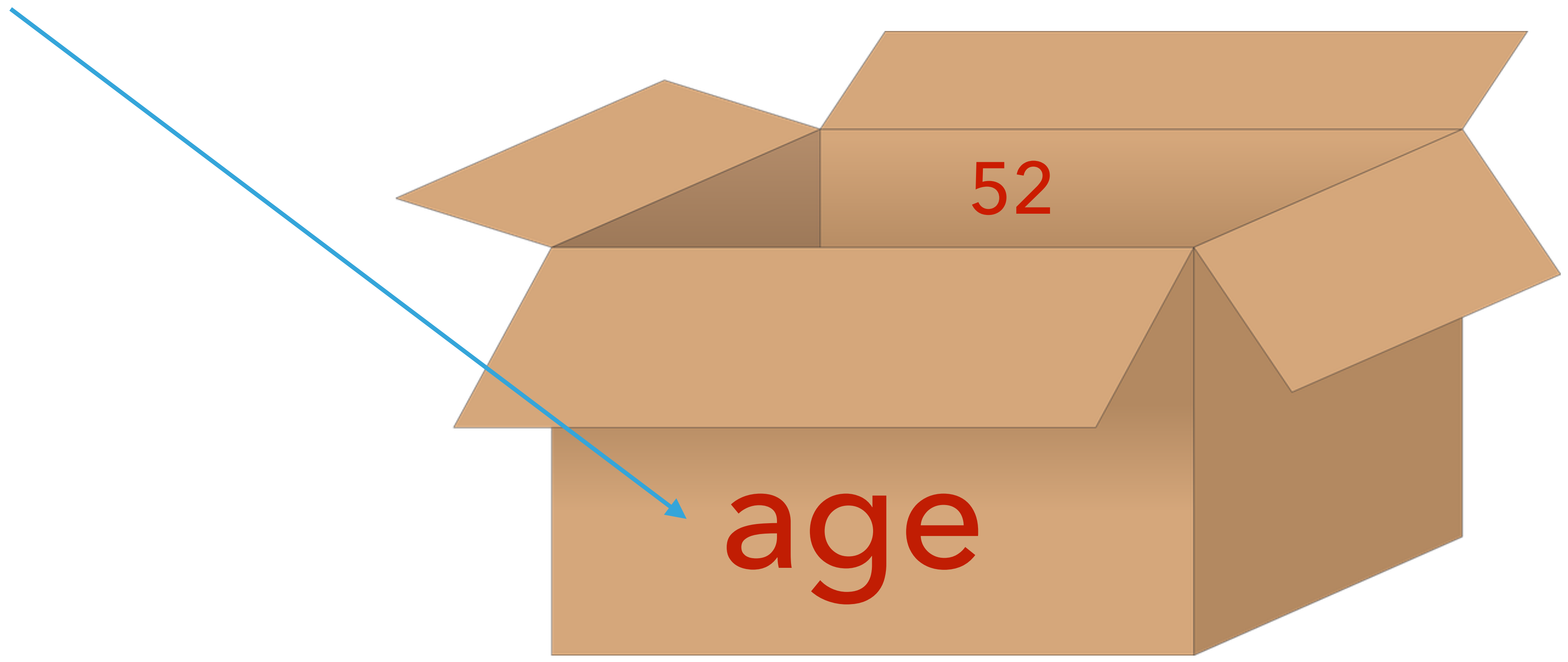
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

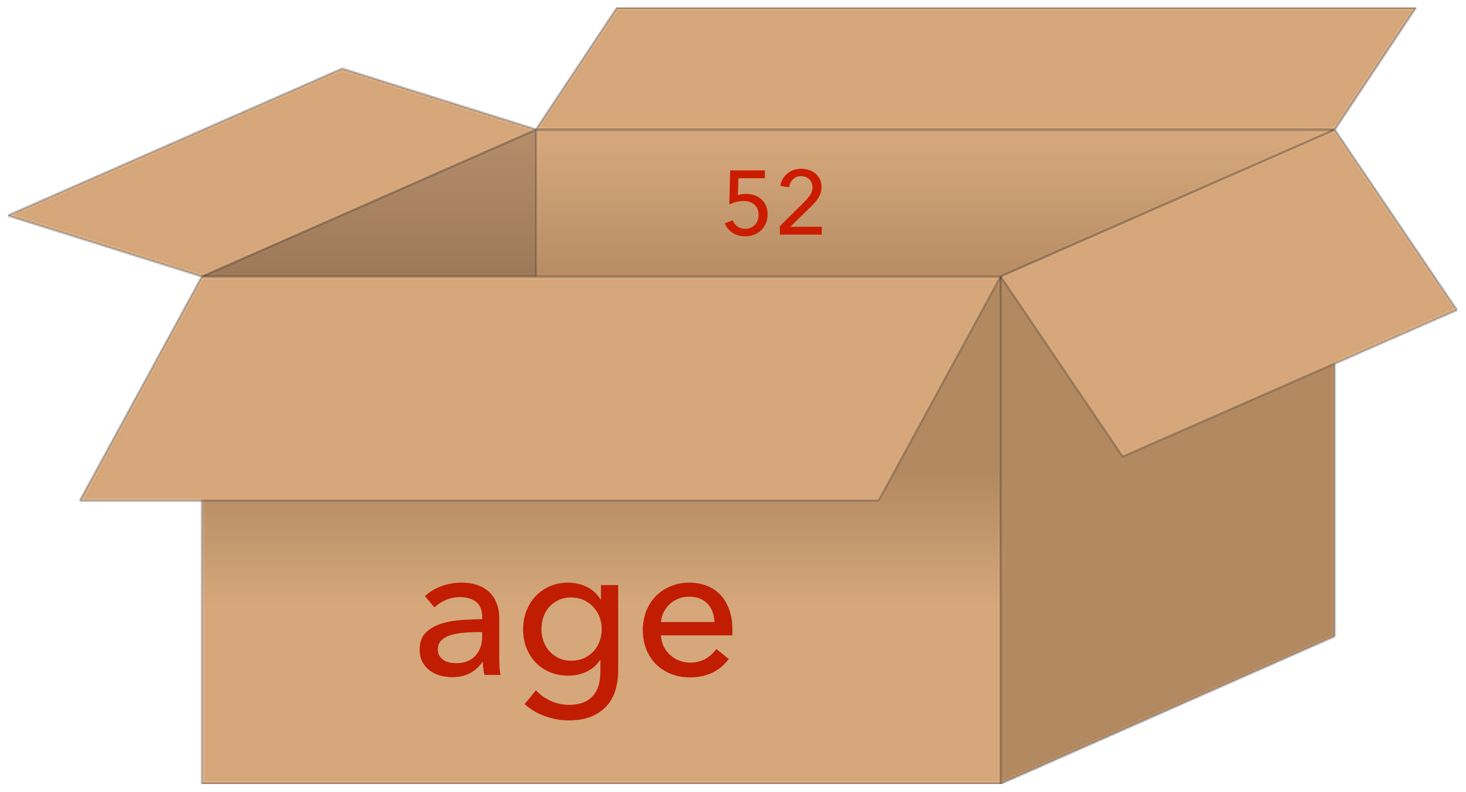
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

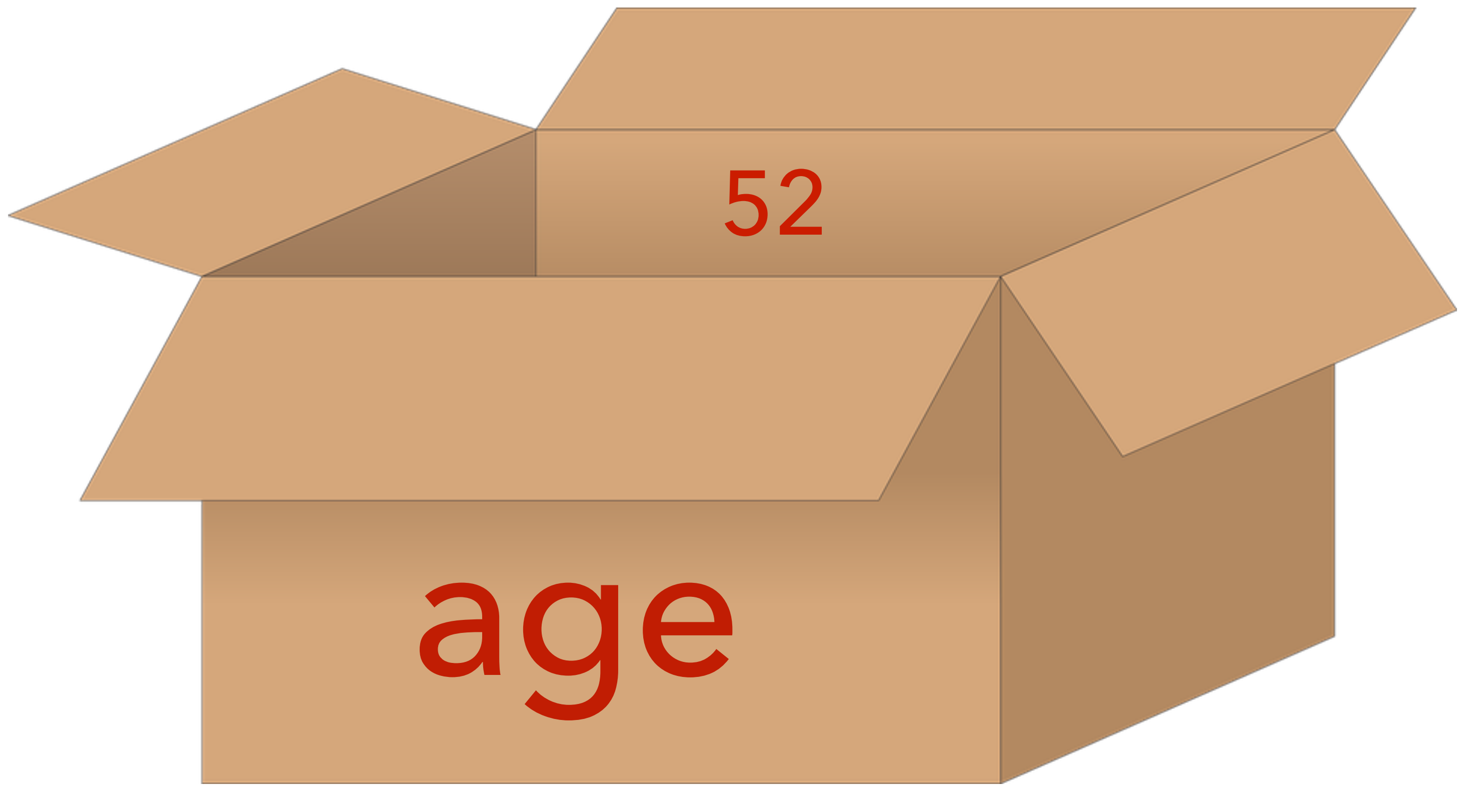
```
var age = 52;
```



VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

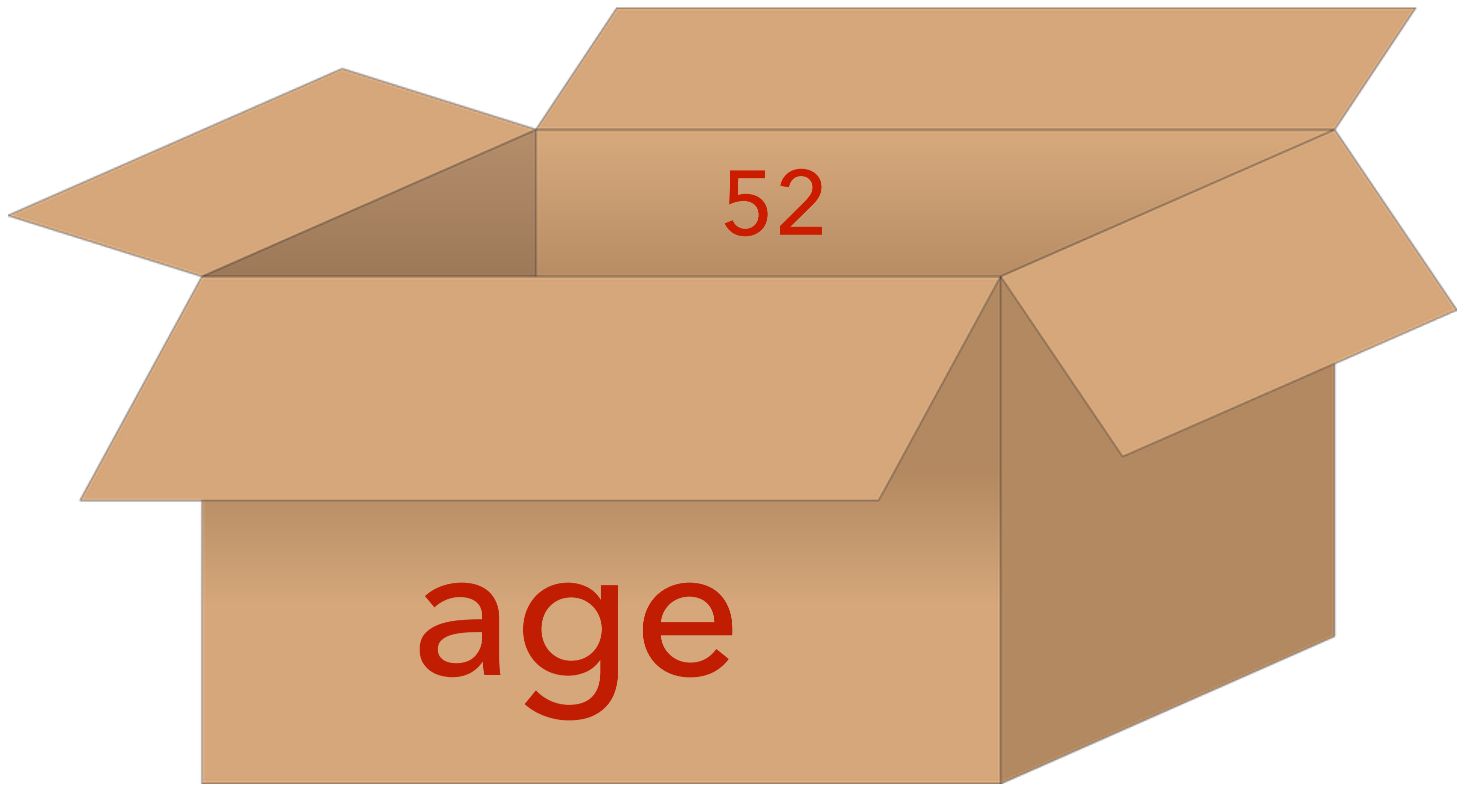


VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

New value assigned.



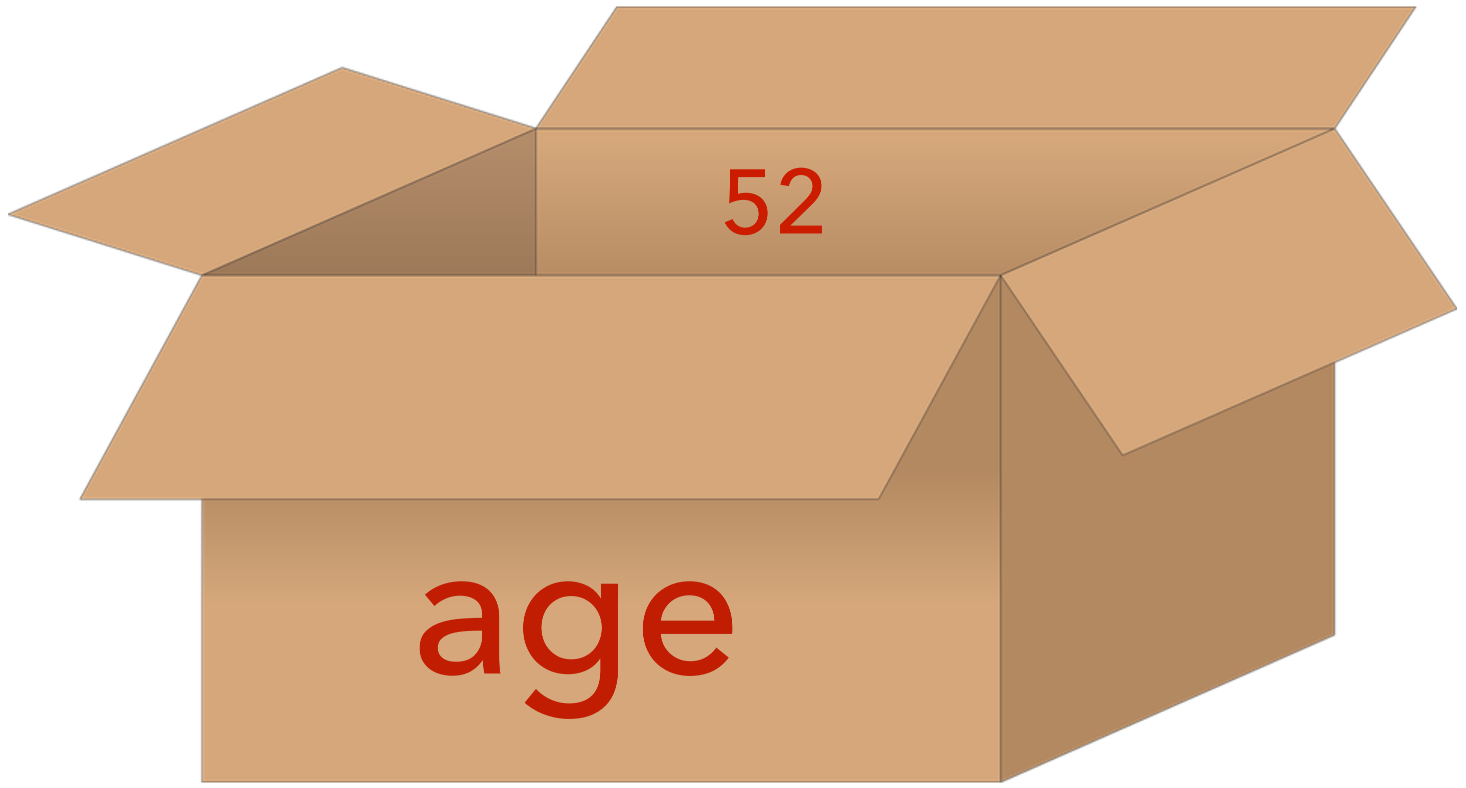
VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

↑

New value assigned.



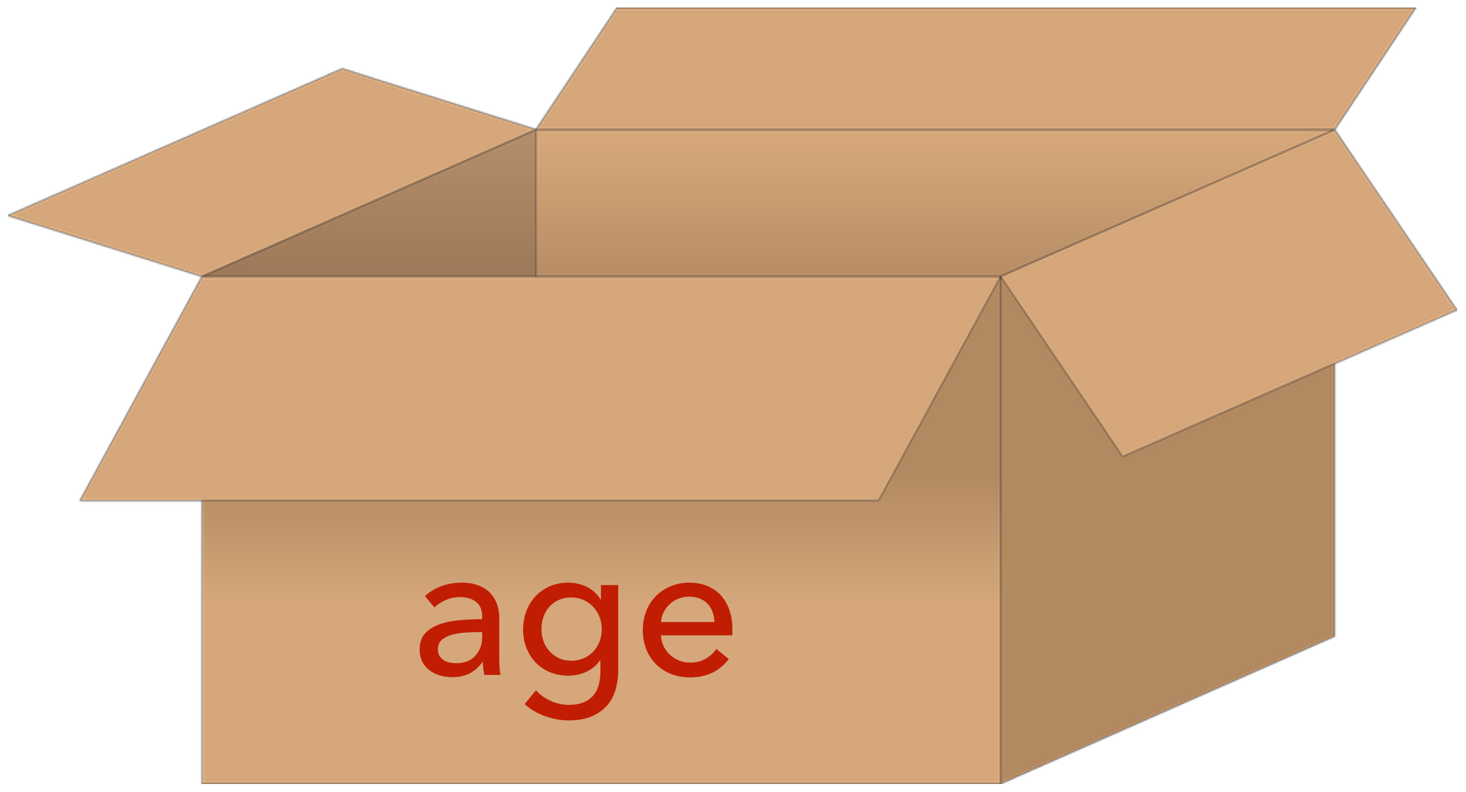
VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

↑

New value assigned.

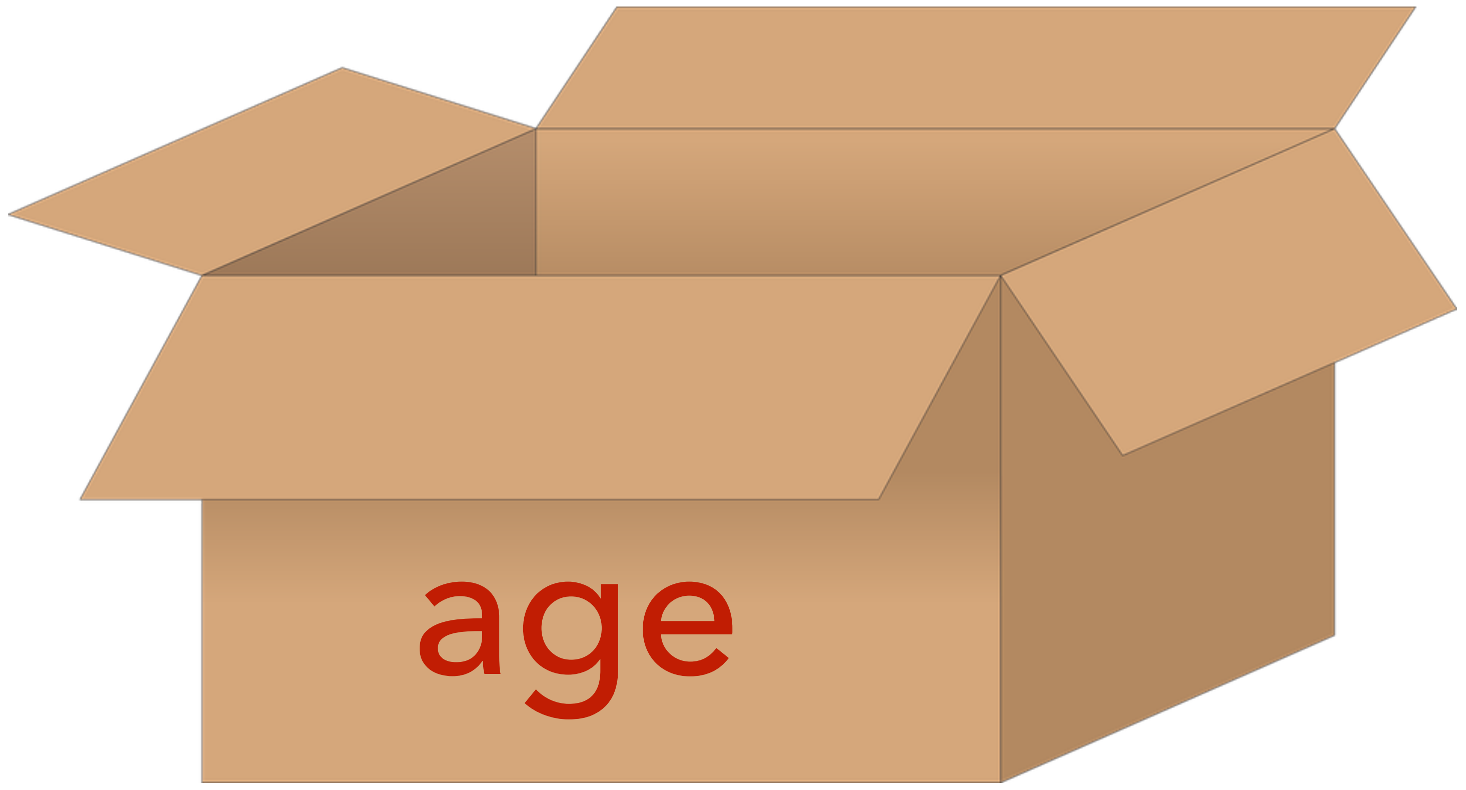


VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

New value assigned.

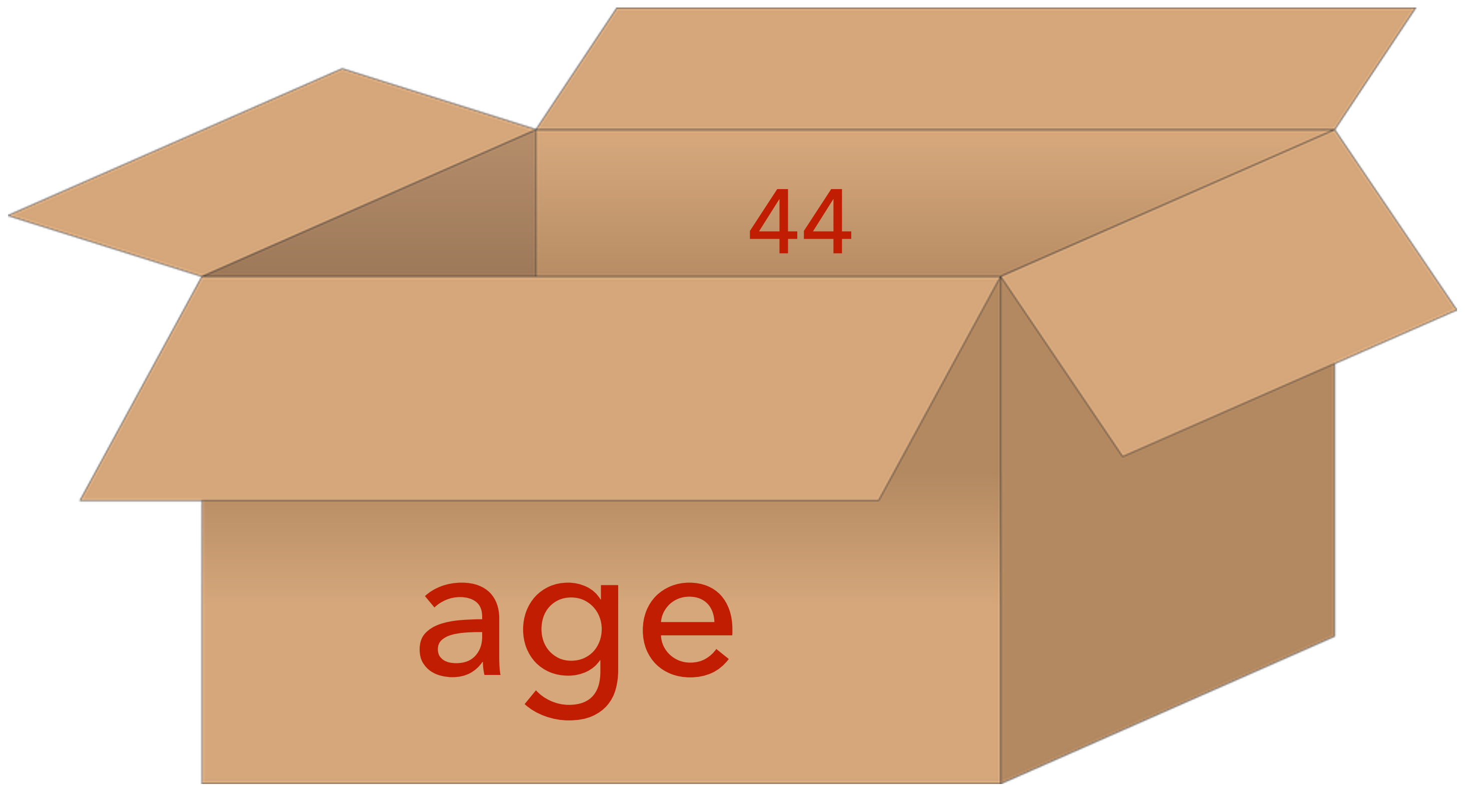


VARIABLES

You can think of a variable as a box that we can store data inside of.

```
var age = 52;

age = 44;
```

New value assigned.



JAVASCRIPT -DAY 1

DATA TYPES

DATA TYPES

DATA TYPES

Boolean

DATA TYPES

Boolean

Null

DATA TYPES

Boolean

Null

Undefined

DATA TYPES

Boolean

Null

Undefined

Number

DATA TYPES

Boolean

Null

Undefined

Number

String

DATA TYPES

Boolean

Null

Undefined

Number

String

Object

DATA TYPES

Boolean

Null

Undefined

Number

String

Object

Array

DATA TYPES

Boolean

Null

Undefined

Number

String

Object

Array

Function

DATA TYPES

Boolean $\longrightarrow$

Null

Undefined

Number

String

Object

Array

Function

DATA TYPES

Boolean $\longrightarrow$ true/false

Null

Undefined

Number

String

Object

Array

Function

DATA TYPES

Boolean $\longrightarrow$ true/false

Null $\longrightarrow$

Undefined

Number

String

Object

Array

Function

DATA TYPES

Boolean	$\longrightarrow$	true/false
Null	$\longrightarrow$	null
Undefined		
Number		
String		
Object		
Array		
Function		

DATA TYPES

Boolean	→	true/false
Null	→	null
Undefined	→	
Number		
String		
Object		
Array		
Function		

DATA TYPES

Boolean	→	true/false
Null	→	null
Undefined	→	undefined
Number		
String		
Object		
Array		
Function		

DATA TYPES

Boolean	→	true/false
Null	→	null
Undefined	→	undefined
Number	→	
String		
Object		
Array		
Function		

DATA TYPES

Boolean	$\longrightarrow$	true/false
Null	$\longrightarrow$	null
Undefined	$\longrightarrow$	undefined
Number	$\longrightarrow$	9
String		
Object		
Array		
Function		

DATA TYPES

Boolean	→	true/false
Null	→	null
Undefined	→	undefined
Number	→	9
String	→	
Object		
Array		
Function		

DATA TYPES

Boolean	$\longrightarrow$ true/false
Null	$\longrightarrow$ null
Undefined	$\longrightarrow$ undefined
Number	$\longrightarrow$ 9
String	$\longrightarrow$ 'LOTR'
Object	
Array	
Function	

DATA TYPES

Boolean	$\longrightarrow$ true/false
Null	$\longrightarrow$ null
Undefined	$\longrightarrow$ undefined
Number	$\longrightarrow$ 9
String	$\longrightarrow$ ‘LOTR’
Object	$\longrightarrow$
Array	
Function	

DATA TYPES

Boolean	→ true/false
Null	→ null
Undefined	→ undefined
Number	→ 9
String	→ ‘LOTR’
Object	→ {id: 2}
Array	
Function	

DATA TYPES

Boolean	true/false
Null	null
Undefined	undefined
Number	9
String	'LOTR'
Object	{id: 2}
Array	
Function	

DATA TYPES

Boolean	true/false
Null	null
Undefined	undefined
Number	9
String	'LOTR'
Object	{id: 2}
Array	[1, 2, 3]
Function	

DATA TYPES

Boolean	true/false
Null	null
Undefined	undefined
Number	9
String	‘LOTR’
Object	{id: 2}
Array	[1, 2, 3]
Function	

DATA TYPES

Boolean	true/false
Null	null
Undefined	undefined
Number	9
String	'LOTR'
Object	{id: 2}
Array	[1, 2, 3]
Function	function() { }

ARRAY

ARRAY

[1, 2, 3]

ARRAY

Square brackets

[1, 2, 3]

ARRAY

$$\overbrace{[\ 1, 2, 3\ ]}^{\text{Square brackets}}$$

ARRAY

$$\overbrace{[\ 1, 2, 3\ ]}^{\text{Square brackets}}$$

Values, separated by commas.

ARRAY

Square brackets

```
[ 1, 2, 3 ]
```

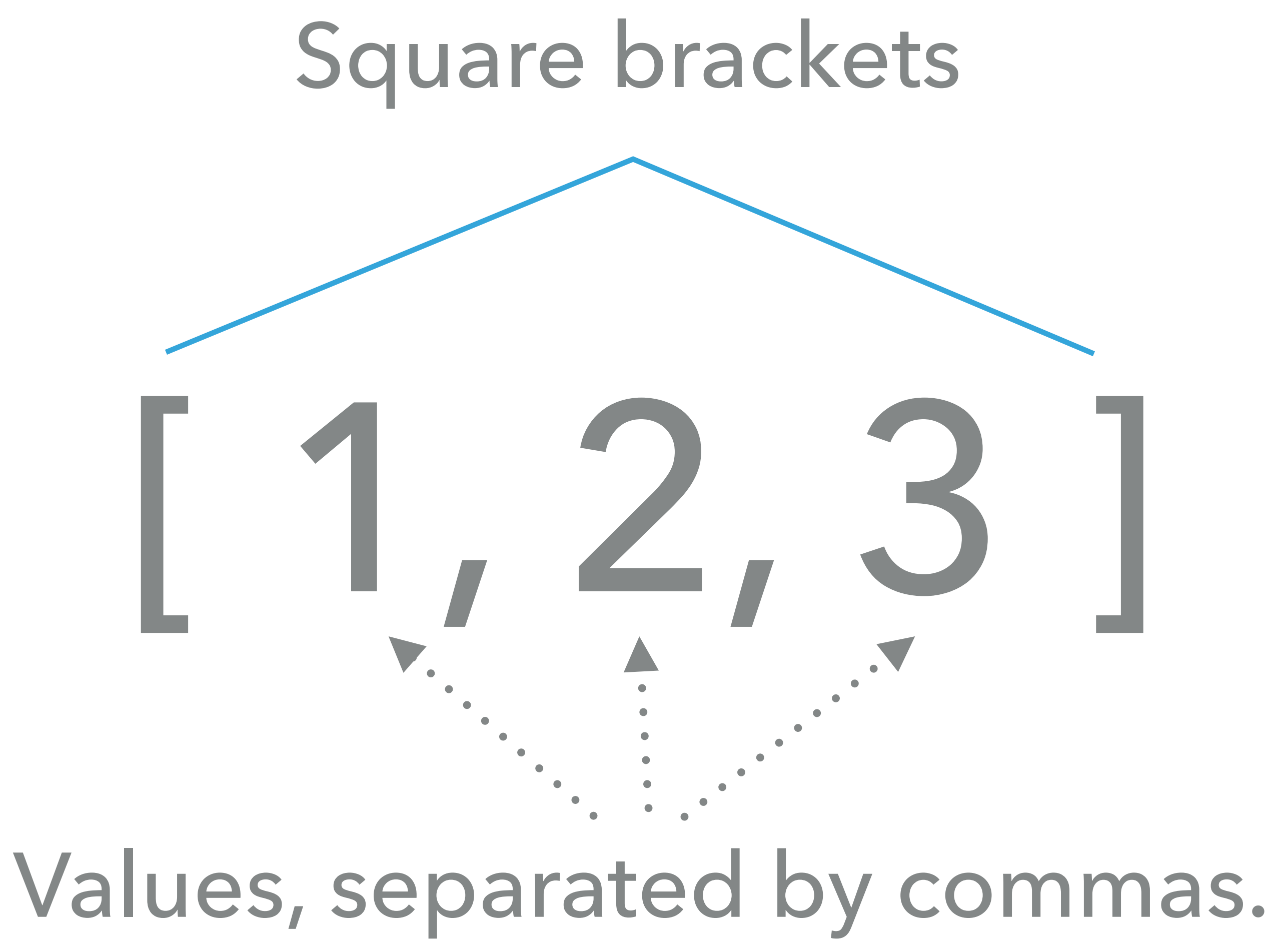
Values, separated by commas.

ARRAY

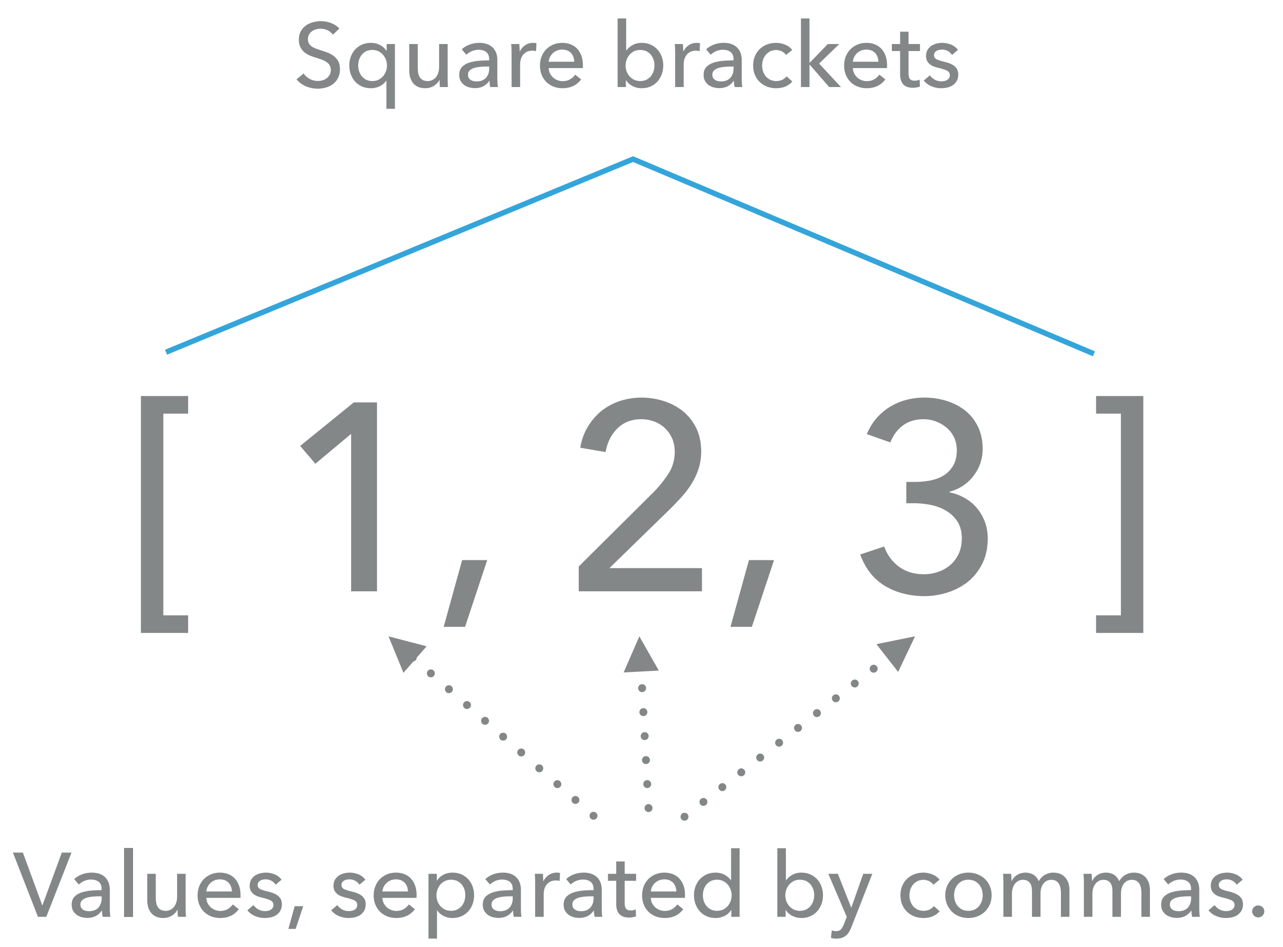
$$\overbrace{[\ 1,\ 2,\ 3\ ]}^{\text{Square brackets}}$$

Values, separated by commas.

ARRAY



ARRAY



Values in arrays can have multiple data types.

ARRAY

Square brackets

[1, 2, 3]

Values, separated by commas.

Values in arrays can have multiple data types.

[1, ‘2’, false]

ARRAY

Square brackets

[1, 2, 3]

Values, separated by commas.

Values in arrays can have multiple data types.

[1, ‘2’, false]   [true, ‘cats’, null]

OBJECTS

OBJECTS

```
var car = {
    make: ‘Toyota’,
    model: ‘Corolla’,
    color: ‘red’,
    year: 2006
}
```

OBJECTS

Opening curly bracket

```
var car = {
    make: ‘Toyota’,
    model: ‘Corolla’,
    color: ‘red’,
    year: 2006
}
```

OBJECTS

Opening curly bracket

```
var car = {
      make: 'Toyota',
      model: 'Corolla',
      color: 'red',
      year: 2006
}
```

OBJECTS

Opening curly bracket

```
var car = {
      make: 'Toyota',
      model: 'Corolla',
      color: 'red',
      year: 2006
}
```

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
      make: ‘Toyota’,
      model: ‘Corolla’,
      color: ‘red’,
      year: 2006
}
```

Closing curly bracket

OBJECTS

Opening curly bracket

property

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

property

key

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

property

key

name

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

property

key

name

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

Value

property

key

name

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: ‘Toyota’,
    model: ‘Corolla’,
    color: ‘red’,
    year: 2006
}
```

Value

property

key

name

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: ‘Toyota’,
    model: ‘Corolla’,
    color: ‘red’,
    year: 2006
}
```

Value

property

key

name

Key/value pairs

separated by commas.

Closing curly bracket

OBJECTS

Opening curly bracket

```
var car = {
    make: 'Toyota',
    model: 'Corolla',
    color: 'red',
    year: 2006
}
```

Value

property

key

name

Key/value pairs

separated by commas.

Closing curly bracket

HTTPS://REPL.IT/JZ5C/1

IF STATEMENT

IF STATEMENTS

IF STATEMENTS

```
var five = 5;

if ( five === 5 ) {

  console.log(‘Five is awesome cause it equals 5’);

} else {

    console.log(‘five does not equal 5’)

}
```

IF STATEMENTS

condition to test

```
var five = 5;

if ( five === 5 ) {

  console.log(‘Five is awesome cause it equals 5’);

} else {

    console.log(‘five does not equal 5’)

}
```

IF STATEMENTS

condition to test

```
var five = 5;

if ( five === 5 ) {

  console.log('Five is awesome cause it equals 5');

} else {

    console.log('five does not equal 5')

}
```

IF STATEMENTS

condition to test

block of code that runs if condition evaluates to true

```
var five = 5;

if ( five === 5 ) {

  console.log('Five is awesome cause it equals 5');

} else {

    console.log('five does not equal 5')

}
```

IF STATEMENTS

condition to test

block of code that runs if condition evaluates to true

```
var five = 5;

if ( five === 5 ) {

  console.log(‘Five is awesome cause it equals 5’);

} else {

    console.log(‘five does not equal 5’)

}
```

IF STATEMENTS

condition to test

```
var five = 5;

if ( five === 5 ) {

  console.log('Five is awesome cause it equals 5');

} else {

    console.log('five does not equal 5')

}
```

block of code that runs if condition evaluates to true

block of code that runs if condition evaluates to false

IF STATEMENTS

condition to test

block of code that runs if condition evaluates to true

block of code that runs if condition evaluates to false

```
var five = 5;

if ( five === 5 ) {

  console.log('Five is awesome cause it equals 5');

} else {

    console.log('five does not equal 5')

}
```

JAVASCRIPT - DAY 1

FUNCTIONS

FUNCTIONS

FUNCTIONS

function <u>expression</u>:

FUNCTIONS

function expression:

FUNCTIONS

function expression:

FUNCTIONS

function expression:

```
var sayName = function() {
```

FUNCTIONS

function <u>expression</u>:

```
var sayName = function() {
    alert('Fred');
```

FUNCTIONS

function expression:

```
var sayName = function() {
    alert('Fred');
}
```

FUNCTIONS

function expression:	function declaration:

`var sayName = function() {`
`    alert('Fred');`
`}`

FUNCTIONS

function expression:

function declaration:

```
var sayName = function() {
    alert('Fred');
}
```

FUNCTIONS

function expression:

function declaration:

```
var sayName = function() {
    alert('Fred');
}
```

FUNCTIONS

function expression:

function declaration:

```
var sayName = function() {
    alert('Fred');
}
```

```
function sayName() {
```

FUNCTIONS

function expression:	function declaration:

```
var sayName = function() {
    alert('Fred');
}
```

```
function sayName() {
    alert('Fred');
```

FUNCTIONS

function <u>expression</u>:	function <u>declaration</u>:

`var sayName = function() {`
`    alert('Fred');`
`}`
	`function sayName() {`
	`    alert('Fred');`
	`}`

FUNCTIONS

FUNCTIONS

```
function sayName(
```

FUNCTIONS

```
function sayName( ) {
```

FUNCTIONS

```
function sayName( ) {


}
```

FUNCTIONS

```
function sayName( ) {



}

sayName(
```

FUNCTIONS

```
function sayName( ) {


}

sayName(  )
```

FUNCTIONS

```
function sayName( ) {


}
```

call

```
sayName(  )
```

FUNCTIONS

```
function sayName( ) {


}
```

call

invoke

```
sayName(  )
```

FUNCTIONS

```
function sayName( ) {


}
```

call

invoke

run

```
sayName(  )
```

FUNCTIONS

```
function sayName( ) {


}
```

call

invoke

run

```
sayName(  )
```

FUNCTIONS

```
function sayName( ) {


}
```

call

invoke

run

```
sayName(                )
```

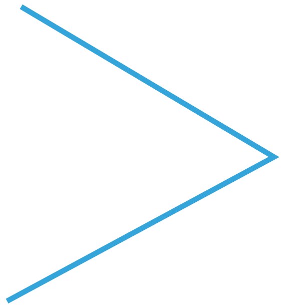
FUNCTIONS

```
function sayName( ) {


}
```

call

invoke

run



sayName(arguments)

FUNCTIONS

```
function sayName(                ) {



}
```

call

invoke

run

```
sayName(  arguments   )
```

FUNCTIONS

```
function sayName( parameters ) {


}
```

call
invoke
run

```
sayName( arguments )
```

FUNCTIONS

```
function sayName( parameters ) {

}
```

call

invoke

run

```
sayName( arguments )
```

[HTTPS://REPL.IT/KAPA/3](https://repl.it/KAPA/3)

JAVASCRIPT - DAY 1

RETURNING FROM FUNCTIONS

RETURNING FROM FUNCTIONS

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =
```

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =  add();
```

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =  add();
```

function stops executing when it reaches return statement

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =  add();
```

function stops executing when it reaches return statement $\longrightarrow$ `return num + num;`

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =  add();
```

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

function stops executing when it reaches return statement $\longrightarrow$

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}

var addedNums =  add();
```

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

function stops executing when it reaches return statement $\longrightarrow$

4

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =  add();
```

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

function stops executing when it reaches return statement $\longrightarrow$ `return num + num;` $\longleftarrow$ 4

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

```
function add() {
  var num = 2;
  return num + num;
}

var addedNums =  add();
```

function stops executing when it reaches return statement $\longrightarrow$

4

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =
```

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

4

function stops executing when it reaches return statement $\longrightarrow$

RETURNING FROM FUNCTIONS

When we invoke a function, we can have it return a value.

We do this by using a return statement.

```
function add() {
  var num = 2;
  return num + num;
}
var addedNums =    4
```

the function will compute the code to the right of the return statement, if necessary, then return the value to where the function was called

function stops executing when it reaches return statement $\longrightarrow$

JAVASCRIPT - DAY 1

SCOPE

SCOPE

The context in which values and expressions are "visible," or can be referenced.

The global scope is “visible” to all of your code.

Scopes can also be layered in a hierarchy, so that child scopes have access to parent scopes, but not vice versa.

https://developer.mozilla.org/en-US/docs/Glossary/Scope

SCOPE

SCOPE

Functions have their own scope.

SCOPE

SCOPE

```
var name1 = 'Lucy';
```

SCOPE

```
var name1 = 'Lucy';
```

SCOPE

```
var name1 = ‘Lucy’;


function sayName() {
```

SCOPE

```
var name1 = ‘Lucy’;



function sayName() {

console.log( name1);
```

SCOPE

```
var name1 = ‘Lucy’;


function sayName() {

console.log( name1);

var name2 = ‘Nancy’;
```

SCOPE

```
var name1 = ‘Lucy’;


function sayName() {
console.log( name1);
var name2 = ‘Nancy’;
}
```

SCOPE

```
var name1 = ‘Lucy’;


function sayName() {
console.log( name1);
var name2 = ‘Nancy’;
}
console.log( name2 )
```

SCOPE

```
var name1 = ‘Lucy’;


function sayName() {
console.log( name1);
var name2 = ‘Nancy’;
}
console.log( name2 )
```

Global variable. Can be seen by all code.

SCOPE

```
var name1 = 'Lucy';                 Global variable. Can be seen by all code.


function sayName() {
console.log( name1);                Can access name1 variable.
var name2 = 'Nancy';
}

console.log( name2 )
```

SCOPE

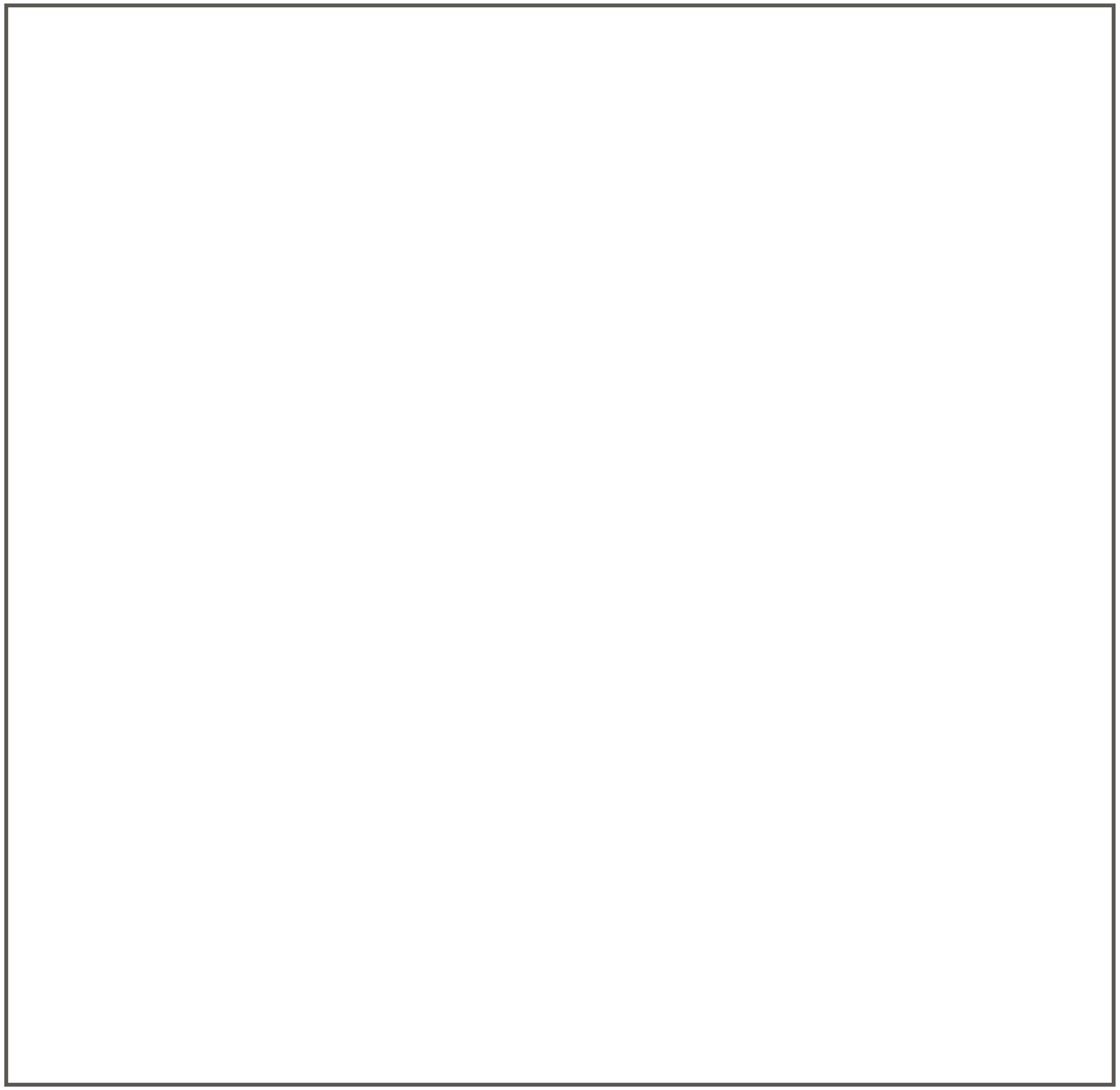
```
var name1 = 'Lucy';                    Global variable. Can be seen by all code.


function sayName() {

console.log( name1);                   Can access name1 variable.

var name2 = 'Nancy';

}

console.log( name2 )                   undefined
```

SCOPE

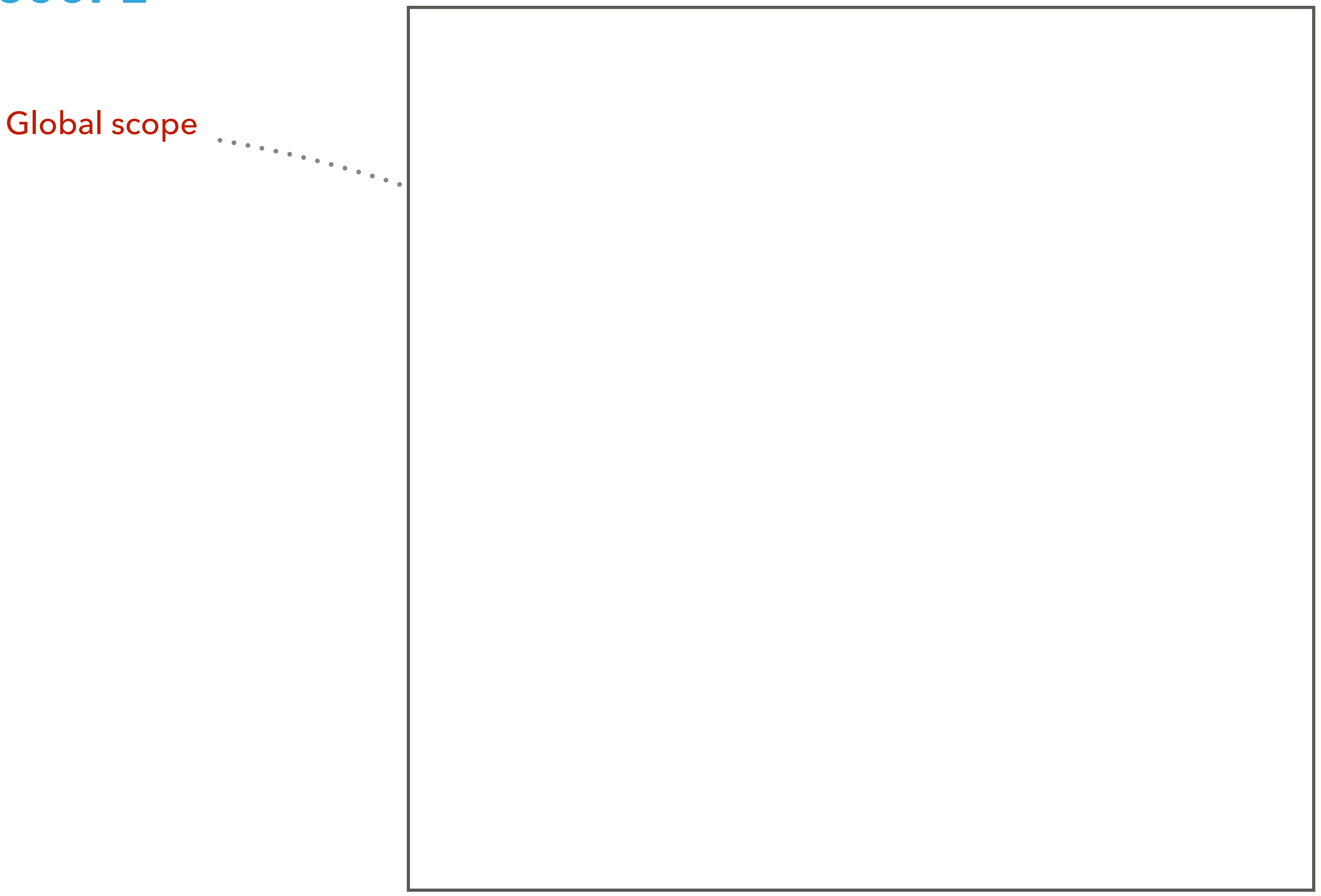
SCOPE

SCOPE

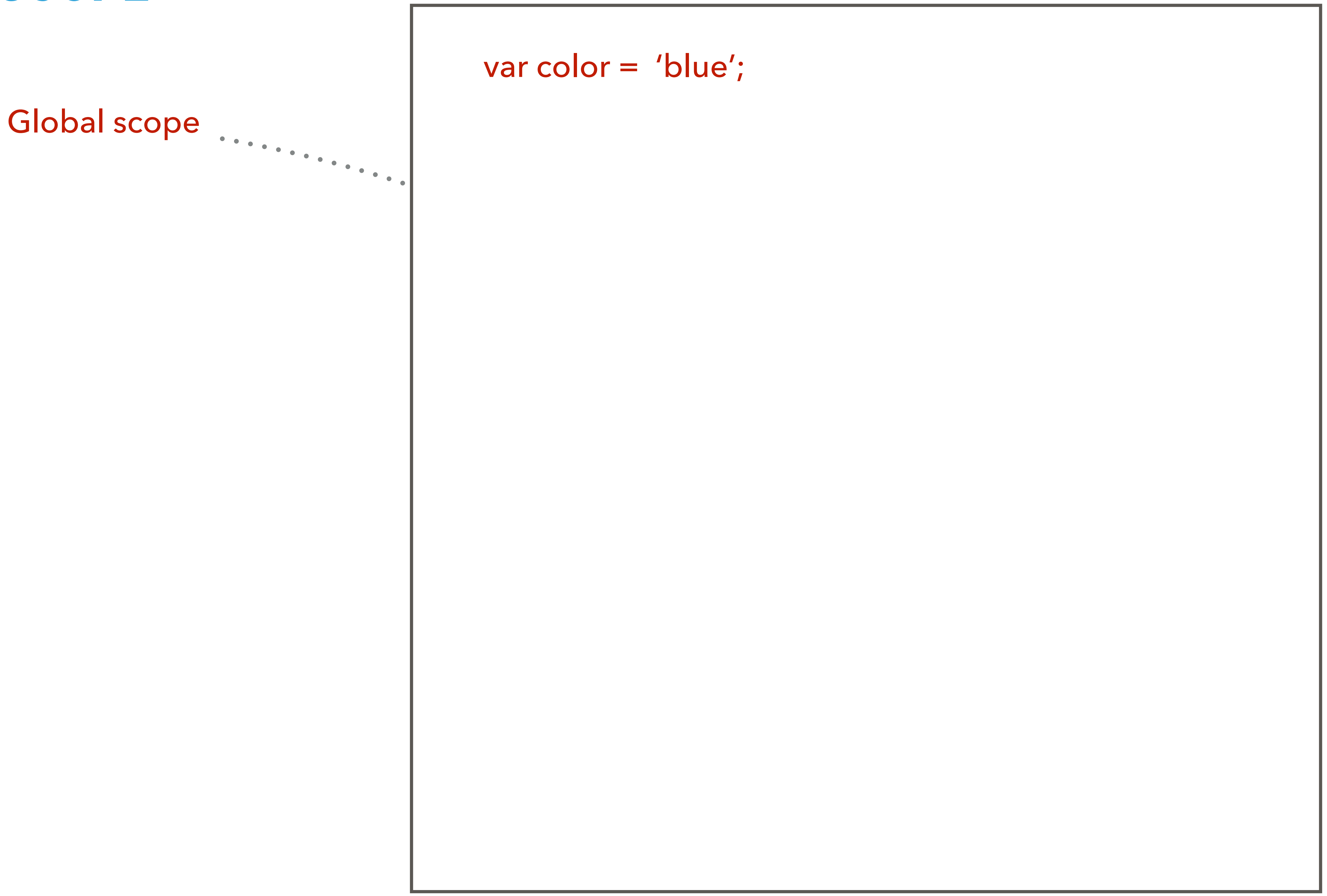
Global scope



SCOPE

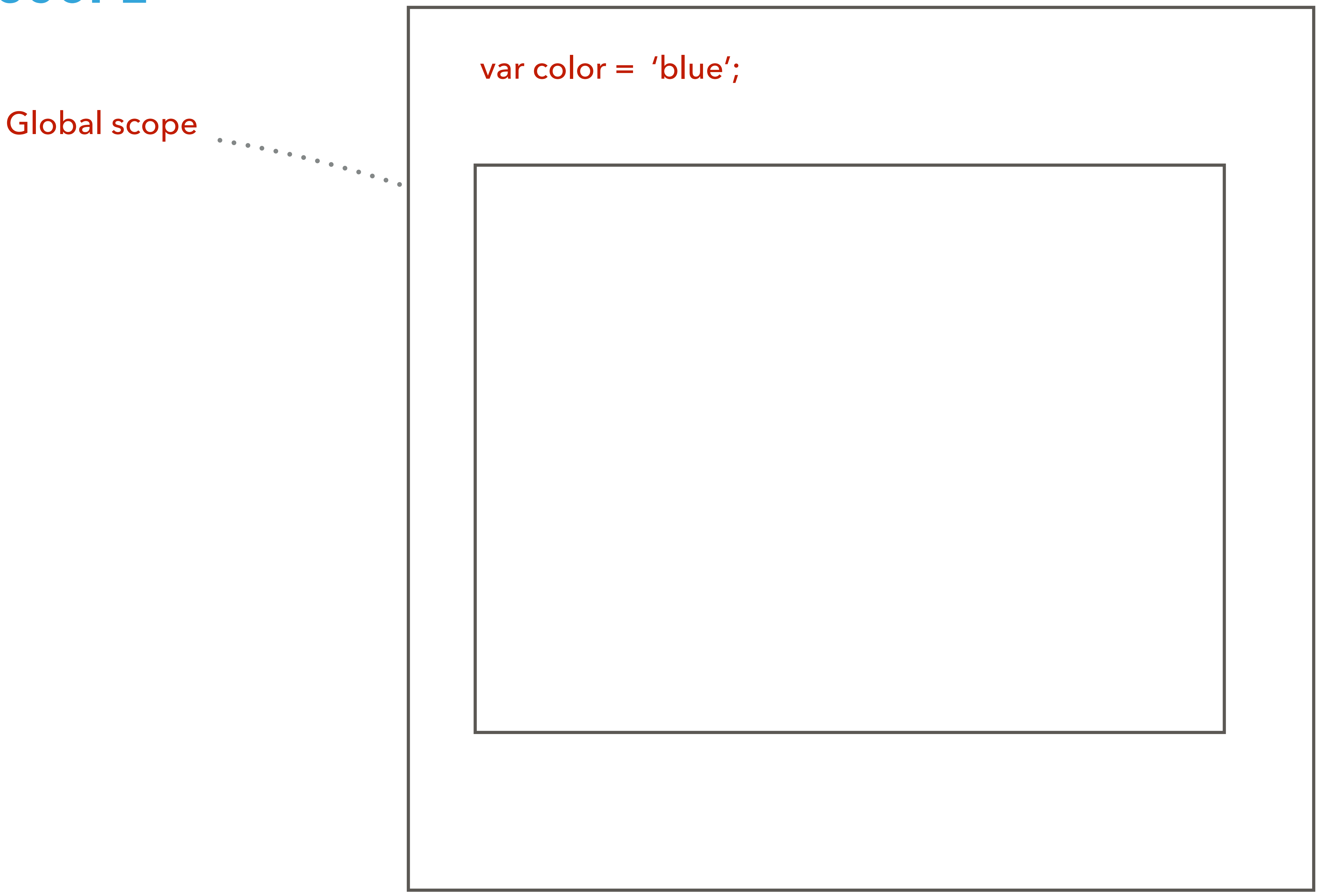


SCOPE

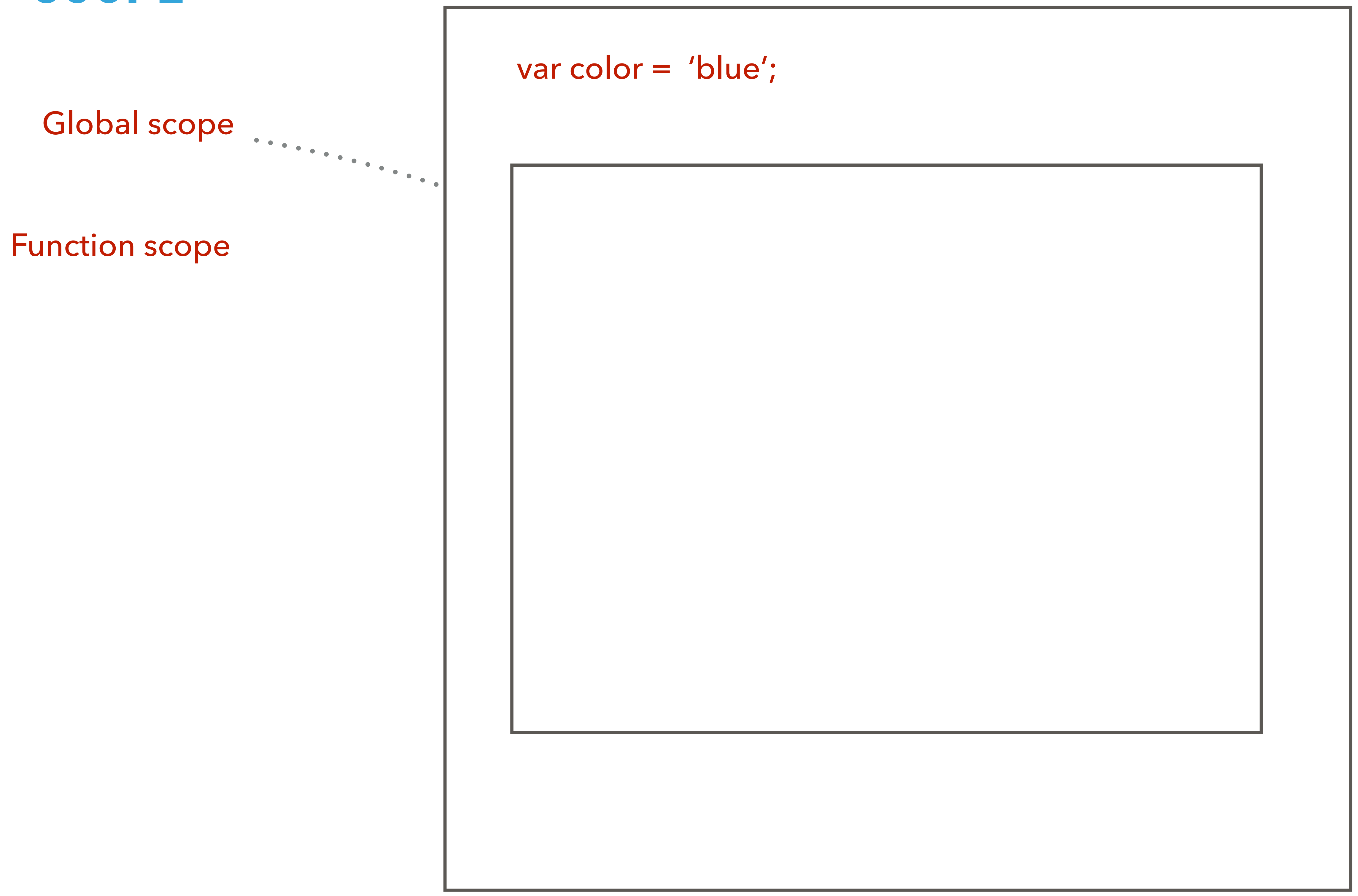


```
var color =  'blue';
```

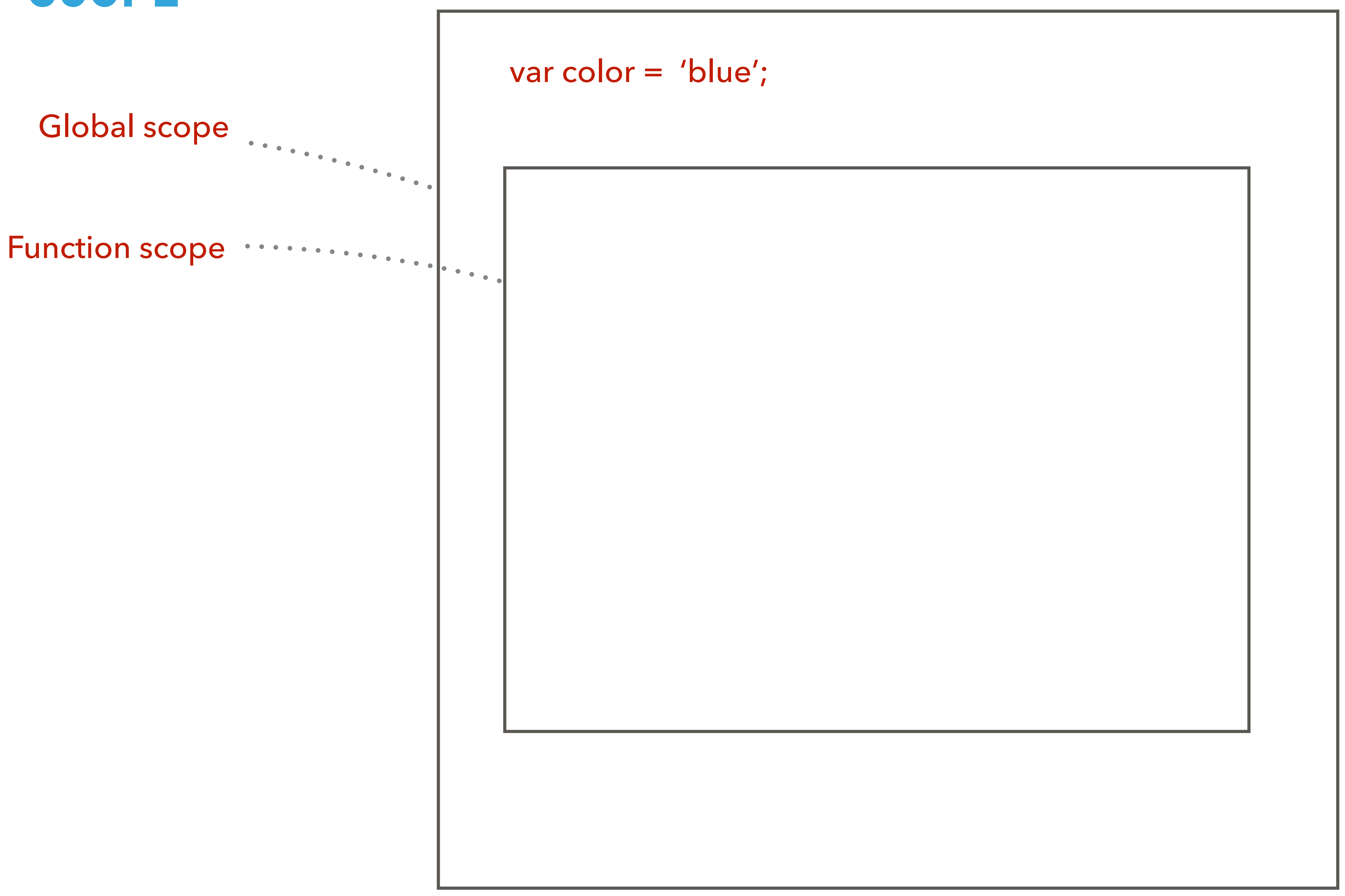
SCOPE



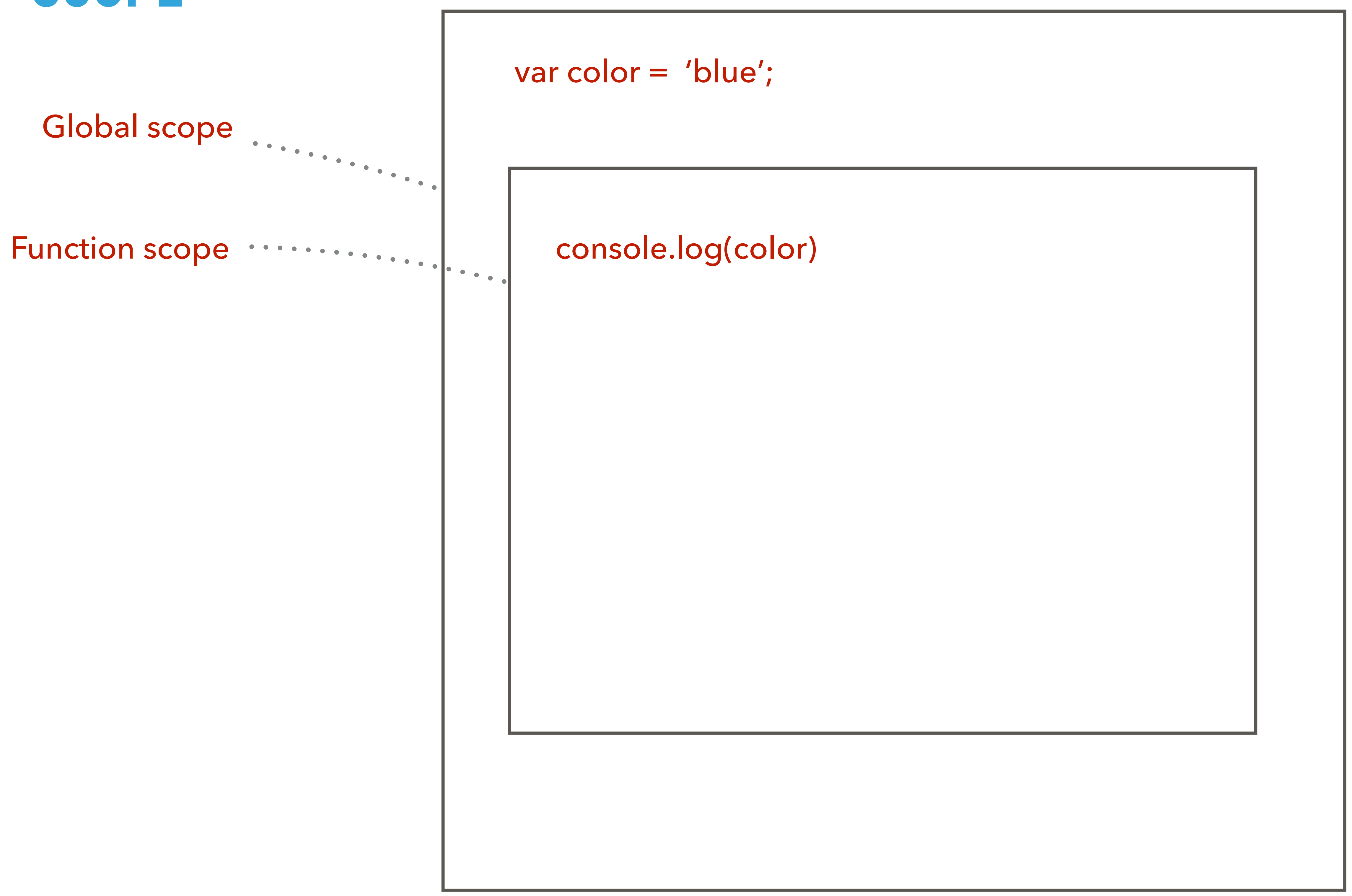
SCOPE



SCOPE



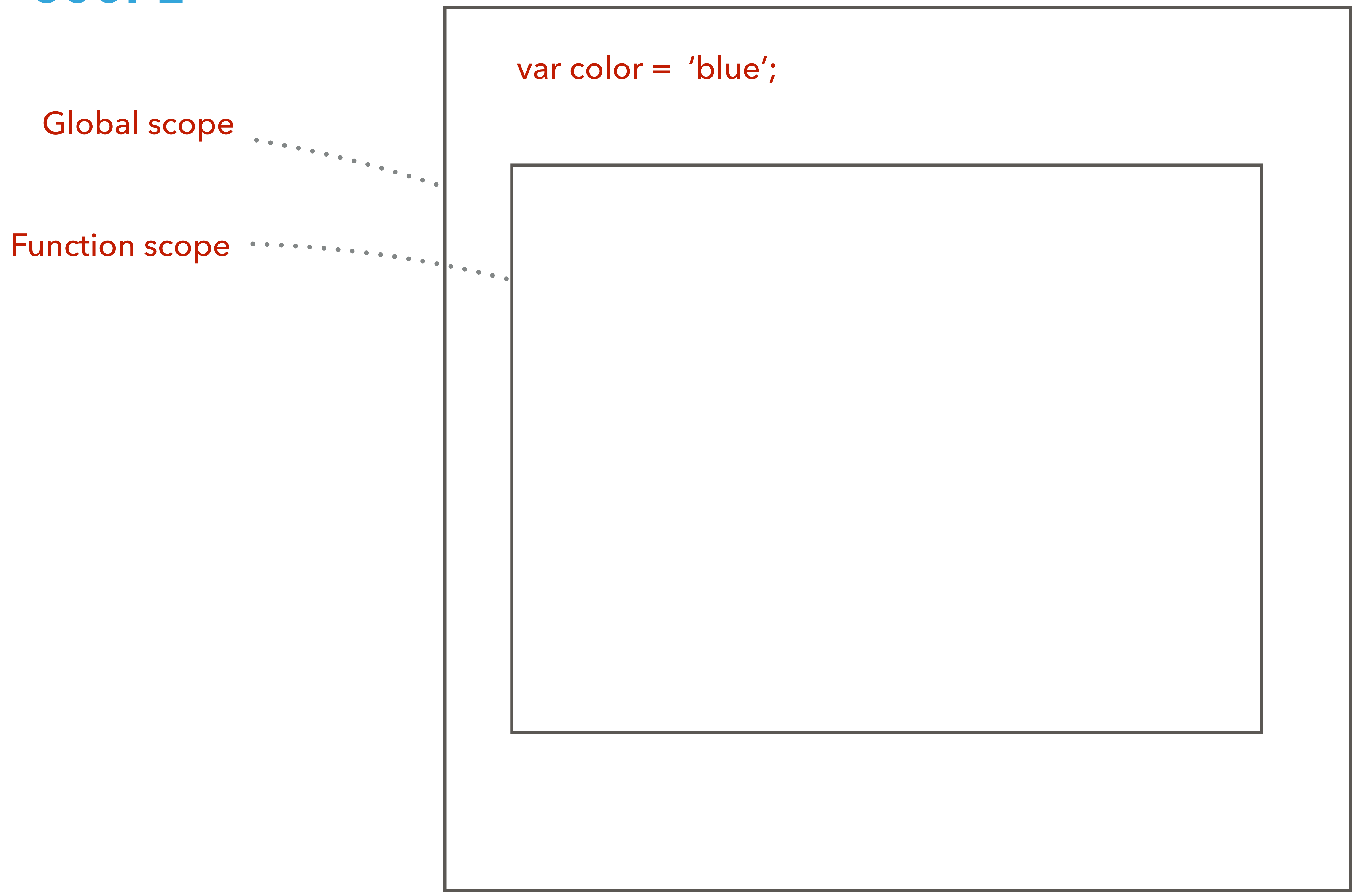
SCOPE



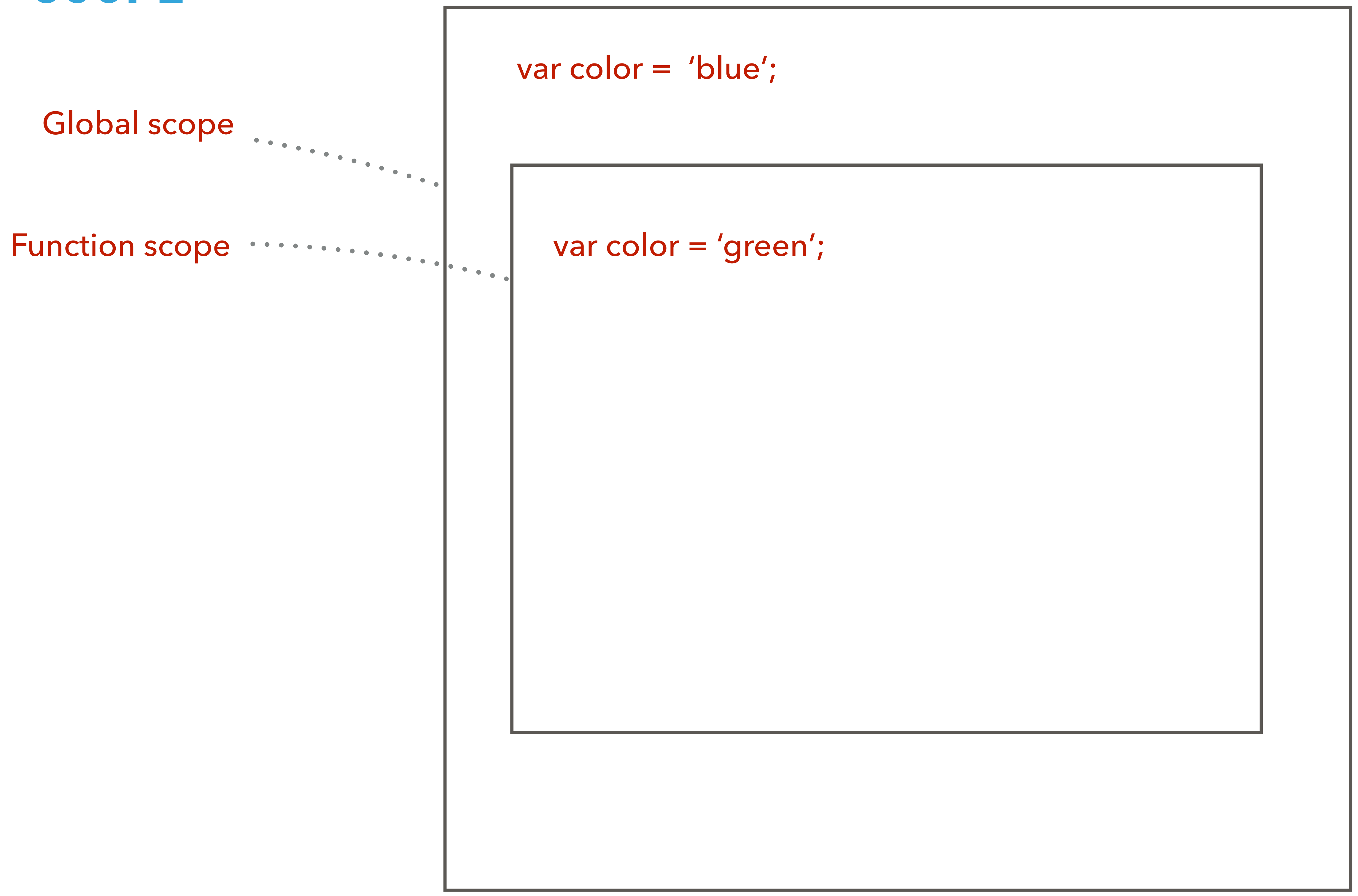
```
var color =  'blue';
```

```
console.log(color)
```

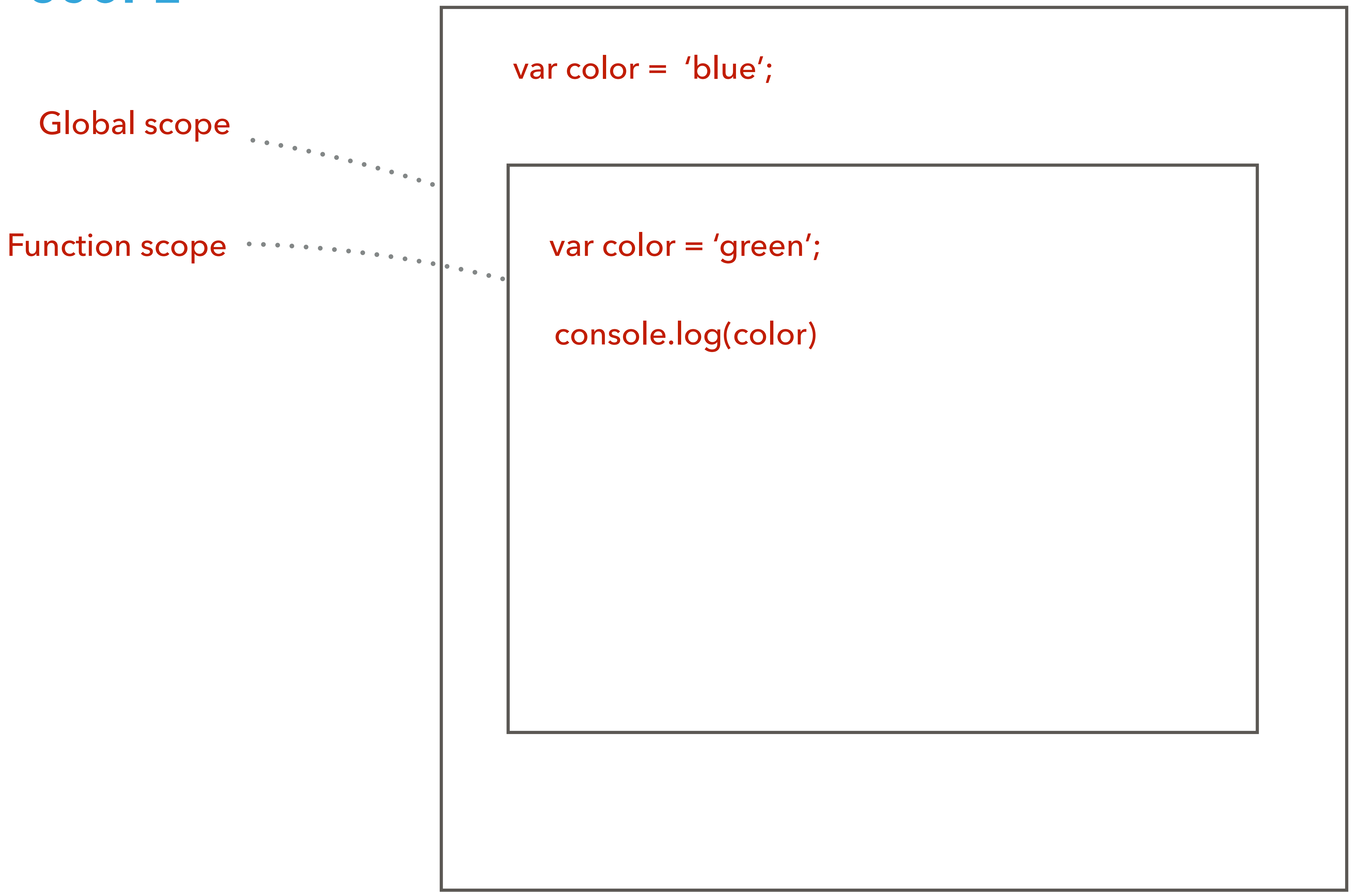
SCOPE



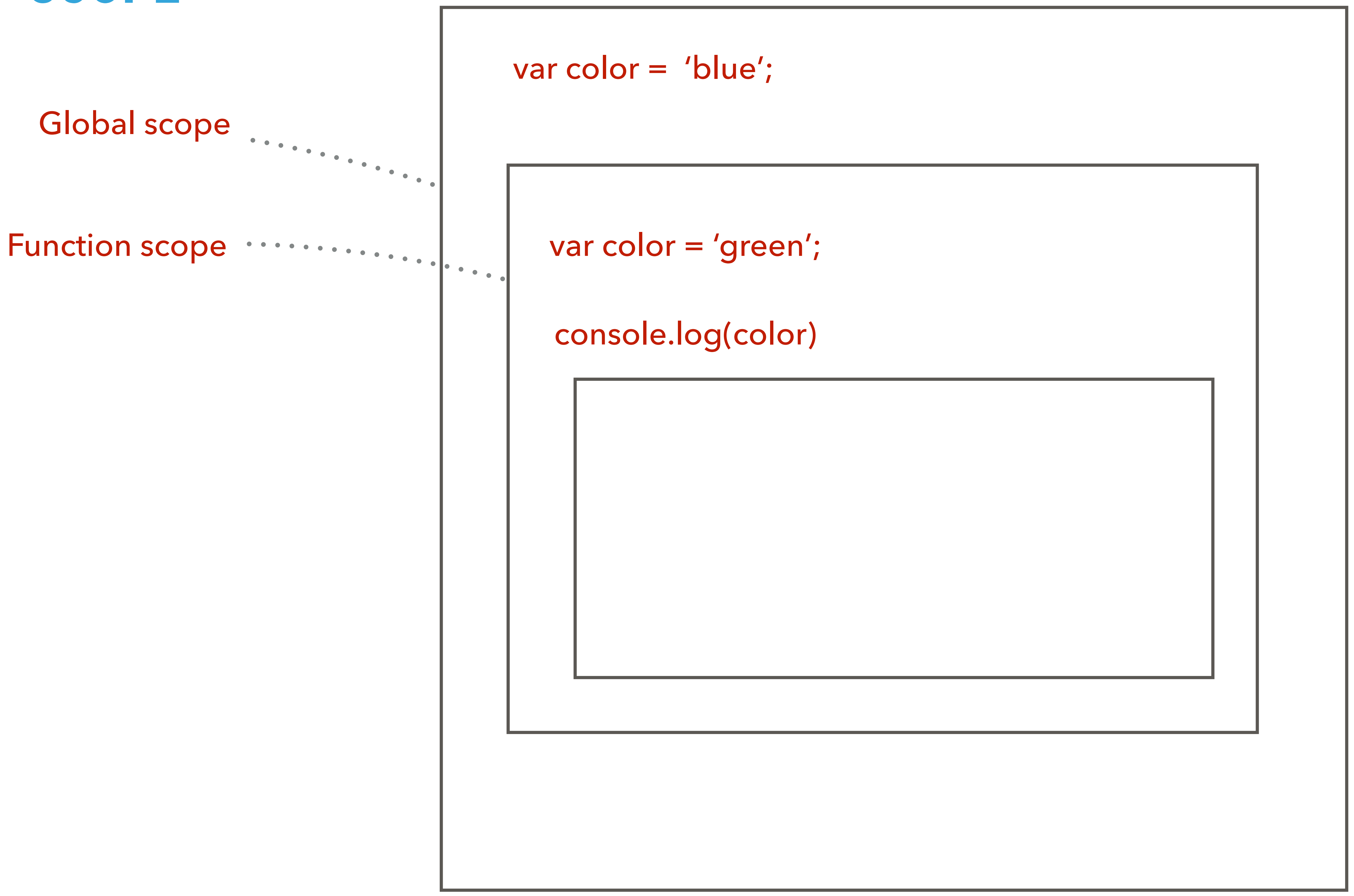
SCOPE



SCOPE

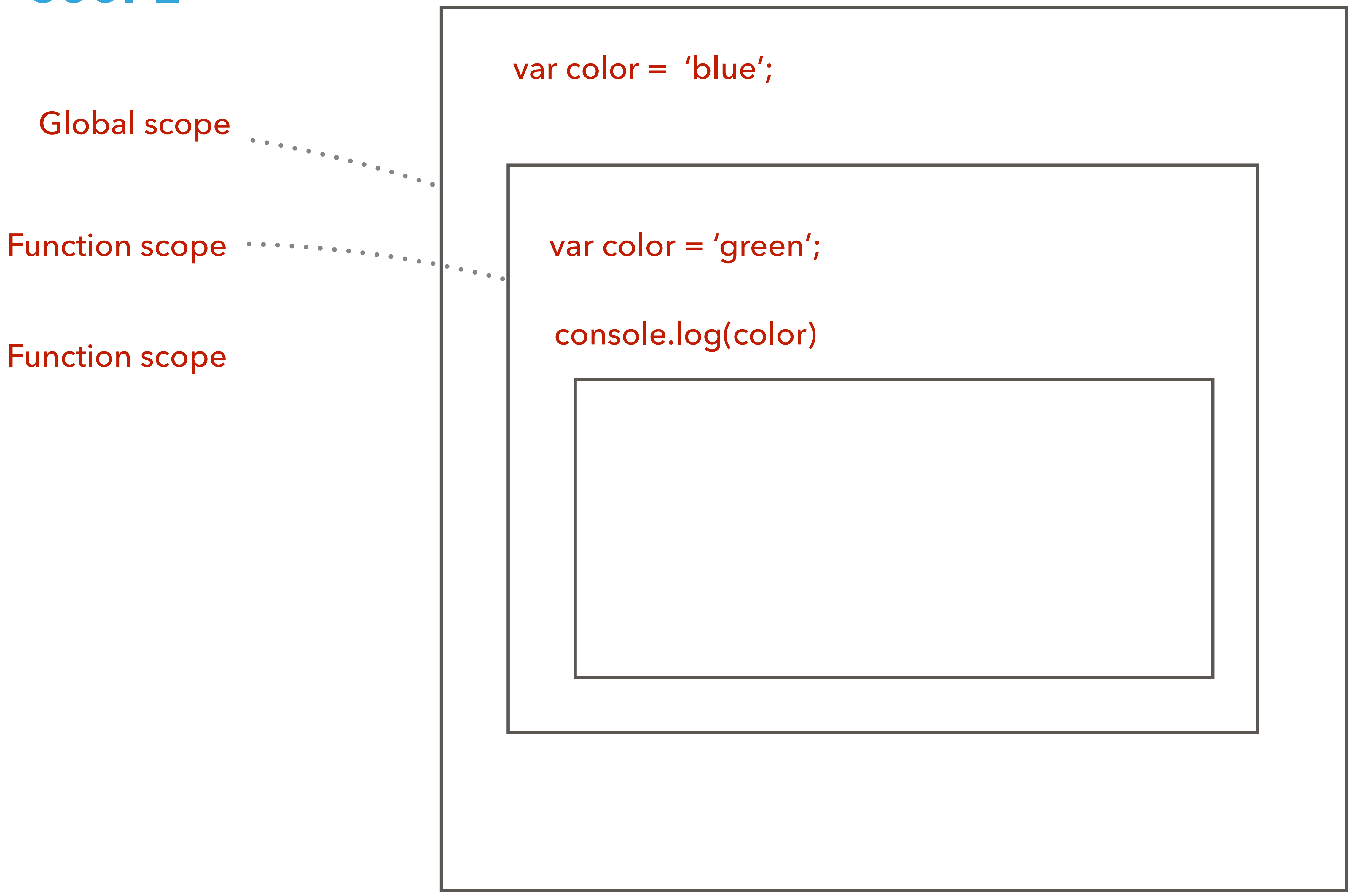

var color = 'blue';
Global scope
Function scope
var color = 'green';
console.log(color)

SCOPE



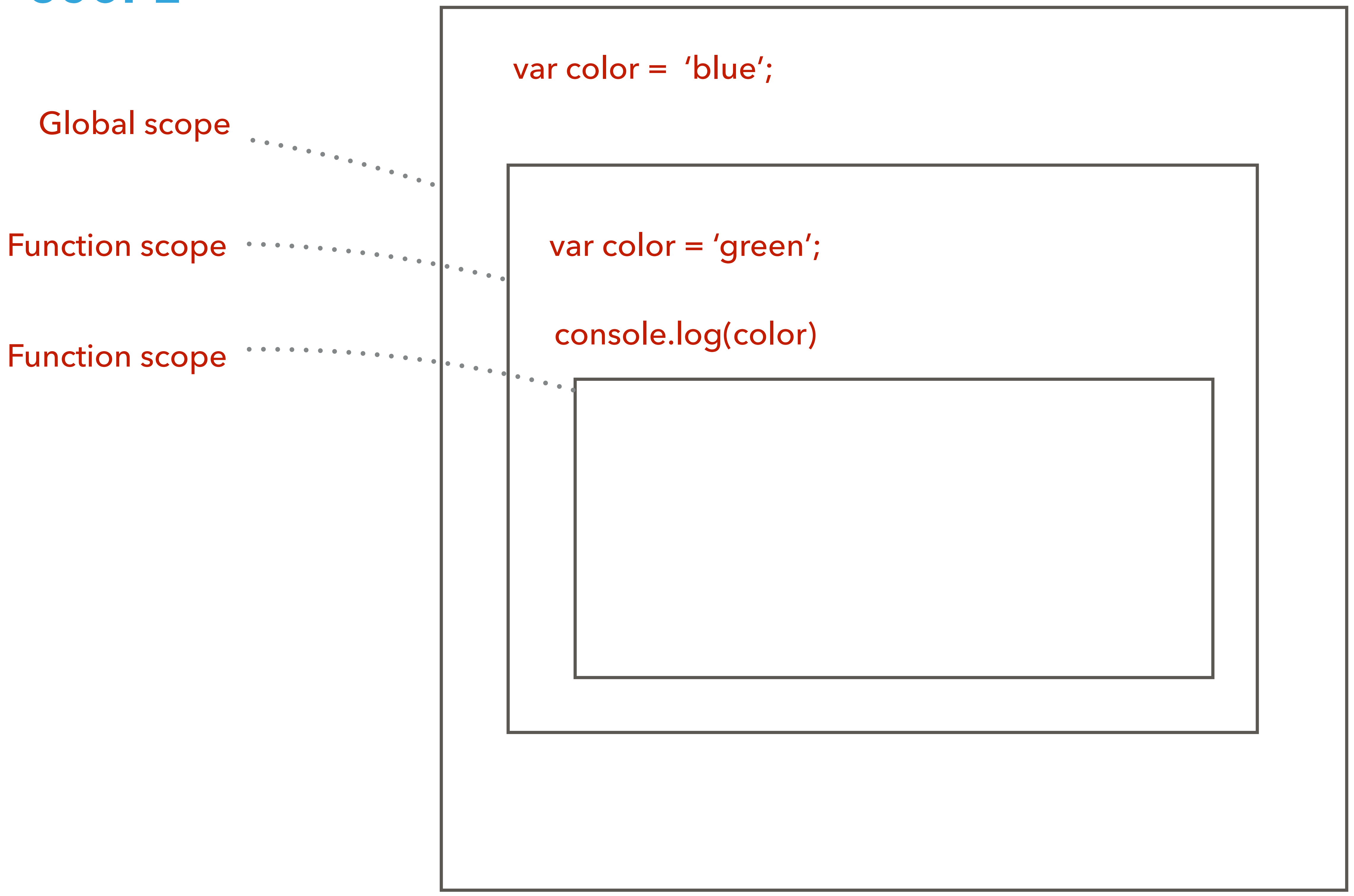
```
var color =  'blue';
var color = 'green';
console.log(color)
```

SCOPE

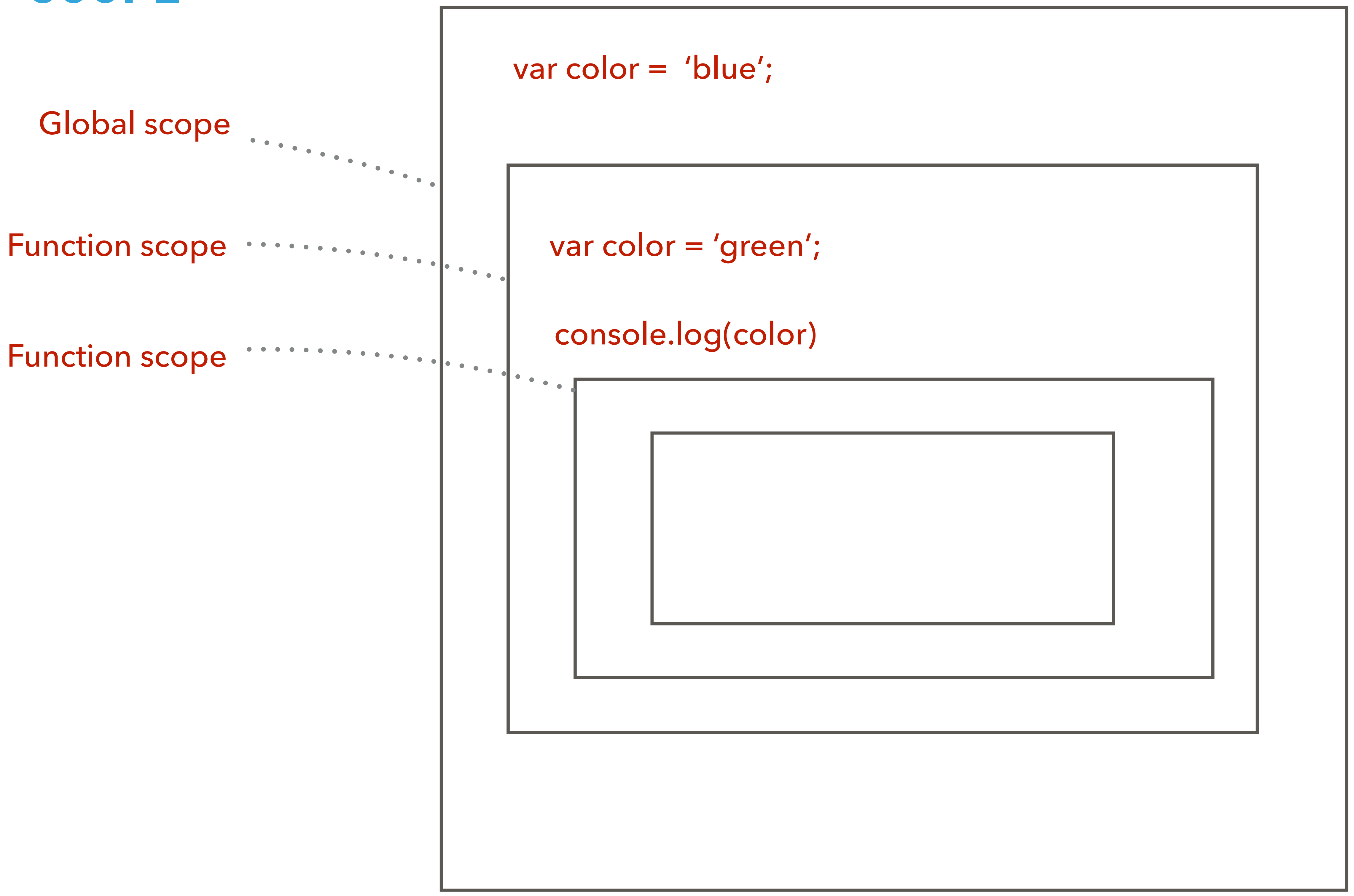


```
var color =  'blue';
var color = 'green';
console.log(color)
```

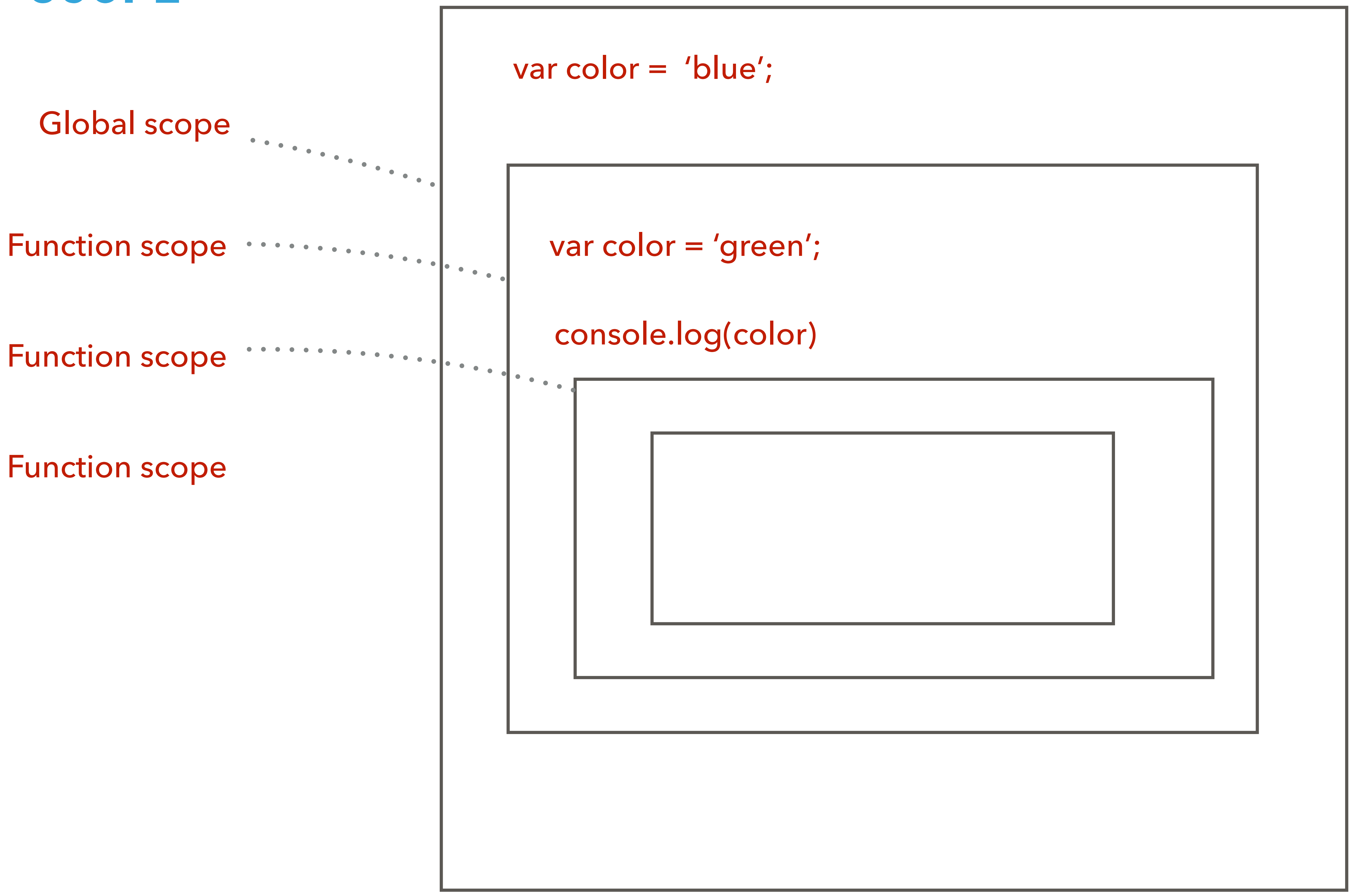
SCOPE


Global scope
Function scope
Function scope
var color = 'blue';
var color = 'green';
console.log(color)

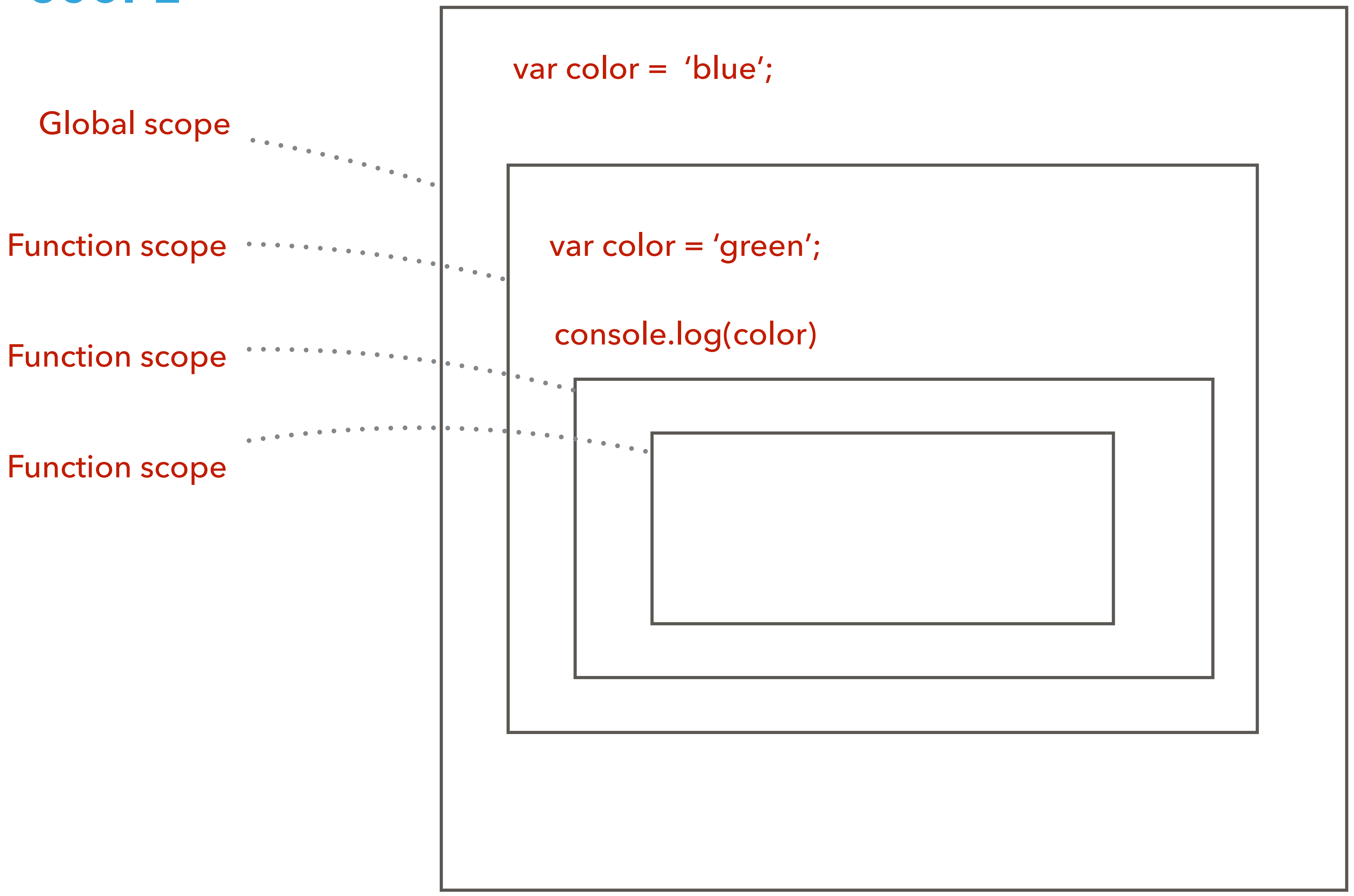
SCOPE


Global scope
Function scope
Function scope
var color = 'blue';
var color = 'green';
console.log(color)

SCOPE


Global scope
Function scope
Function scope
Function scope
var color = 'blue';
var color = 'green';
console.log(color)

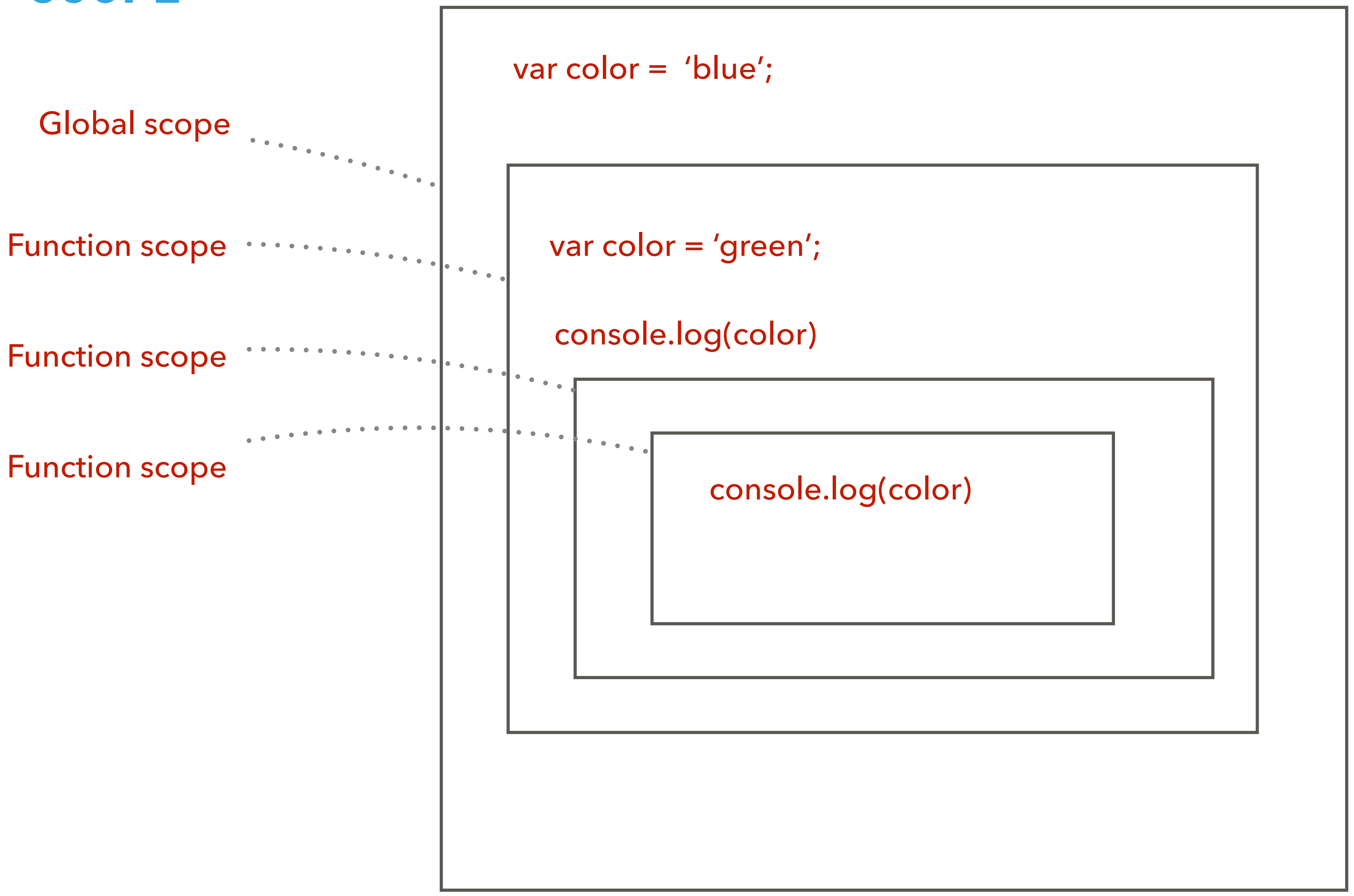
SCOPE



```
var color =  'blue';
```

```
var color = 'green';

console.log(color)
```

SCOPE



```
var color =  'blue';
var color = 'green';
console.log(color)
console.log(color)
```

HTTPS://REPL.IT/KAP7/2

LET

LET

- let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the var keyword, which defines a variable globally, or locally to an entire function regardless of block scope. *

* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

LET

LET

```
function varTest() {
  var myName = ‘Gary’;
}
console.log(myName)
```

LET

```
function varTest() {
  var myName = ‘Gary’;
}
console.log(myName)
```

ERROR

LET

```
function varTest() {
  var myName = 'Gary';
}
console.log(myName)
```

ERROR

```
function letTest() {
  let myName = 'Gary';
}
console.log(myName)
```

LET

```
function varTest() {
  var myName = 'Gary';
}
console.log(myName)
```

ERROR

```
function letTest() {
  let myName = 'Gary';
}
console.log(myName)
```

ERROR

LET

LET

```
if (3 === 3) {
  var threeEquals3 = true;
}
console.log(threeEquals3)
```

LET

```
if (3 === 3) {
  var threeEquals3 = true;
}
console.log(threeEquals3)
```

TRUE

LET

```
if (3 === 3) {
  var threeEquals3 = true;
}
console.log(threeEquals3)
```

TRUE

```
if (3 === 3) {
  let threeEquals3 = true;
}
console.log(threeEquals3)
```

LET

```
if (3 === 3) {
  var threeEquals3 = true;
}
console.log(threeEquals3)
```

TRUE

```
if (3 === 3) {
  let threeEquals3 = true;
}
console.log(threeEquals3)
```

ERROR

LET

LET

```
for (var i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

LET

```
for (var i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

```
5
```

LET

```
for (var i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

```
5
```

```
for (let i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

LET

```
for (var i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

5

```
for (let i = 0; i < 4; i++) {
  // code
}
console.log( i );
```

ERROR

MINI-PROJECT