

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.

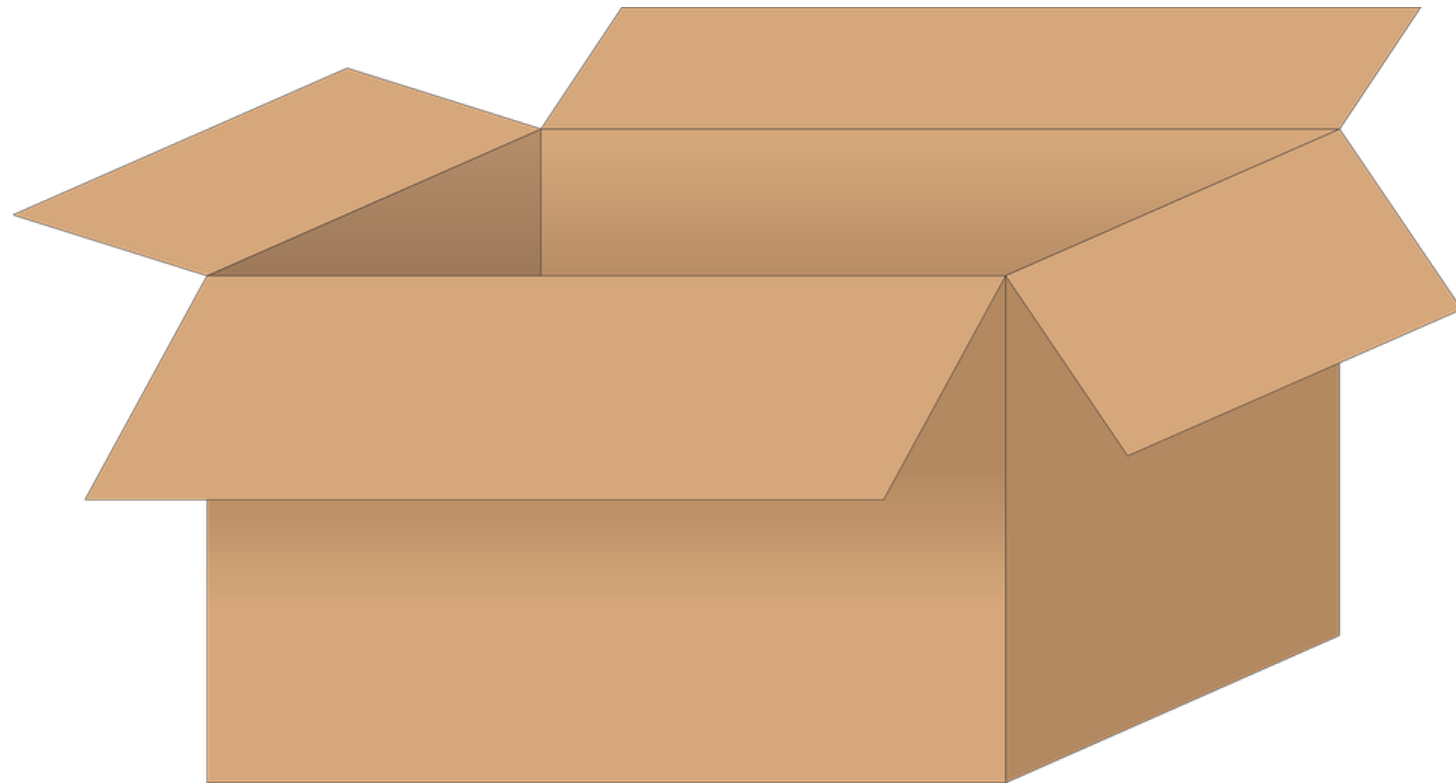
```
var age = 52;
```



## VARIABLES

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

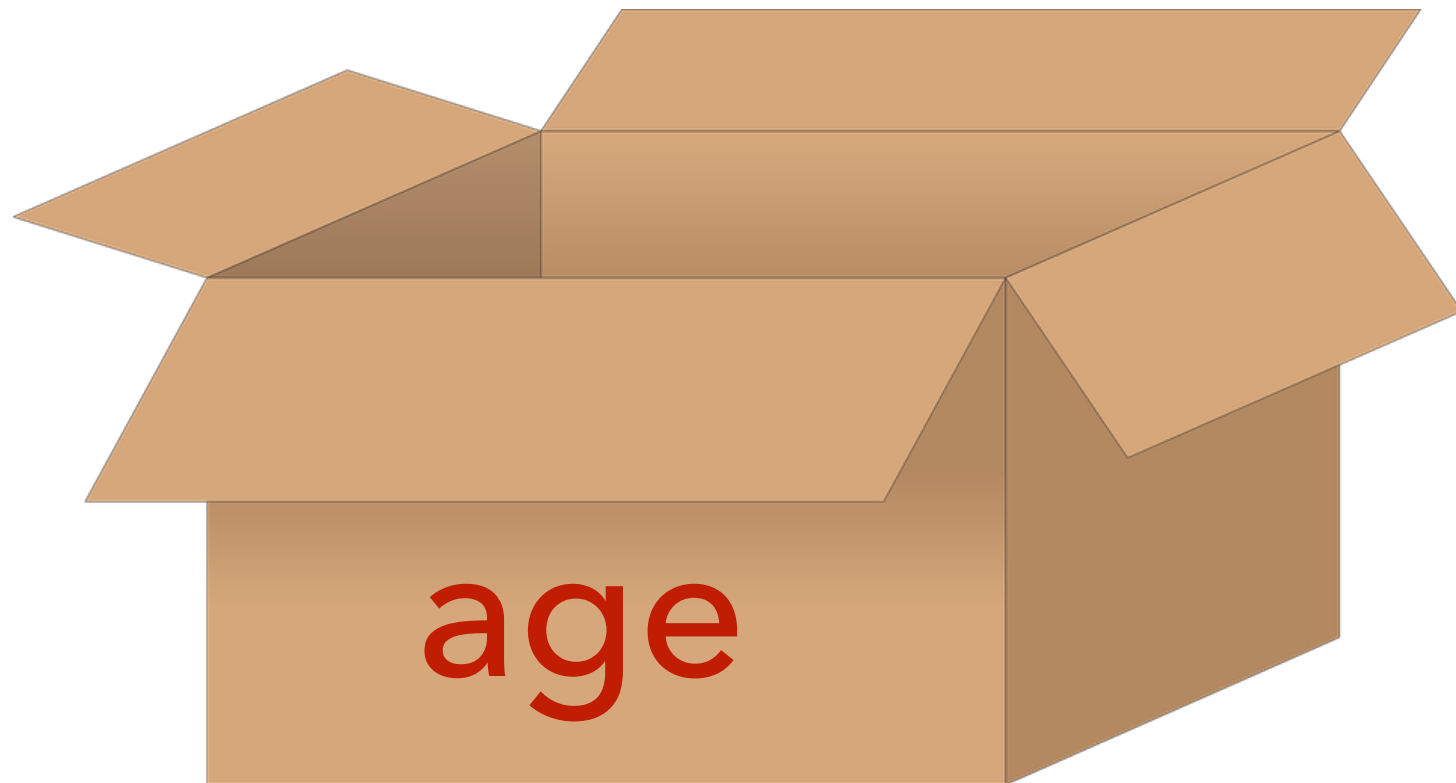
```
var age = 52;
```



## VARIABLES

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

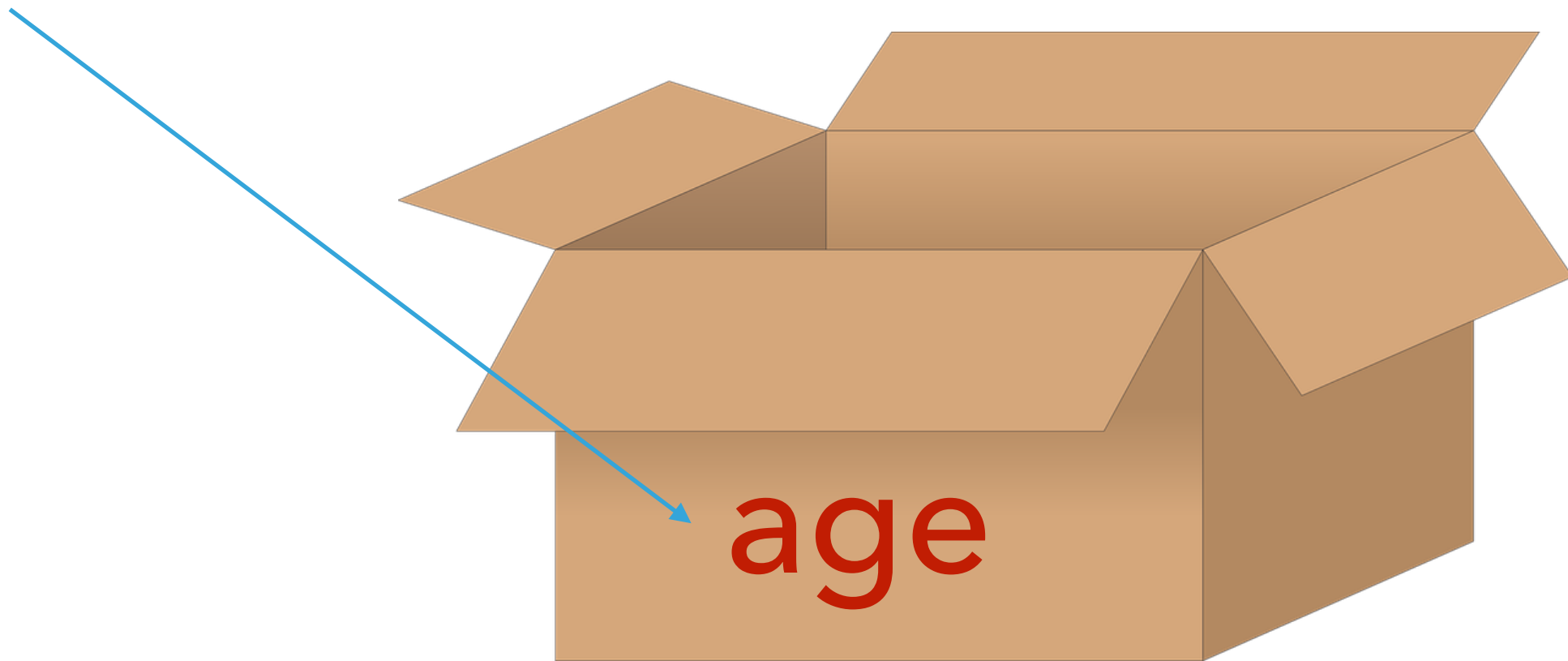
```
var age = 52;
```



## VARIABLES

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

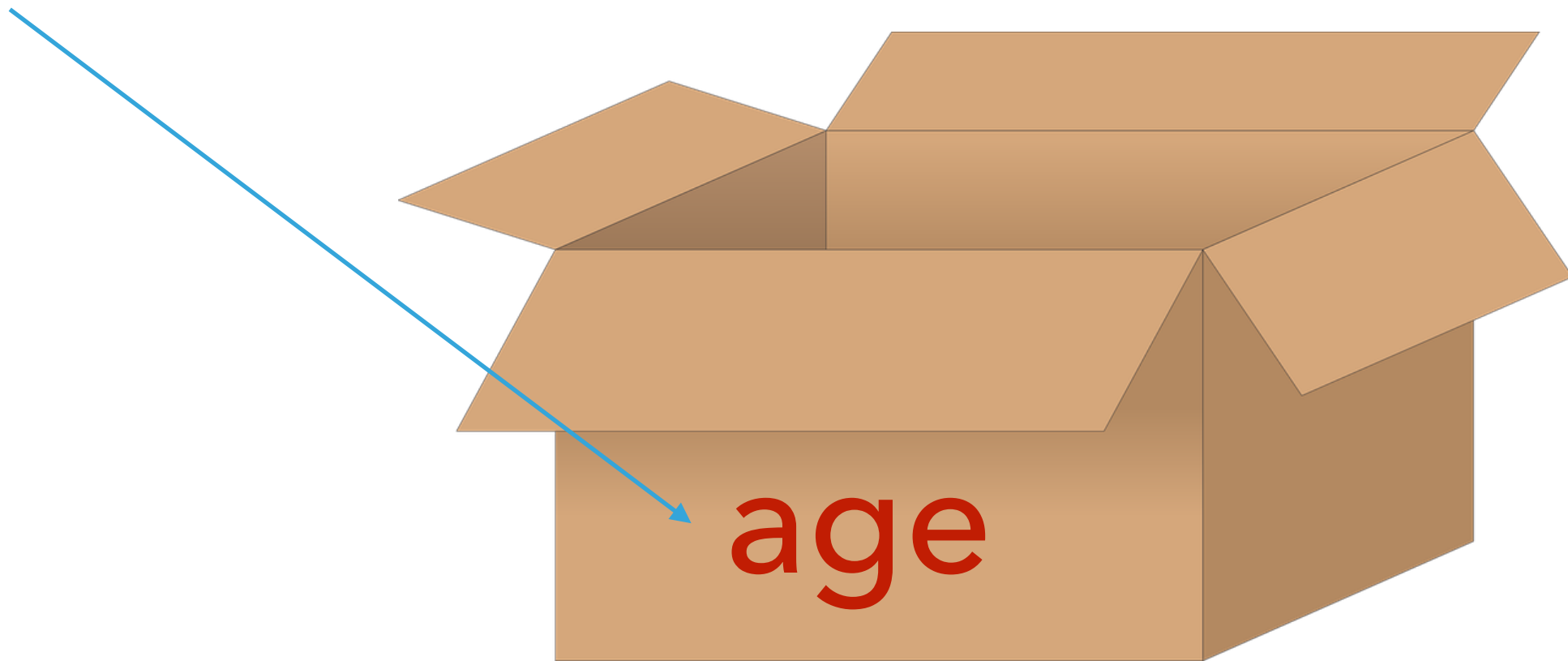
```
var age = 52;
```



## VARIABLES

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

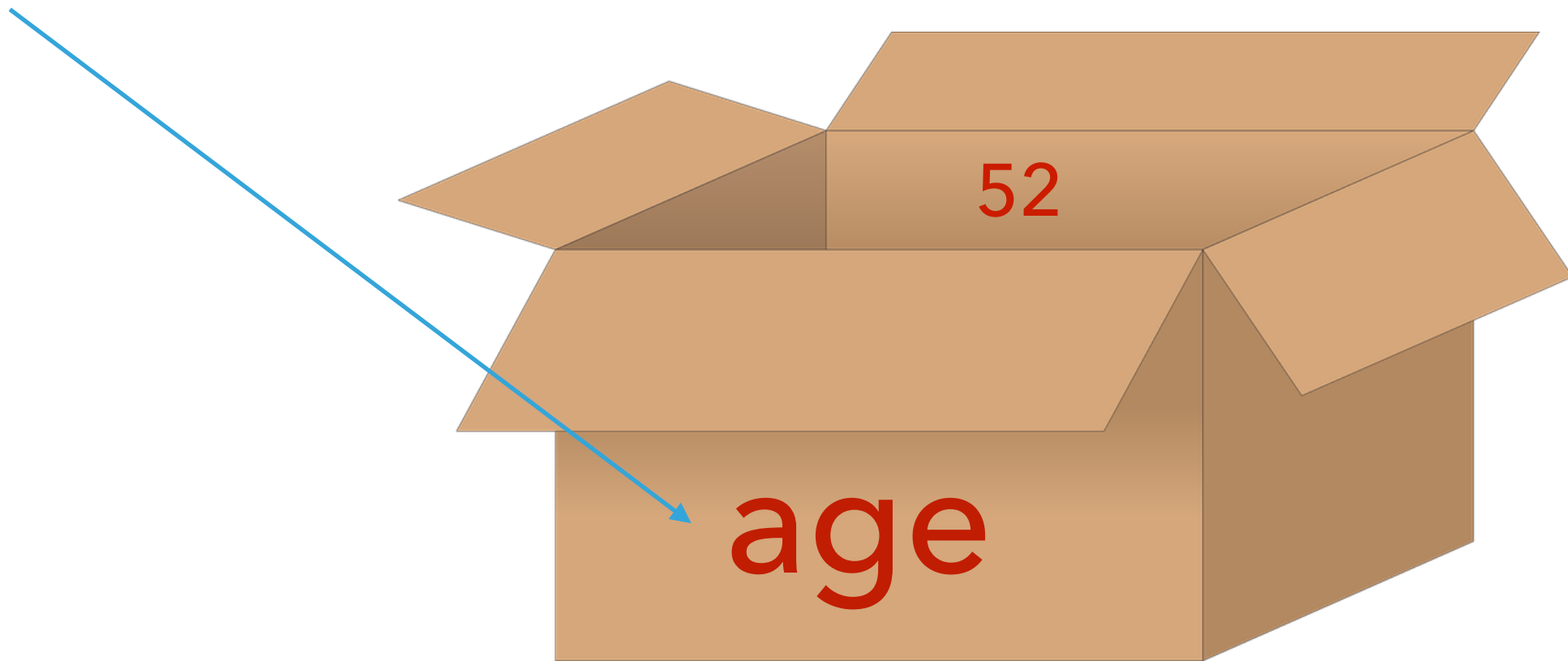
```
var age = 52;
```



## VARIABLES

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

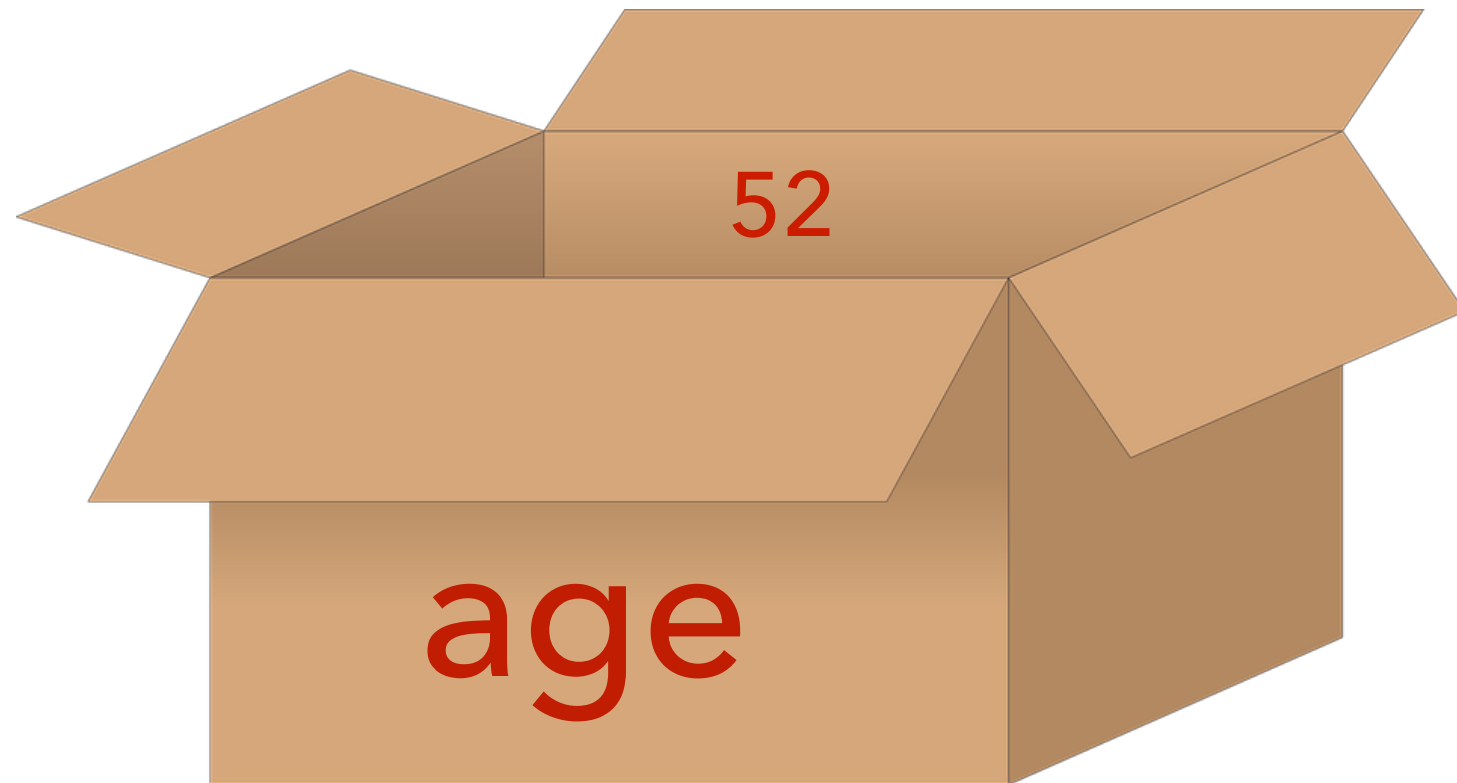
```
var age = 52;
```



## VARIABLES

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

```
var age = 52;
```

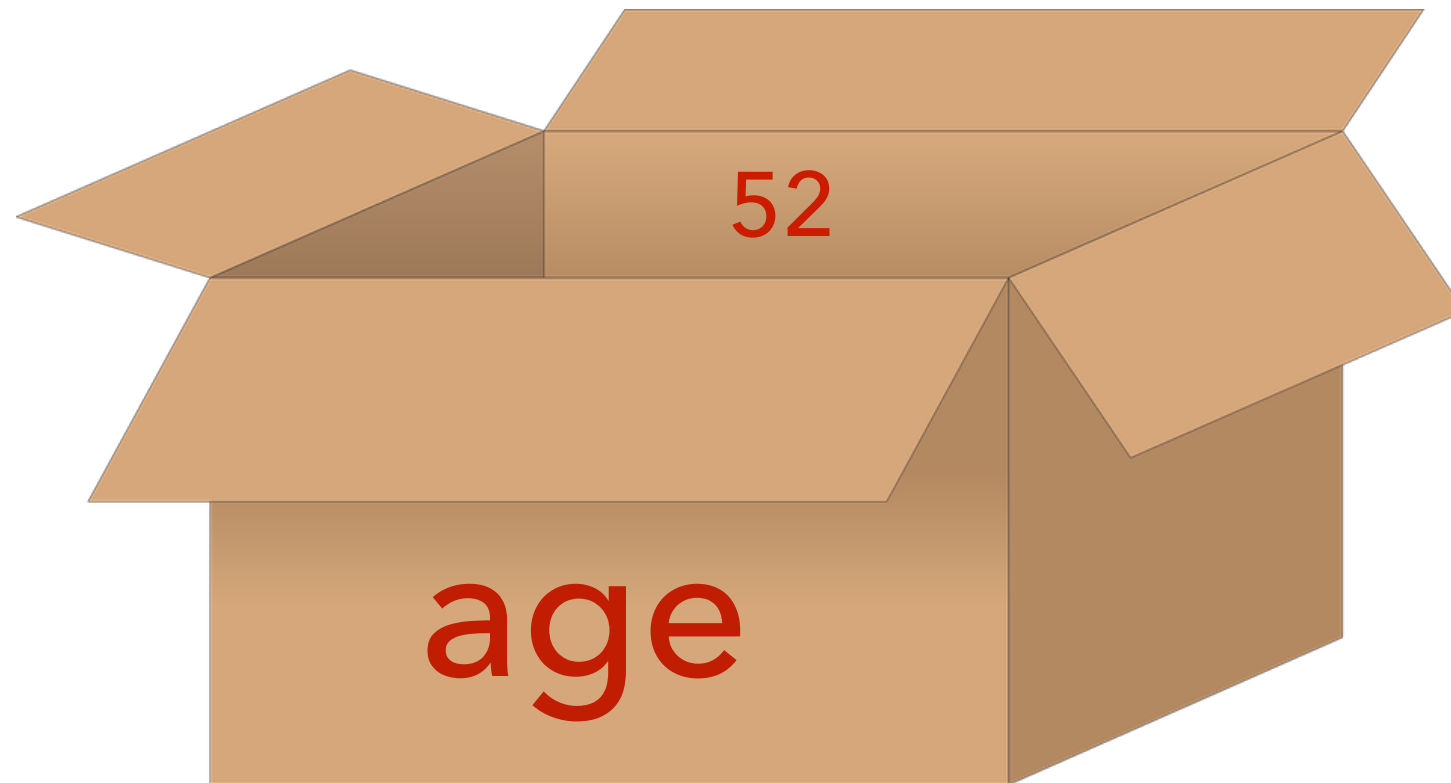


## VARIABLES

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

```
var age = 52;
```

```
age = 44;
```



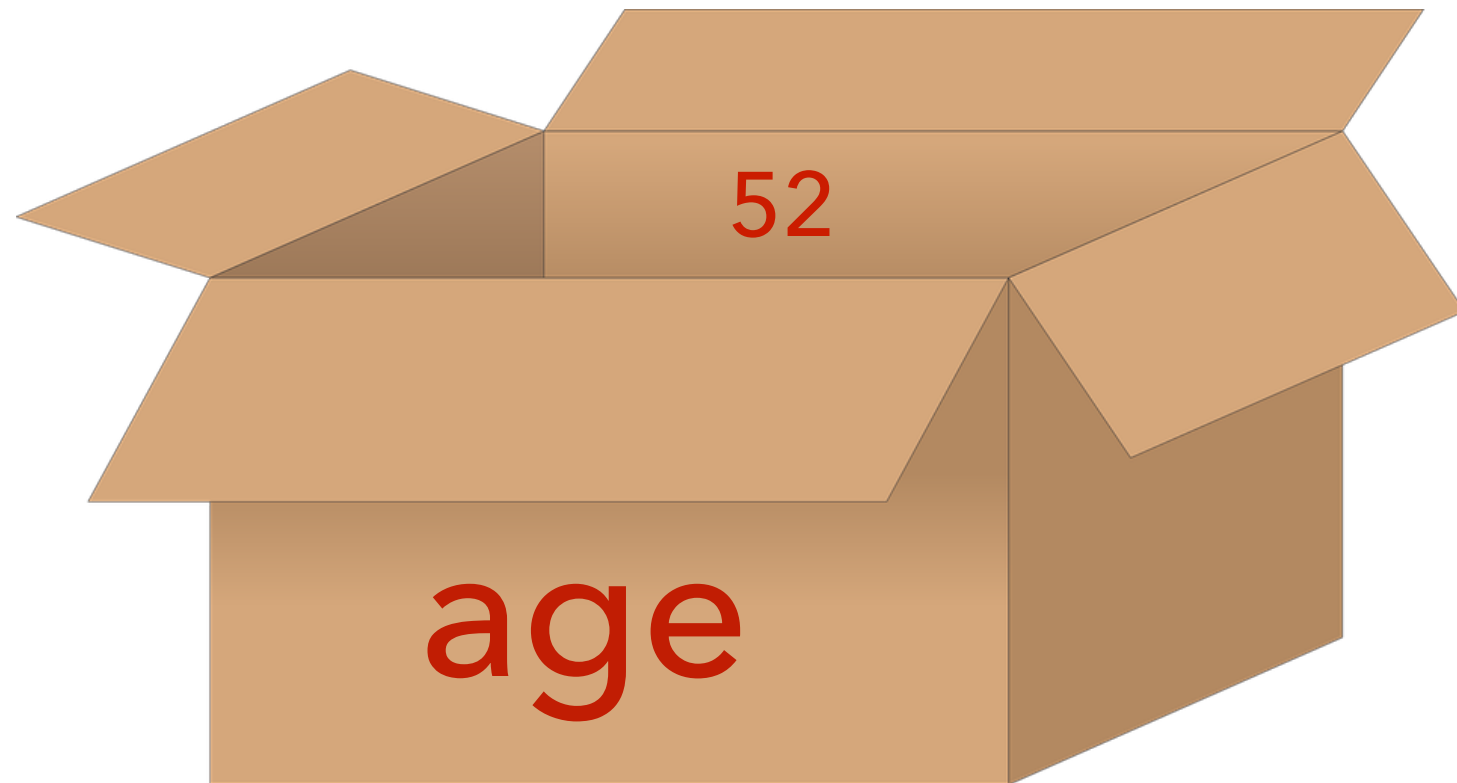
## VARIABLES

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

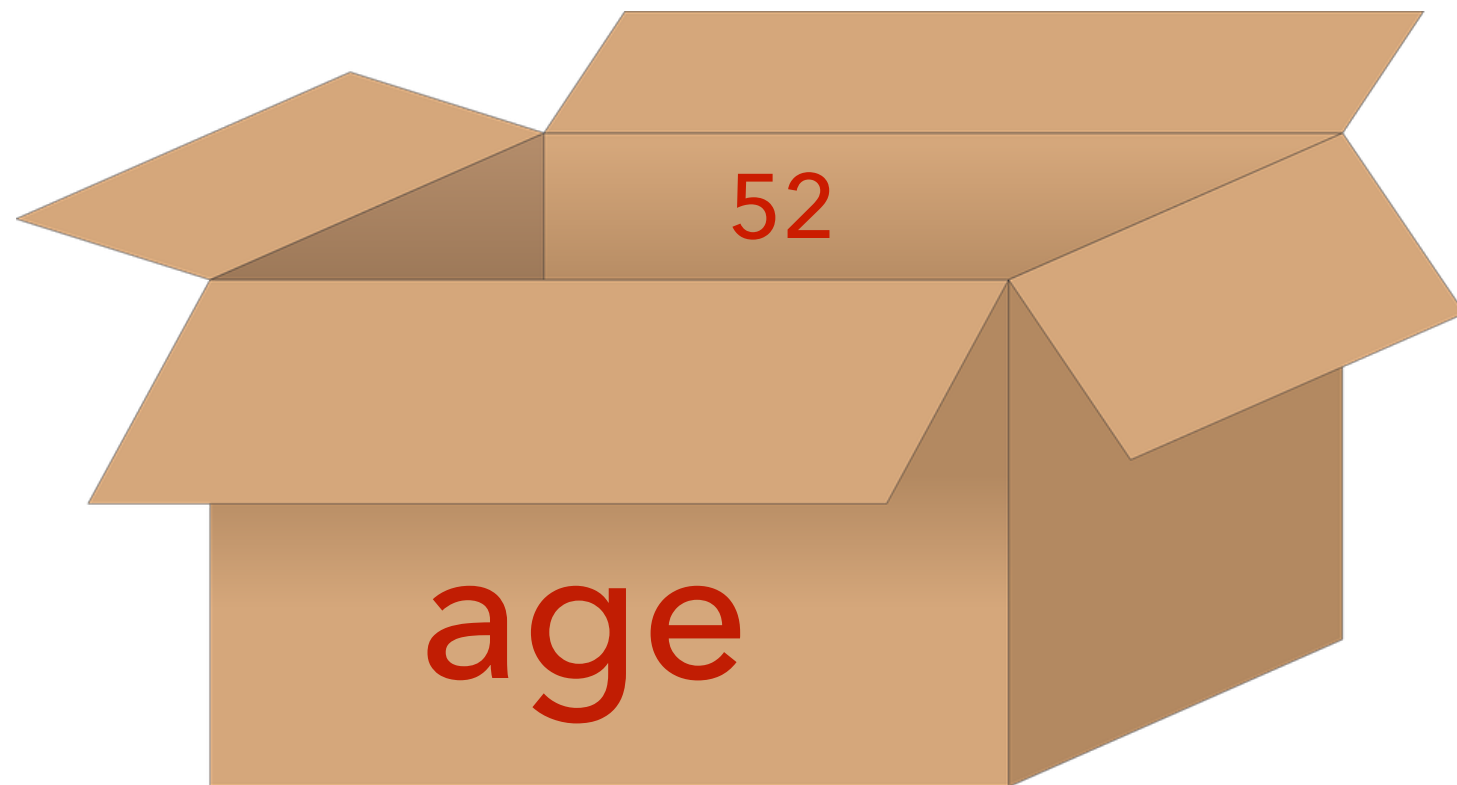
You can think of a variable as a box that we can store data inside of.

```
var age = 52;
```

```
age = 44;
```



New value assigned.



## VARIABLES

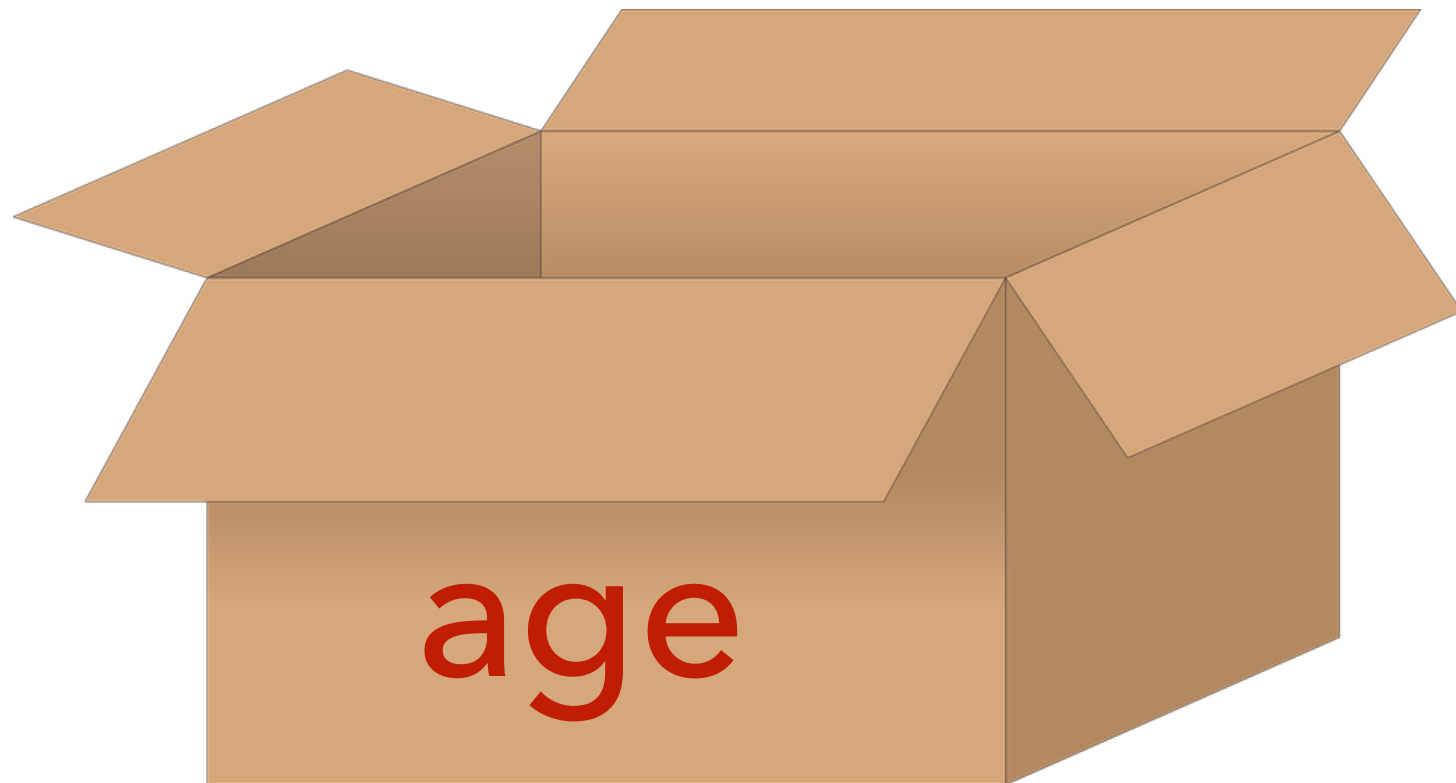
You can think of a variable as a box that we can store data inside of.

```
var age = 52;
```

```
age = 44;
```



New value assigned.



## VARIABLES

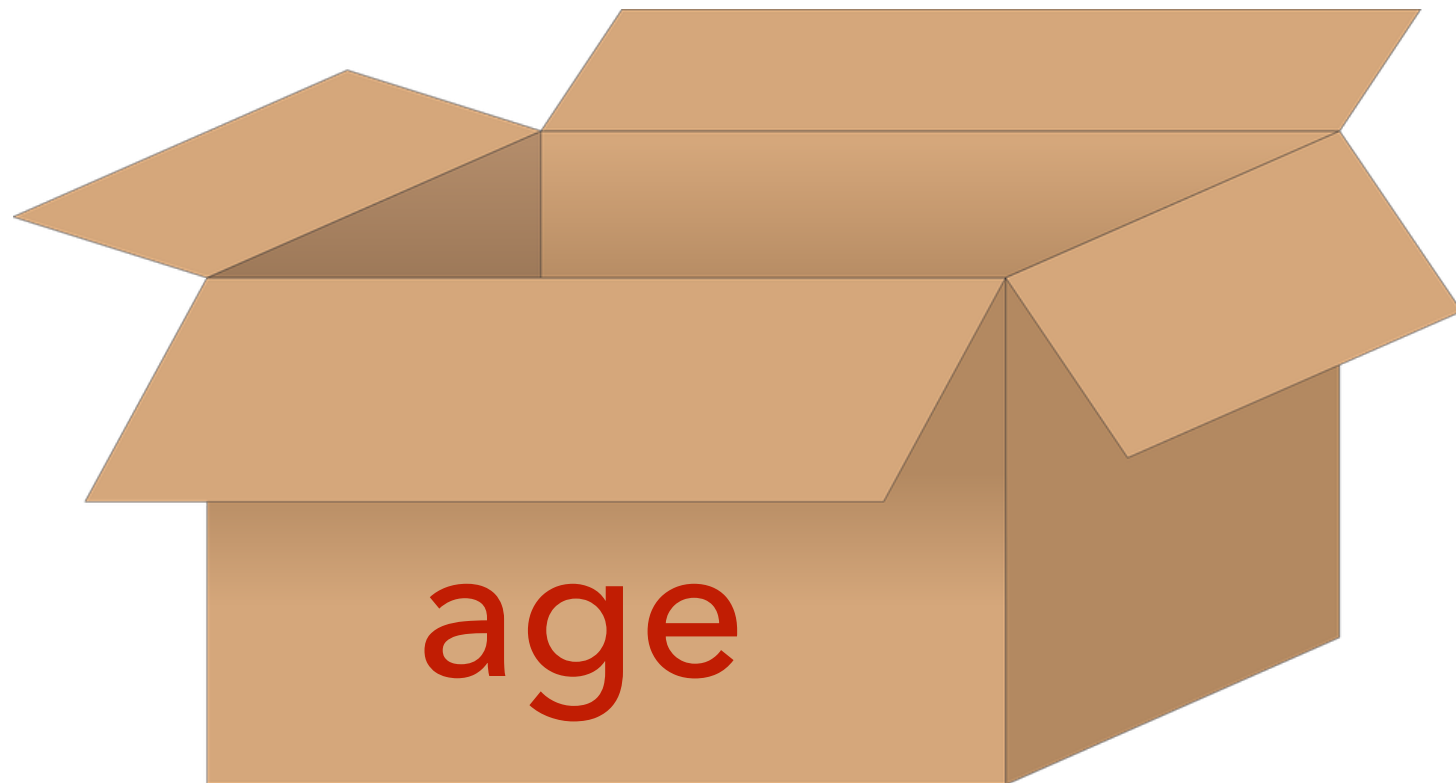
You can think of a variable as a box that we can store data inside of.

```
var age = 52;
```

```
age = 44;
```



New value assigned.



## VARIABLES

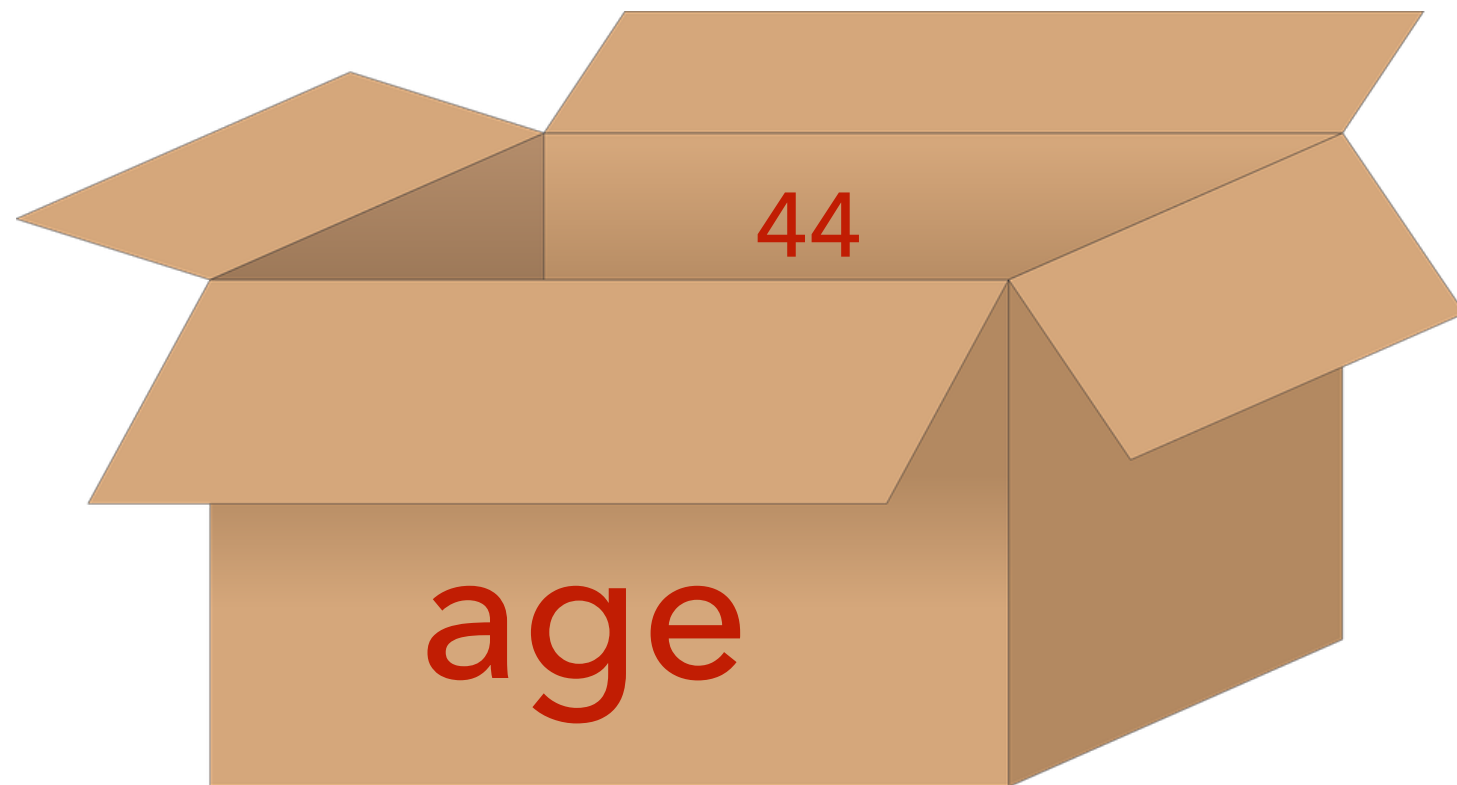
You can think of a variable as a box that we can store data inside of.

```
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



Null

Undefined

Number

String

Object

Array

Function

## DATA TYPES

Boolean



true/false

Null

Undefined

Number

String

Object

Array

Function



## DATA TYPES

Boolean



true/false

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



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



true/false

Null



null

Undefined



undefined

Number



9

String

Object

Array

Function

## DATA TYPES

Boolean



true/false

Null



null

Undefined



undefined

Number



9

String








Object

Array







Function

## DATA TYPES







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










## DATA TYPES

Boolean		true/false
Null		null
Undefined		undefined
Number		9
String		'LOTR'
Object		
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

Square brackets



[ 1 , 2 , 3 ]

## ARRAY

Square brackets



[ 1 , 2 , 3 ]

Values, separated by commas.

# ARRAY

Square brackets



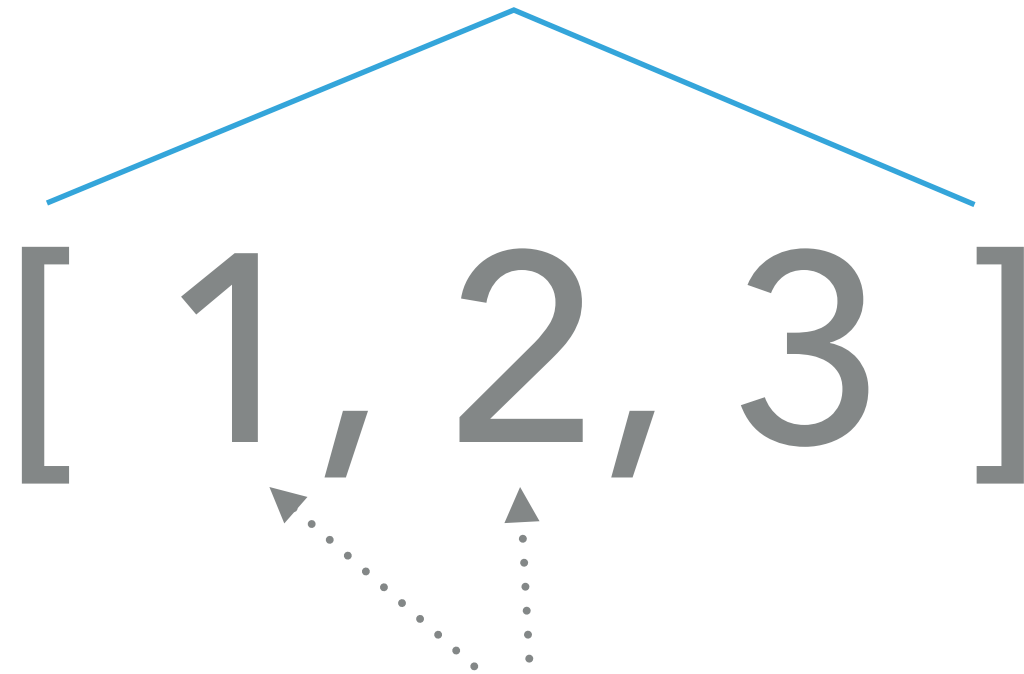
[ 1, 2, 3 ]



Values, separated by commas.

# ARRAY

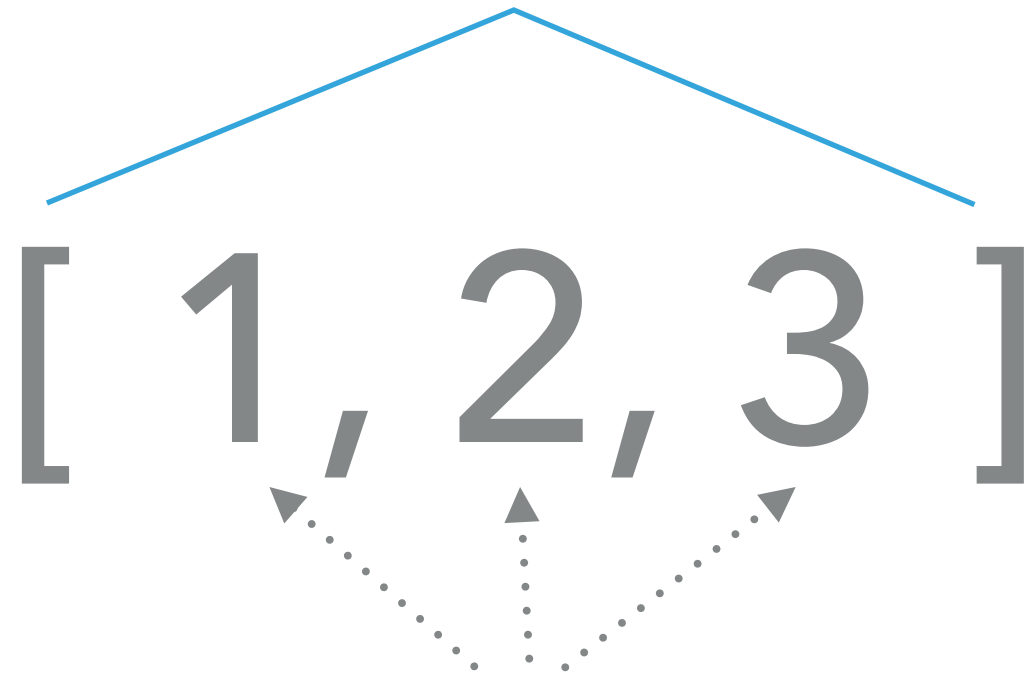
Square brackets



Values, separated by commas.

# ARRAY

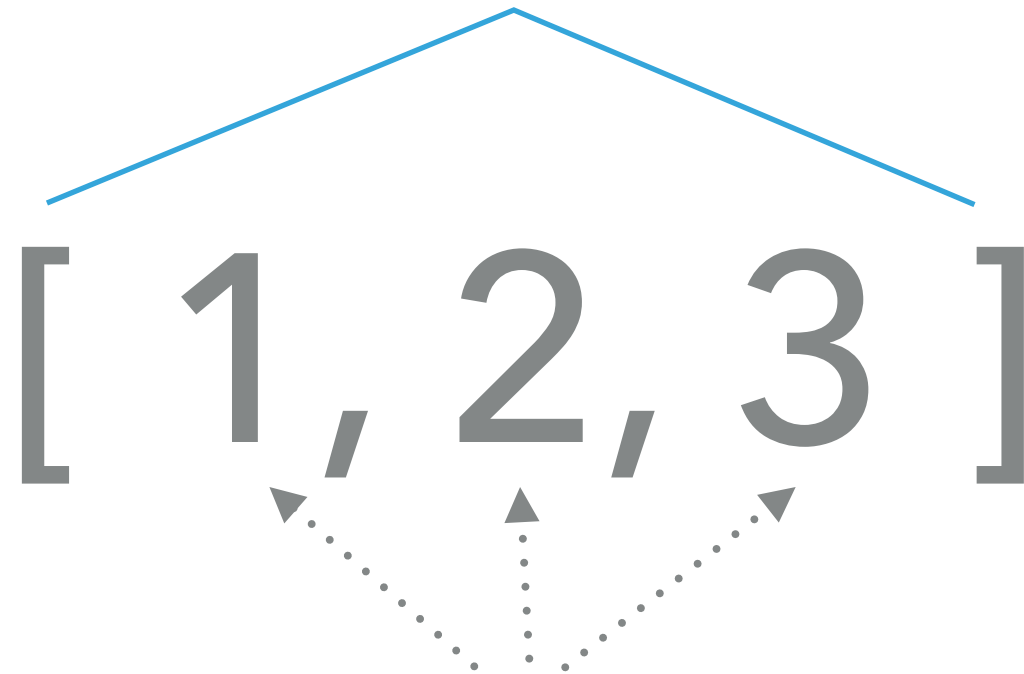
Square brackets



Values, separated by commas.

# ARRAY

Square brackets

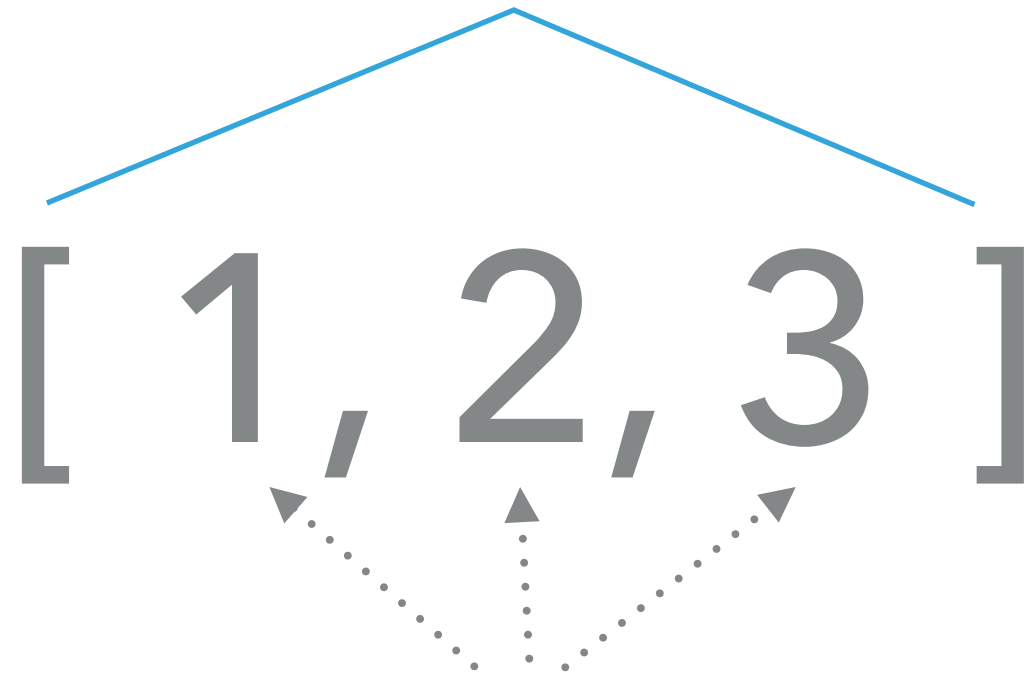


Values, separated by commas.

Values in arrays can have multiple data types.

# ARRAY

Square brackets



Values, separated by commas.

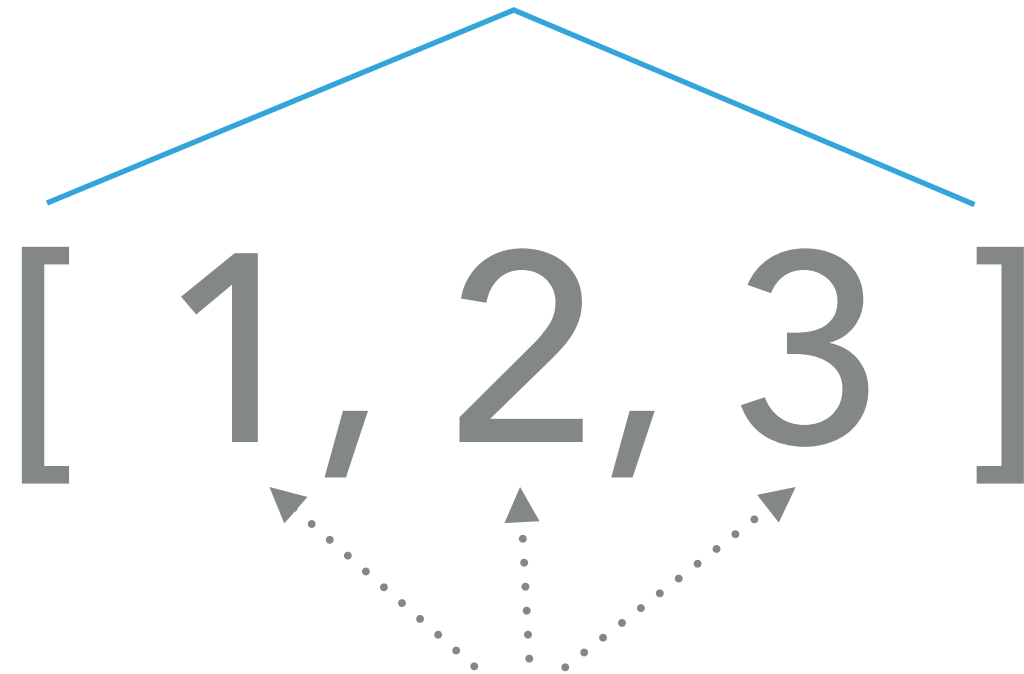
Values in arrays can have multiple data types.

`[ 1, '2', false ]`



# ARRAY

Square brackets



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

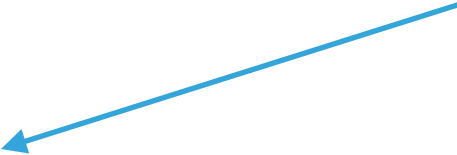
Opening curly bracket

```
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  
}
```

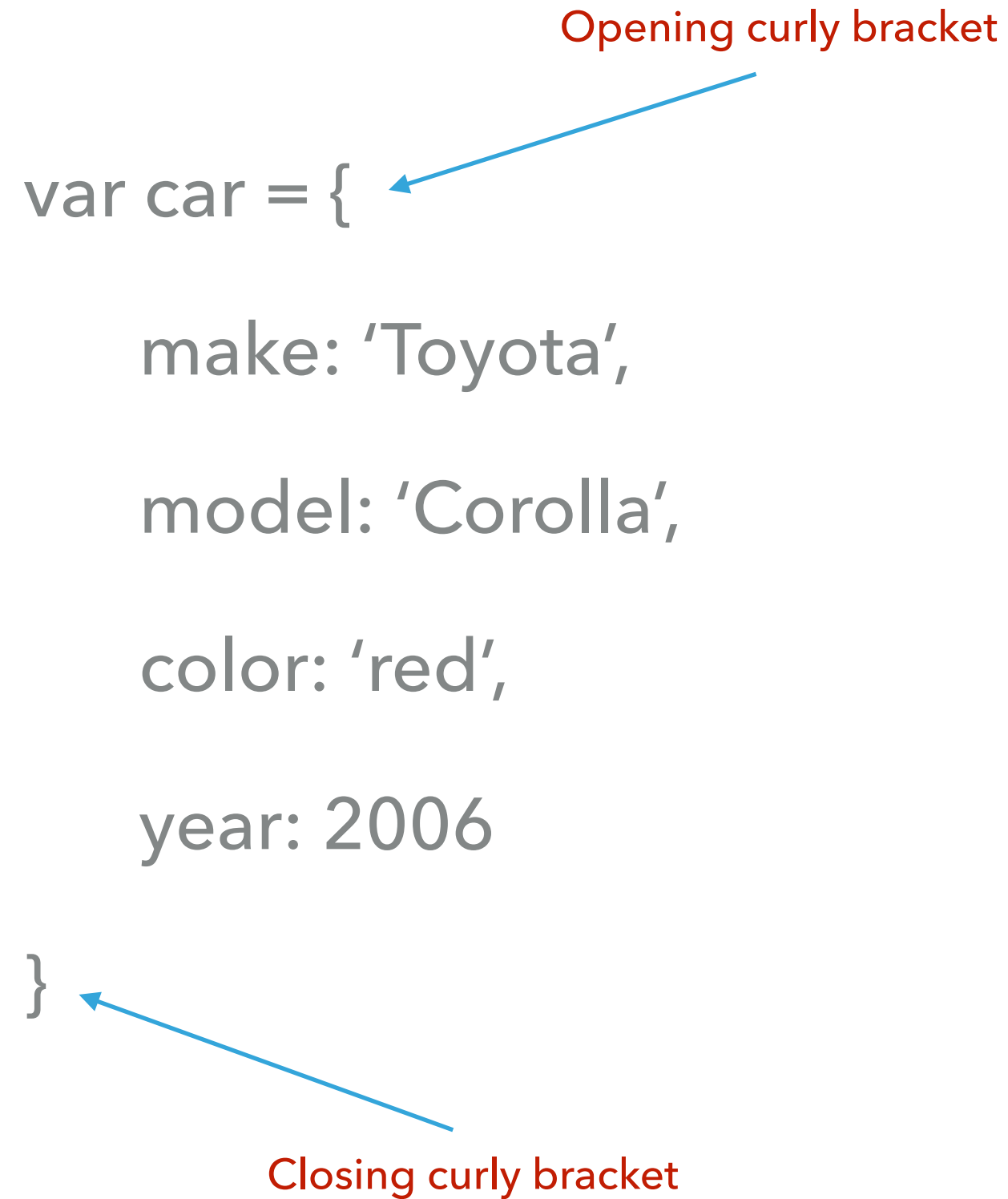
Closing curly bracket

## OBJECTS

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

Opening curly bracket

Closing curly bracket

A diagram illustrating the structure of a JavaScript object. The code 'var car = {' is shown, with a blue arrow pointing from the text 'Opening curly bracket' to the opening curly brace '{'. The object properties are listed on subsequent lines: 'make: 'Toyota'', 'model: 'Corolla'', 'color: 'red'', and 'year: 2006'. The closing curly brace '}' is shown on a new line, with a blue arrow pointing from the text 'Closing curly bracket' to it.

## OBJECTS

Opening curly bracket

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

Closing curly bracket

property



## OBJECTS

Opening curly bracket

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

Closing curly bracket

property

key

## OBJECTS

Opening curly bracket

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

Closing curly bracket

property

key

name

## OBJECTS

Opening curly bracket

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

Closing curly bracket

property

key

name

## OBJECTS

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

Opening curly bracket

Value

Closing curly bracket

property

key

name

## OBJECTS

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

Opening curly bracket

Value

Closing curly bracket

property

key

name

## OBJECTS

var car = {

make: 'Toyota',

model: 'Corolla',

color: 'red',

year: 2006

}

Opening curly bracket

Value

property

key

name

Key/value pairs  
separated by commas.

Closing curly bracket

The diagram illustrates the syntax of a JavaScript object. The code 'var car = {' is shown with an arrow pointing to the opening curly bracket, labeled 'Opening curly bracket'. The object contains four key-value pairs: 'make: 'Toyota'', 'model: 'Corolla'', 'color: 'red'', and 'year: 2006'. An arrow points from the word 'property' to the 'make' key, and another from 'key' to the same 'make' key. A third arrow points from 'name' to the 'make' key. An arrow points from the word 'Value' to the value 'Toyota'. The closing curly bracket '}' is pointed to by an arrow labeled 'Closing curly bracket'. A note on the right states 'Key/value pairs separated by commas.'.

## OBJECTS

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

Opening curly bracket

Value

property  
key  
name

Key/value pairs  
separated by commas.

Closing curly bracket

The diagram illustrates the syntax of a JavaScript object literal. It shows the code `var car = { make: 'Toyota', model: 'Corolla', color: 'red', year: 2006 }`. Annotations with blue arrows point to specific parts: 'Opening curly bracket' points to the opening brace, 'Value' points to the string 'Toyota', 'property', 'key', and 'name' all point to the word 'make', 'Key/value pairs separated by commas.' points to the comma after 'red', and 'Closing curly bracket' points to the closing brace.

**[HTTPS://REPL.IT/JZ5C/1](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 {
```

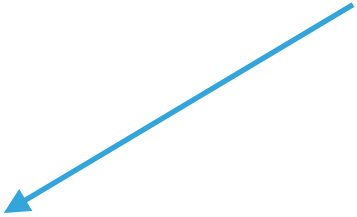
```
    console.log('five does not equal 5')
```

```
}
```

## 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')  
}
```

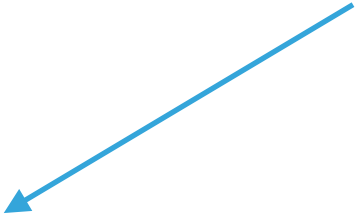


## IF STATEMENTS

```
var five = 5;  
if ( five === 5 ) {  
    console.log('Five is awesome cause it equals 5');  
} else {  
    console.log('five does not equal 5')  
}
```

condition to test

block of code that runs if condition evaluates to true



## 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')  
}
```

condition to test

block of code that runs if condition evaluates to true

## 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')  
}
```

condition to test

block of code that runs if condition evaluates to true

block of code that runs if condition evaluates to false



## 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')  
}
```

condition to test

block of code that runs if condition evaluates to true

block of code that runs if condition evaluates to false

JAVASCRIPT - DAY 1

---

# FUNCTIONS

# FUNCTIONS

--	--

## FUNCTIONS

function expression:

--	--

## FUNCTIONS

function expression:

--	--

## FUNCTIONS

function expression:

--	--

## FUNCTIONS

function expression:

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

## FUNCTIONS

function expression:

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



## FUNCTIONS

function expression:

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

## FUNCTIONS

function expression:

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

function declaration:

## FUNCTIONS

function expression:

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

function declaration:

## FUNCTIONS

function expression:

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

function declaration:

## FUNCTIONS

function expression:

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

function declaration:

```
function sayName() {
```

## FUNCTIONS

function expression:

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

function declaration:

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

## FUNCTIONS

function expression:

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

function declaration:

```
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

```
sayName( )
```

run



# FUNCTIONS

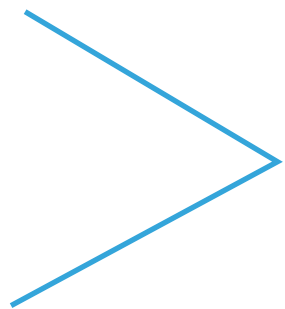
```
function sayName( ) {
```

```
}
```

call

invoke

run



```
sayName( )
```

## FUNCTIONS

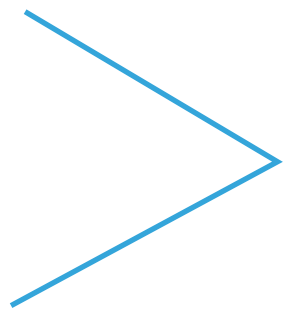
```
function sayName( ) {
```

```
}
```

call

invoke

run



```
sayName( )
```

## FUNCTIONS

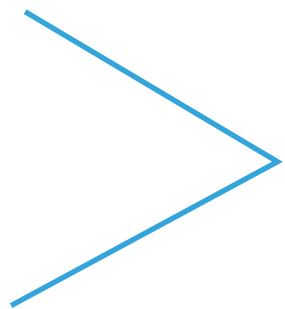
```
function sayName( ) {
```

```
}
```

call

invoke

run



```
sayName( arguments )
```

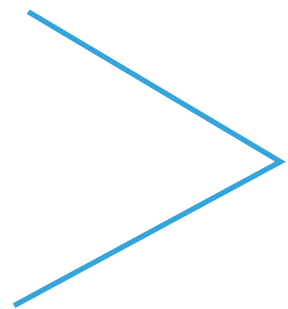
# FUNCTIONS

```
function sayName(      ) {  
  
    }  
}
```

call

invoke

run



```
sayName( arguments )
```

## FUNCTIONS

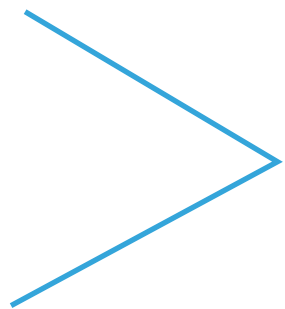
```
function sayName( parameters ) {
```

```
}
```

call

invoke

run



```
sayName( arguments )
```

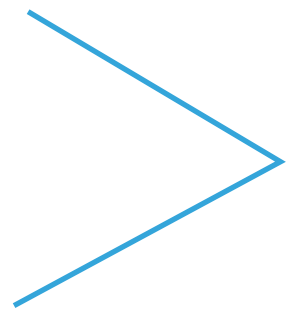
# FUNCTIONS

```
function sayName( parameters ) {  
  
}
```

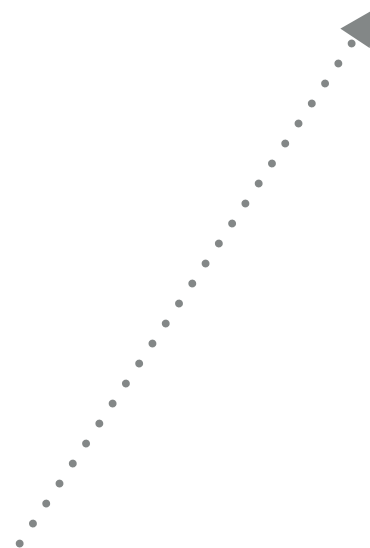
call

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;  
  
}
```

function stops executing  
when it reaches return  
statement

```
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.

	<pre>function add() {     var num = 2;     return num + num; }</pre>	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	→	
	<pre>var addedNums = add();</pre>	



## RETURNING FROM FUNCTIONS

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

We do this by using a return statement.

function stops executing  
when it reaches return  
statement

—————→

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

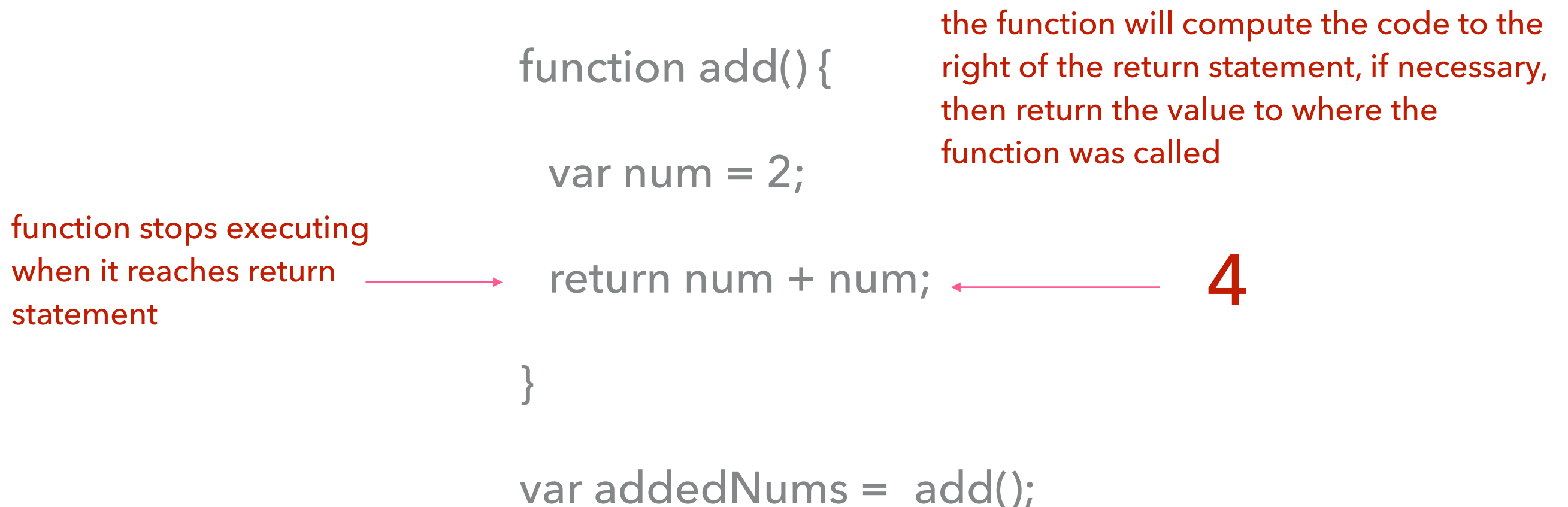
the function will compute the code to the  
right of the return statement, if necessary,  
then return the value to where the  
function was called

**4**

## 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.

	<pre>function add() {     var num = 2;     return num + num; }</pre>	<p>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</p>
<p>function stops executing when it reaches return statement</p>	<p>→</p>	<p>4</p>
	<pre>var addedNums = add();</pre>	

## 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;  
}
```

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



4

```
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 stops executing  
when it reaches 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

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.

## 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';
```

Global variable. Can be seen by all code.

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

```
console.log( name2 )
```

## SCOPE

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

Global variable. Can be seen by all code.

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

Can access name1 variable.

```
console.log( name2 )
```

## SCOPE

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

Global variable. Can be seen by all code.

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

Can access name1 variable.

```
console.log( name2 )
```

undefined

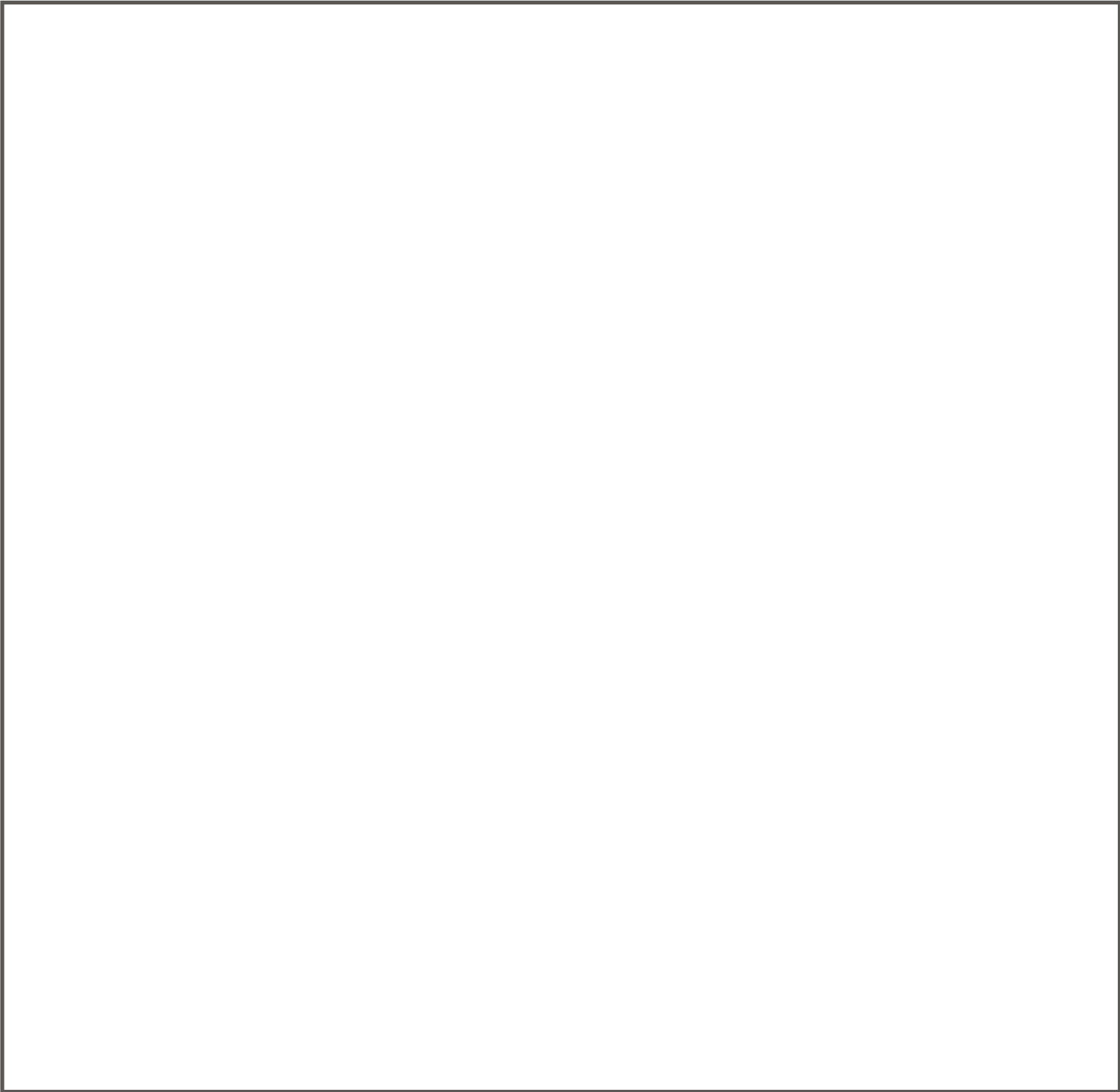
## SCOPE

## SCOPE



# SCOPE

Global scope



# SCOPE

Global scope





## SCOPE

Global scope



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

## SCOPE

Global scope



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



## SCOPE

Global scope



Function scope



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

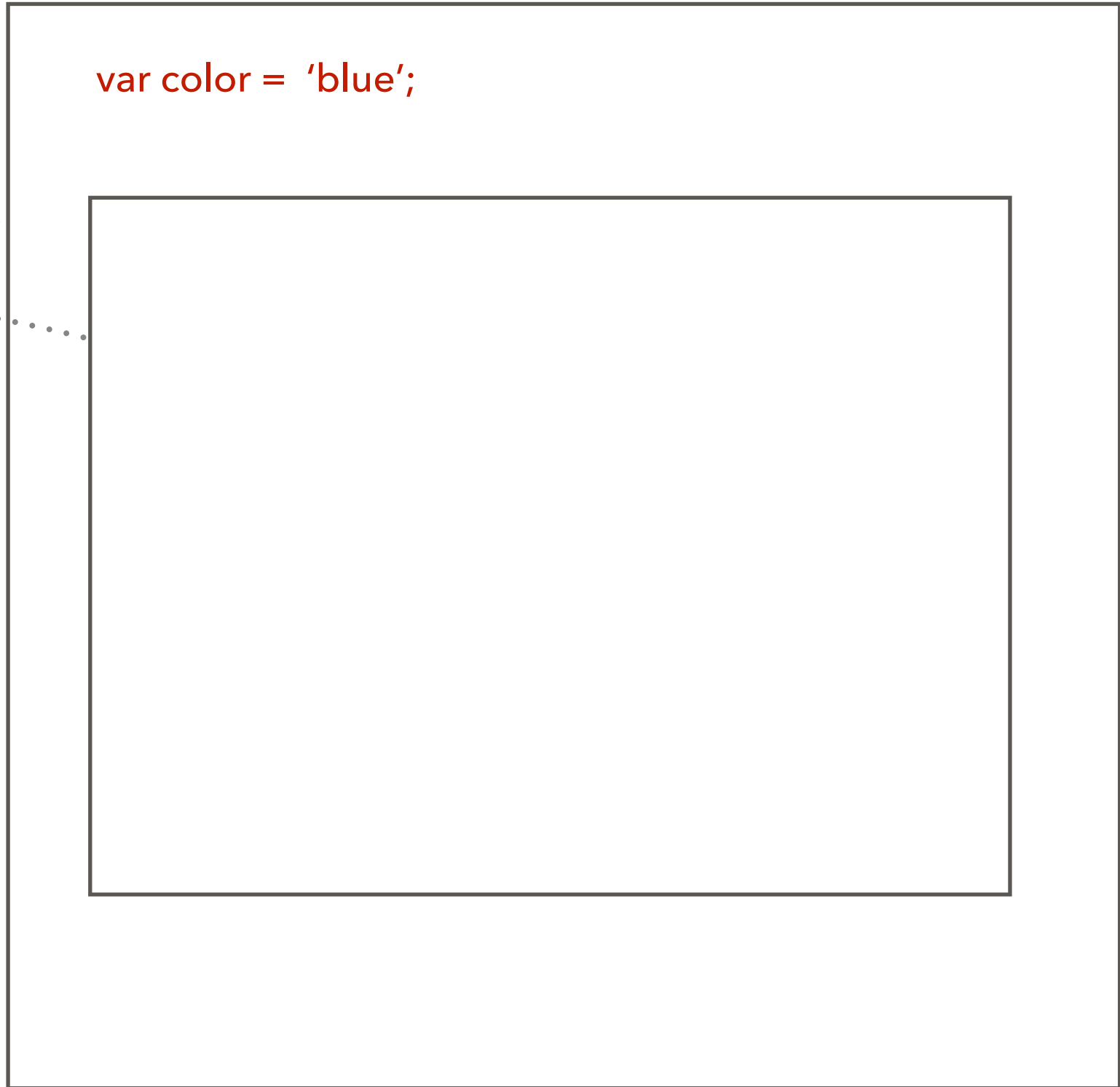
## SCOPE

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

Global scope



Function scope



## SCOPE

Global scope



Function scope



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

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

## SCOPE

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

Global scope



Function scope



## SCOPE

Global scope



Function scope



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

```
var color = 'green';
```

## SCOPE

Global scope



Function scope



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

```
var color = 'green';
```

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



## SCOPE

Global 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



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

```
var color = 'green';
```

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



## SCOPE

Global scope

Function scope

Function scope

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

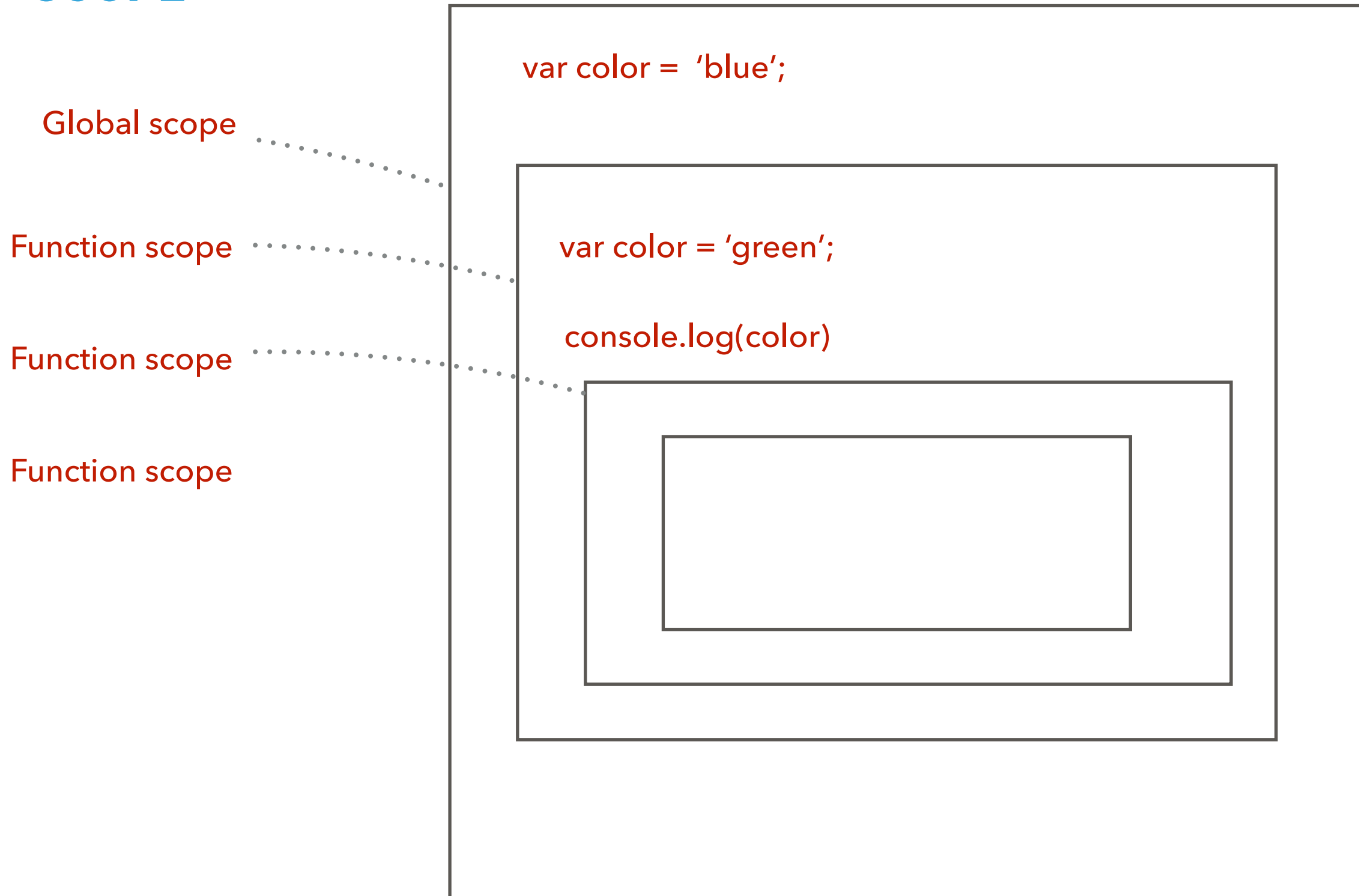
```
var color = 'green';
```

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

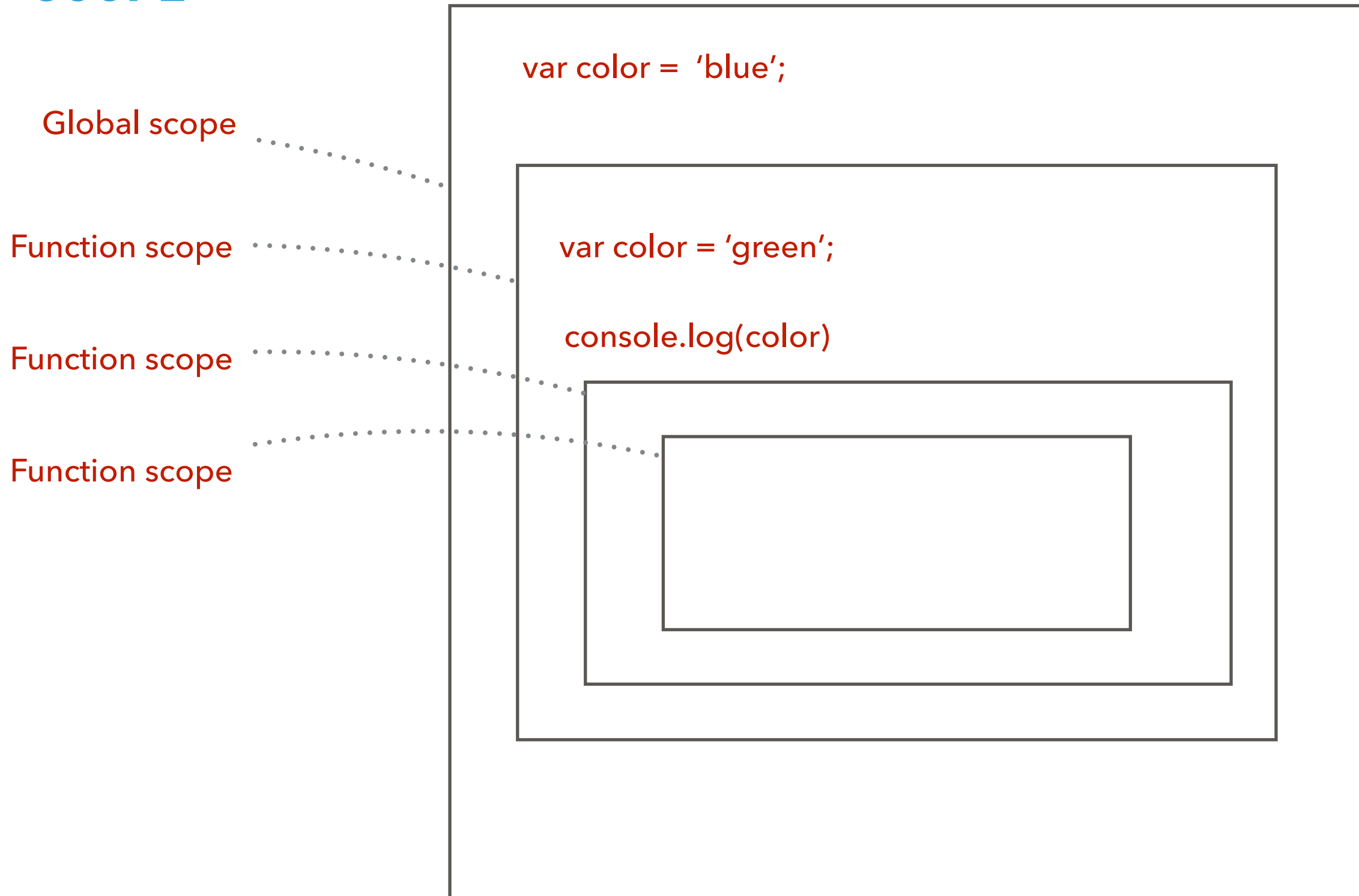


The diagram illustrates JavaScript scope resolution using four nested rectangles. The outermost rectangle represents the Global scope and contains the code `var color = 'blue';`. Inside it is a second rectangle representing a Function scope, containing `var color = 'green';` and `console.log(color)`. Inside the second rectangle is a third rectangle representing another Function scope. The innermost rectangle is empty. Dotted lines connect the labels on the left to the corresponding rectangles: 'Global scope' to the outermost, 'Function scope' to the second, and another 'Function scope' to the third.

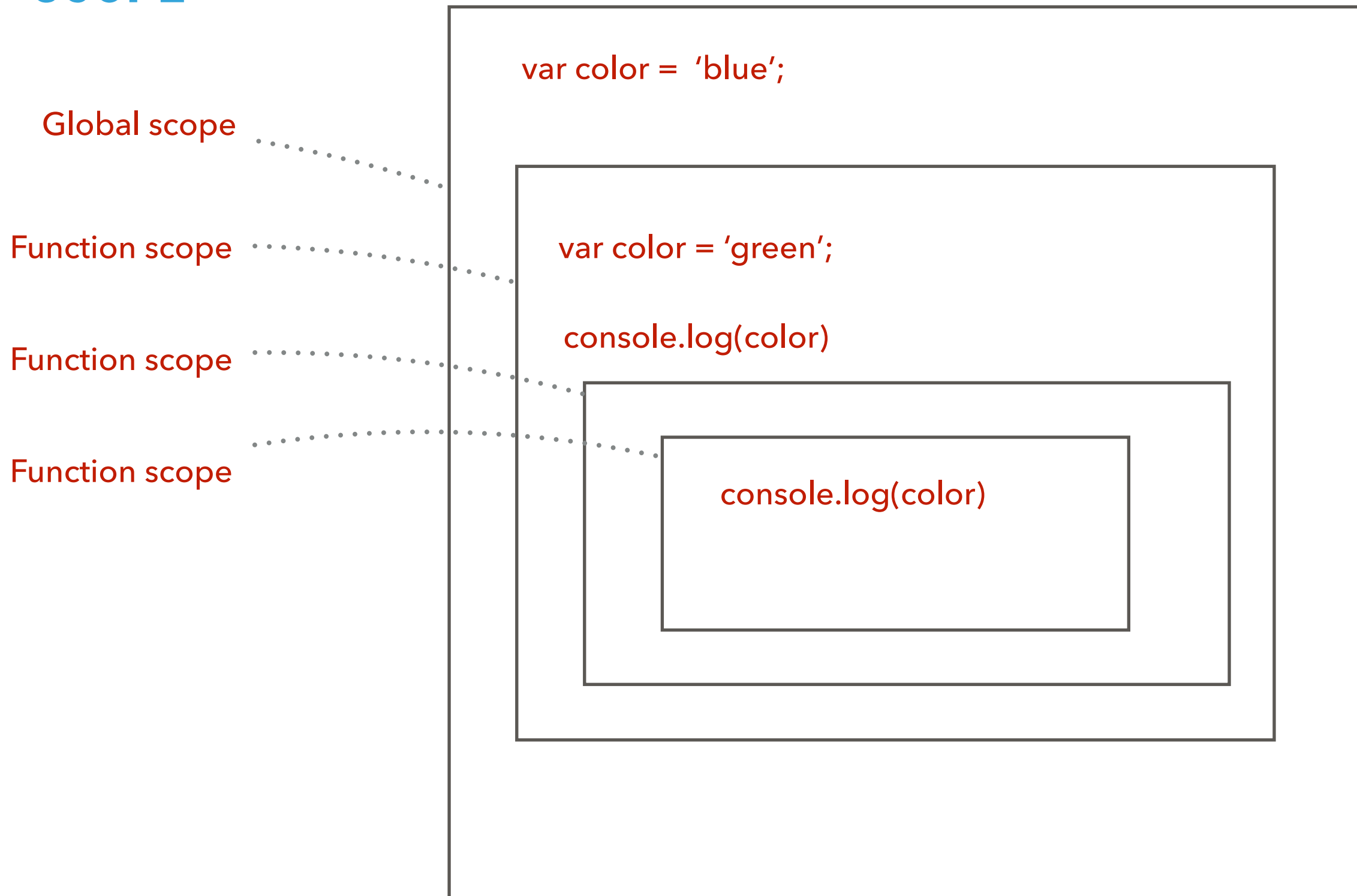
## SCOPE



## SCOPE



## SCOPE



**[HTTPS://REPL.IT/KAP7/2](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 );
```

## LET

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

```
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