JAVASCRIPT - DAY 1

VARIABLES

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

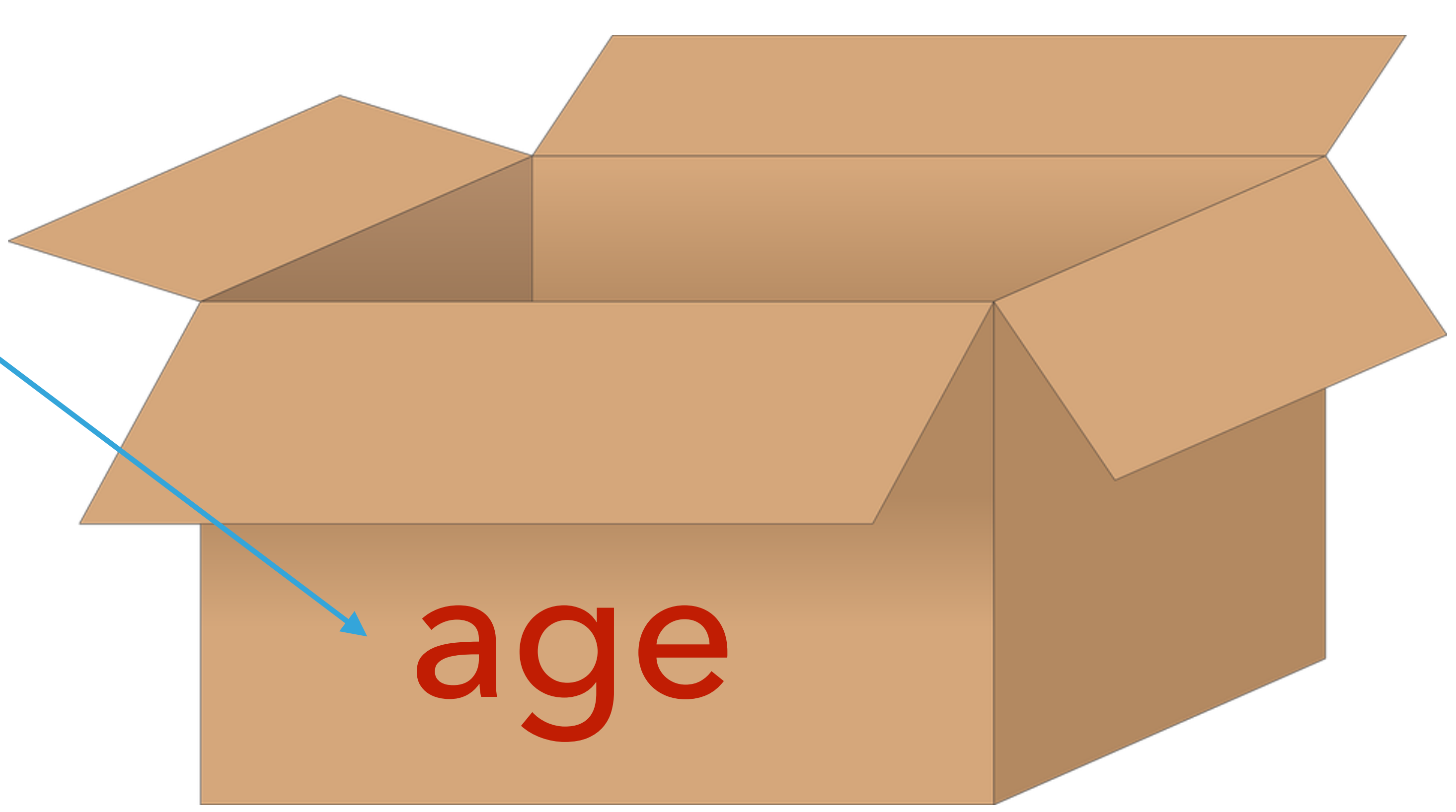
in the variable ‘myName‘.

VARIABLES

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

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

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

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

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

function expression:

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

function declaration:

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

FUNCTIONS

```
function sayName( ){

}
```

parameters

call

invoke

run

```
sayName( )
```

arguments

HTTPS://REPL.IT/KAPA/3

JAVASCRIPT - DAY 1

RETURNING FROM FUNCTIONS

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

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

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

Global variable. Can be seen by all code.

Functions have their own scope.

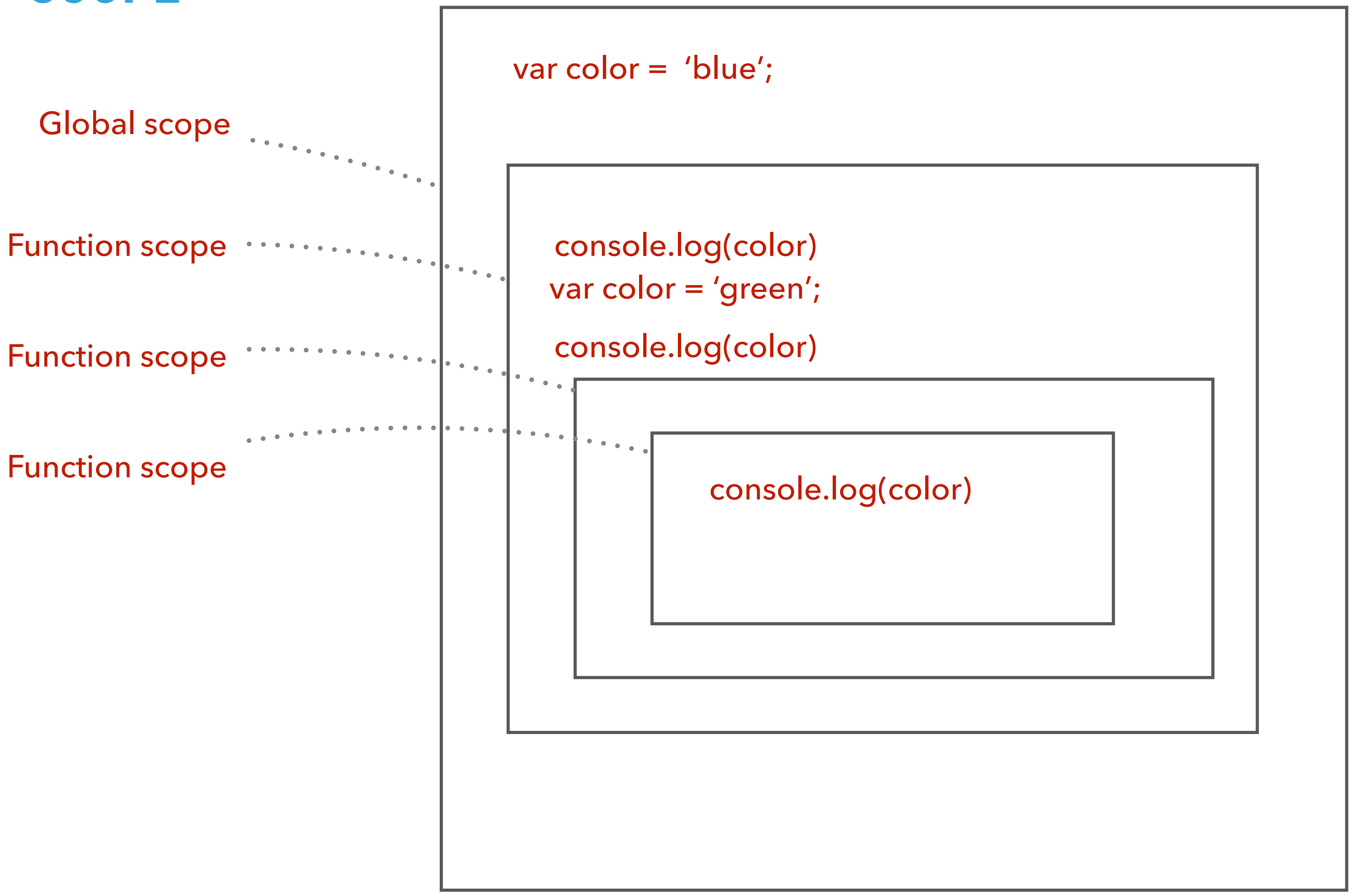
```
function sayName() {

console.log( name1);

var name2 = 'Nancy';

}
```

Can access name1 variable.

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

undefined

SCOPE



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

```
console.log(color)
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

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

ERROR

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

ERROR

LET

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

TRUE

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

ERROR

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