CLOSURES

BY DEFINITION, A NESTED FUNCTION NEEDS AN OUTER FUNCTION

SAY WE HAVE A NESTED FUNCTION

CLOSURES

OUTER FUNCTION

VARIABLES LOCAL TO THE OUTER FUNCTION

NESTED FUNCTION
CAN BE ACCESSED FROM HERE..

CLOSURES

EVEN AFTER THE OUTER FUNCTION CEASES TO EXIST

CAN BE ACCESSED FROM HERE..

CLOSURES

EVEN AFTER THE OUTER FUNCTION CEASES TO EXIST

CAN BE ACCESSED FROM HERE..

CLOSURES

OUTER FUNCTION

VARIABLES LOCAL TO THE OUTER FUNCTION

NESTED FUNCTION

CAN BE ACCESSED FROM HERE..
EVEN AFTER THE OUTER FUNCTION
CEASES TO EXIST

CLOSURES

BY DEFINITION, A NESTED FUNCTION NEEDS AN OUTER FUNCTION

VARIABLES LOCAL TO THE OUTER FUNCTION

NESTED FUNCTION

CAN BE ACCESSED FROM HERE..
EVEN AFTER THE OUTER FUNCTION
CEASES TO EXIST

CLOSURE = SAY WE HAVE A NESTED FUNCTION + VARIABLES LOCAL TO THE OUTER FUNCTION

CLOSURE = SAY WE HAVE A NESTED FUNCTION + VARIABLES LOCAL TO THE OUTER SCOPE

SCOPE IS THE MORE TECHNICAL, AND GENERAL TERM FOR THE OUTER FUNCTION

CLOSURE = SAY WE HAVE A NESTED FUNCTION + VARIABLES LOCAL TO THE OUTER SCOPE

SCOPE IS THE MORE TECHNICAL, AND GENERAL TERM FOR THE OUTER FUNCTION

CLOSURE = SAY WE HAVE A NESTED FUNCTION + VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

REMEMBER THIS EQUATION, AND WE WILL BE JUST FINE!

CLOSURE =

SAY WE HAVE A NESTED FUNCTION +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

THIS SEEMS LIKE MAGIC - AND IT IS.

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #1: THE NESTED FUNCTION CAN ACCESS THE REFERENCING ENVIRONMENT - EVEN THOUGH THOSE VARIABLES ARE OUTSIDE THE SCOPE

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #1: THE NESTED FUNCTION CAN ACCESS THE REFERENCING ENVIRONMENT - EVEN THOUGH THOSE VARIABLES ARE OUTSIDE THE SCOPE

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #1: THE NESTED FUNCTION CAN ACCESS THE REFERENCING ENVIRONMENT - EVEN THOUGH THOSE VARIABLES ARE OUTSIDE THE SCOPE

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #2: THE NESTED FUNCTION CARRIES AROUND THAT REFERENCING ENVIRONMENT EVEN AFTER THE SCOPE HAS "GONE AWAY"!

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #2: THE NESTED FUNCTION CARRIES AROUND THAT REFERENCING ENVIRONMENT EVEN AFTER THE SCOPE HAS "GONE AWAY"!

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #2: THE NESTED FUNCTION CARRIES AROUND THAT REFERENCING ENVIRONMENT **EVEN AFTER THE SCOPE HAS "GONE AWAY"!**

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

MAGIC TRICK #2: THE NESTED FUNCTION CARRIES AROUND THAT REFERENCING ENVIRONMENT EVEN AFTER THE SCOPE HAS **"GONE AWAY"!**

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

NOW THERE COULD BE **MULTIPLE NESTED FUNCTIONS** IN THE REFERENCING ENVIRONMENT -

THEY WILL ALL **SHARE THE SAME VARIABLES!** (NOT COPIES!!)

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

THE NESTED FUNCTION COULD BE EITHER A DECLARED FUNCTION OR A FUNCTION LITERAL

EITHER TYPE OF NESTED FUNCTION WILL DO JUST FINE!

CLOSURE =

SAY WE HAVE A **NESTED FUNCTION** +

VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

TYPICALLY, THE OUTER SCOPE IS A FUNCTION THAT RETURNS THE NESTED FUNCTION..

..BUT IT COULD ALSO BE THAT THE NESTED FUNCTION IS PASSED IN AS A FUNCTION ARGUMENT (MORE SOON!)

LETS' APPLY OUR EQUATION TO THE EXAMPLE WE JUST SAW

CLOSURE = SAY WE HAVE A NESTED FUNCTION + VARIABLES LOCAL TO THE OUTER SCOPE

"REFERENCING ENVIRONMENT"

TO MAKE A PROPERTY PRIVATE, JUST MAKE IT LOCAL TO THE CONSTRUCTOR:-)

```
function Rectangle(length,breadth,color) {
  this.length = length;
  this.breadth = breadth;
  this.color = color;

  var privateVar = "I don't want anyone to know this, but I am actually not just a rectangle, but also a square”;

  this.sayHello = function() {
    console.log(privateVar);
  };
}
```

CLOSURE =

SAY WE HAVE A NESTED + VARIABLES LOCAL TO THE OUTER SCOPE "REFERENCING

TO MAKE A PROPERTY PRIVATE, JUST MAKE IT LOCAL TO THE CONSTRUCTOR :-)

```
function Rectangle(length,breadth,color) {
  this.length = length;
  this.breadth = breadth;
  this.color = color;

  var privateVar = "I don't want anyone to know this, but I am actually not just a rectangle, but also a square";

  this.sayHello = function() {
    console.log(privateVar);
  };
}
```

CLOSURE = SAY WE HAVE A NESTED + VARIABLES LOCAL TO THE OUTER SCOPE "REFERENCING

TO MAKE A PROPERTY PRIVATE, JUST MAKE IT LOCAL TO THE CONSTRUCTOR:-)

```
function Rectangle(length,breadth,color) {
  this.length = length;
  this.breadth = breadth;
  this.color = color;

  var privateVar = "I don't want anyone to know this, but I am actually not just a rectangle, but also a square";

  this.sayHello = function() {
    console.log(privateVar);
  };
}
```

CLOSURE = SAY WE HAVE A NESTED + VARIABLES LOCAL TO THE OUTER SCOPE "REFERENCING

SCOPE

"REFERENCING

```
function Rectangle(length,breadth,color) {
  this.length = length;
  this.breadth = breadth;
  this.color = color;

  var privateVar = "I don't want anyone to know
this, but I am actually not just a rectangle, but
also a square";

  this.sayHello = function() {
    console.log(privateVar);
  };
}
```

CLOSURE = SAY WE HAVE A NESTED + VARIABLES LOCAL TO SCOPE "REFERENCING

WHY ARE CLOSURES SO IMPORTANT?

ITS BECAUSE MANY JAVASCRIPT FRAMEWORKS ARE BUILT ATOP THEM.

JQUERY, NODE.JS, ANGULAR, ETC

WHY ARE CLOSURES SO IMPORTANT?

ITS BECAUSE MANY JAVASCRIPT FRAMEWORKS ARE BUILT ATOP THEM.

JQUERY, NODE.JS, ANGULAR, ETC

ERRM..WHY ARE FRAMEWORKS BUILT USING CLOSURES?

BECAUSE THEY OFFER WAYS TO GET COMPLICATED STUFF DONE IN JAVASCRIPT (EG ACCESS MODIFIERS!)