EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
function Circle(r) {
  this.radius = r;
}
var PI = 3.14;

window.onload = function(){
  var circle1 = new Circle(3);
  var circle2 = new Circle(4);
  var circle3 = new Circle(5);

var circleArray = [circle1,circle2,circle3];
var PI = 3.1415;


var printStuffAboutCircleArray = function(circleArray) {
  var PI = 3.14159;

  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
     var c = circleArray[i];
     console.log(c.radius + "," + getArea(c));
```

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var printStuffAboutCircleArray = function(circleArray) {
  var PI = 3.14159;
  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
     var c = circleArray[i];
     console.log(c.radius + " " + getArea(c));
  }
 return getArea;
}
```

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

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var getArea = function(circle){
  console.log("Inside the nested function getArea, PI = " + PI);
  return PI * circle.radius * circle.radius;
};
```

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

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var PI = 3.14159;

var getArea = function(circle){
  console.log("Inside the nested function getArea, PI = " + PI);
  return PI * circle.radius * circle.radius;
};
for (var i = 0;i<circleArray.length;i++) {
   var c = circleArray[i];
   console.log(c.radius + " " + getArea(c));
}
return getArea;
}

var areaFunction = printStuffAboutCircleArray(circleArray);
areaFunction(circle1);
```

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

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var printStuffAboutCircleArray = function(circleArray) {
  var PI = 3.14159;

  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
     var c = circleArray[i];
     console.log(c.radius + " " + getArea(c));
  }
 return getArea;
}
```

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

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
function Circle(r) {
  this.radius = r;
}
var PI = 3.14;

window.onload = function(){
  var circle1 = new Circle(3);
  var circle2 = new Circle(4);
  var circle3 = new Circle(5);

var circleArray = [circle1,circle2,circle3];
var PI = 3.1415;


var printStuffAboutCircleArray = function(circleArray) {
  var PI = 3.14159;

  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
     var c = circleArray[i];
     console.log(c.radius + "," + getArea(c));
```

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var PI = 3.14;
```

GLOBAL VARIABLE

```
var PI = 3.1415;
```

LOCAL VARIABLE

```
var PI = 3.141593;
```

CLOSURE VARIABLE

```
var getArea = function(circle){
```

NESTED FUNCTION

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

```
var printStuffAboutCircleArray = function(circleArray) {
  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
    var c = circleArray[i];
    console.log(c.radius + " " + getArea(c));
  }
 return getArea;
}
```

REFERENCING ENVIRONMENT RETURNS THE NESTED FUNCTION..

EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

AND THE NESTED FUNCTION IS USED...

WHAT VALUE OF PI IS USED?!

```
var circle2 = new Circle(4);
var circle3 = new Circle(5);


var circleArray = [circle1,circle2,circle3];
var PI = 3.14159;

var printStuffAboutCircleArray = function(circleArray) {
  var PI = 3.14159;

  var getArea = function(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++){
    var c = circleArray[i];
    console.log(c.radius + "," + getArea(c));
  }
  return getArea;
}

var areaFunction = printStuffAboutCircleArray(circleArray);
areaFunction(circle1);


};
```

```
function Circle(r) {
  this.radius = r;
}
```

```
var PI = 3.14;
```

WHAT VALUE OF PI IS USED?!

GLOBAL VARIABLE?

```
window.onload = function(){
  var circle1 = new Circle(3);
  var circle2 = new Circle(4);
  var circle3 = new Circle(5);
```

```
var circleArray = [circle1,circle2,circle3];
```

```
var PI = 3.1415;
```

LOCAL VARIABLE?

```
var printStuffAboutCircleArray = function(circleArray) {
```

```
var PI = 3.141592;
```

CLOSURE VARIABLE?

```
var getArea = function(circle){
```

NESTED FUNCTION

```
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.radius * circle.radius;
  };
  for (var i = 0;i<circleArray.length;i++) {
     var c = circleArray[i];
```

THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

THE CLOSURE VARIABLE ALWAYS WINS
OVER EITHER LOCAL OR GLOBAL VARIABLES.

IF YOU ARE A JAVA OR C++ OR PYTHON
PROGRAMMER, THIS IS EXPLOSIVE.

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE
A FUNCTION WILL USE, OR WHERE IT CAME FROM.

THE CLOSURE VARIABLE ALWAYS WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.

IF YOU ARE A JAVA OR C++ OR PYTHON

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE
A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS - USING CLOSURES YOU CAN WRITE
FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT THEM
TO, NEVER MIND HOW THEY ARE USED.

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS - USING CLOSURES YOU CAN WRITE FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS - USING CLOSURES YOU CAN WRITE FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS - USING CLOSURES YOU CAN WRITE FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE
A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS -

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE A FUNCTION WILL USE, OR WHERE IT CAME FROM.

YOU CAN WRITE FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT THEM TO, NEVER MIND HOW THEY ARE USED.

YOU CAN NO LONGER BE SURE WHAT VALUE OF A VARIABLE
A FUNCTION WILL USE, OR WHERE IT CAME FROM.

BUT THE FLIP SIDE IS - USING CLOSURES **YOU CAN WRITE FUNCTIONS**
THAT WILL ALWAYS USE THE VALUES
YOU WANT THEM TO, **NEVER MIND**
HOW THEY ARE USED.

BUT THE FLIP SIDE IS - USING CLOSURES YOU CAN WRITE FUNCTIONS THAT WILL ALWAYS USE THE VALUES YOU WANT THEM TO, NEVER MIND HOW THEY ARE USED.