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

```
function Circle(r) {
this Addisol LE 42: THE CLOSURE VARIABLE ALWAYS
WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
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++) {</pre>
        var c = circleArray[i];
        console.log(c.radius + "," + getArea(c));
```

```
var PI = 3.14;
window.onload = function(){
 var circle1 = new Circle(3);
 var circle2 = new Circle(4);
     EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
    WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
 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++) {</pre>
      var c = circleArray[i];
      console.log(c.radices)SURE + ** TArea(c)); NESTED
                                      VARIABLES LOCAL TO THE
  return getArea;
 var areaFunction = printStuffAboutCircleArray(circleArray)
 areaFunction(circle1);
```

}:

```
var PI = 3.14;
window.onload = function(){
 var circle1 = new Circle(3);
 var circle2 = new Circle(4);
     EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
    WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
 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.radictosURE
                                      : VARIABLES LOCAL TO
  return getArea;
 var areaFunction = printStuffAboutCircleArray(cir
 areaFunction(circle1);
```

```
var PI = 3.14;
window.onload = function(){
 var circle1 = new Circle(3);
 var circle2 = new Circle(4);
     EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
    WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
 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++) {</pre>
      var c = circleArray[i];
      console.log(c.radictosVRE
                                 SAY WE HAVE A NESTED
                                       VARIABLES LOCAL TO THE
  return getArea;
                                          OUTERSCOPE
 var areaFunction = printStuffAboutCircleArray(cir
 areaFunction(circle1);
```

```
var PI = 3.14;
window.onload = function(){
 var circle1 = new Circle(3);
 var circle2 = new Circle(4);
     EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
   WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
 var PI = 3.1415:
 var printStuffAboutCircleArray = function(circleArray) {
   var PI = 3.24159;
   var getArea = function(circle){
     console.log("Inside the nested function getArea, PI = " + PI);
```

```
var PI = 3.X(159;

var getArea = full tion(circle){
    console.log("Inside the nested function getArea, PI = " + PI);
    return PI * circle.ladius * circle.radius;
};
for (var i = 0;i<circleArray.length;i++) {
    var c = circleArray[i];
    console log(c rad OLOSURE *** Area(c));
}
return getArea;
}
return getArea;
}
var areaFunction = printStuffAboutCircleArray ** CircleFERENCING**
areaFunction(circle1);</pre>
```

}:

```
function Circle(r) {
this Addisol LE 42: THE CLOSURE VARIABLE ALWAYS
WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
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++) {</pre>
        var c = circleArray[i];
        console.log(c.radius + "," + getArea(c));
```

```
function Circle(r) {
 EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
VA WINS DYER EITHER LOCAL OR GLOBAL VARIABLES.
```

```
var circle1 = new CiGLOBAL VARIABLE
var circle3 = new Circle(5);
var circleArray = [circle1,circle2,circle3];
var PI = 3.1415:
                      LOCAL VARIABLE
```

```
var getArea - function(dircMESTED FUNCTION
                                          getArea, PI = " + PI);
  return PI * circle.radius * circle.radius;
for (var i = 0;i<circleArray.length;i++) {</pre>
   var c = circleArray[i];
```

```
var PI = 3.14;
window.onload = function(){
 var circle1 = new Circle(3);
 EXAMPLE 42: THE CLOSURE VARIABLE ALWAYS
   WINS OVER EITHER LOCAL OR GLOBAL VARIABLES.
 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 EFFRENCING ENVIRONMENT
                         RETURNS THE NESTED FUNCTION...
  return getArea;
 var areaFunction = printStuffAboutCircleArray(circleArray);
 areaFunction(circle1);
```

```
var circle2 = new Circle(4);
var circle3 = new Circle(5);
var circleArray = [circle1,circle2,circle3];
var PEXAMPLE 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<circleAr AND THE NESTED FUNCTION</pre>
     var c = circleArray[i];
     console.log(c.radius + "," + getArea(tS; USE7...
 return getArea;
var areaFunction = practiffAbbutCitdleAnav(PiciArray);
areaFunction(circle1);
```

```
function Circle(r) {
 this.radius = r; WHAT VALUE OF PI IS
                              USED?!
var PI = 3.14;
 var circle1 = new cGLOBAL VARIABLE?
  var circle3 = new Circle(5);
  var circleArray = [circle1,circle2,circle3];
  var PI = 3.1415:
                         LOCAL VARIABLE?
          StuffAhoutCircleArray = function(circleArray) {
PI = 3.14 Lt0$UKE VARIABLE?
   var getArea - function(dircMESTED FUNCTION
      console.log("Inside the nested function getArea, PI = " + PI);
      return PI * circle.radius * circle.radius;
    for (var i = 0;i<circleArray.length;i++) {</pre>
```

var c = circleArray[i];

```
this radius = r; WHAT VALUE OF PI IS var PI = 3.14; USED?!
```

GLOBAL VARIABLE?

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

CLOSURE VARIABLE?

NESTED FUNCTION

return PI * circle.radius * circle.radius;
};

var c = circleArray[i];

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.

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

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

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

BUT THE FLIP SIDE IS -

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

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

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