EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

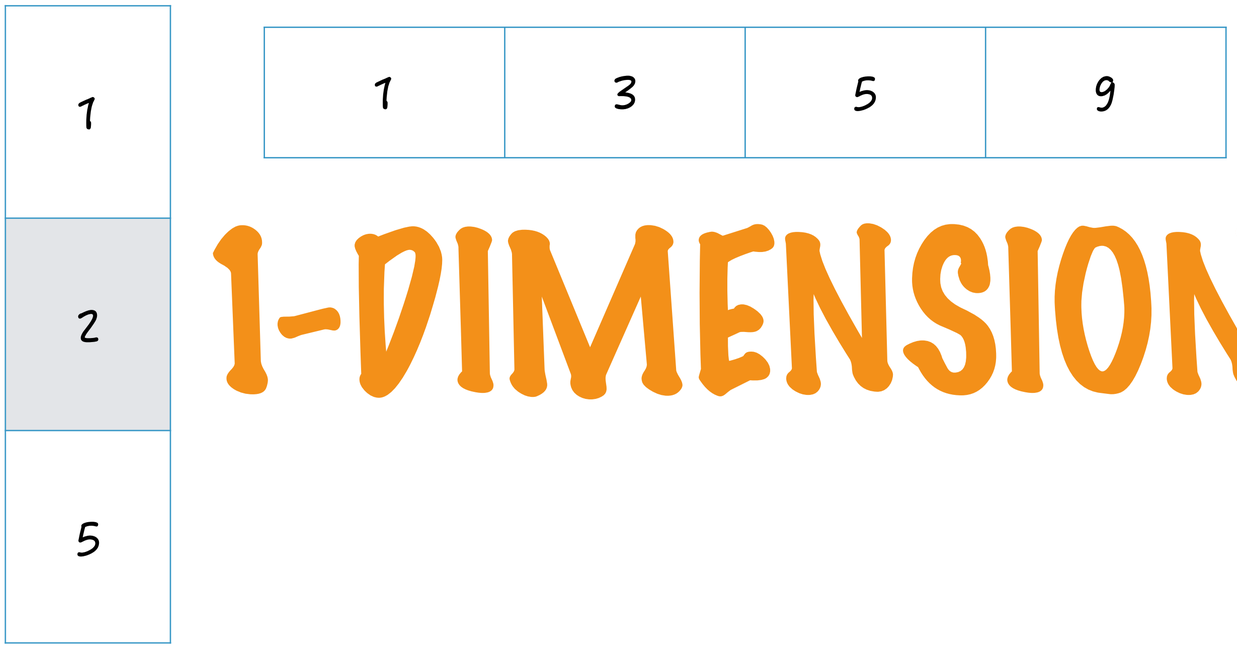
RECAP

DATA STRUCTURES IN R

ARRAY

LIKE A VECTOR, AN ARRAY CAN ONLY HAVE ELEMENTS OF THE SAME TYPE

AN ARRAY HAS IT'S ELEMENTS ARRANGED IN DIMENSIONS (ROW, COLUMNS ETC)



1-DIMENSIONAL ARRAYS

LIST
VECTOR
DATAFRAME
MATRIX

RECAP

DATA STRUCTURES IN R

ARRAY

LIKE A VECTOR, AN ARRAY CAN ONLY HAVE ELEMENTS OF THE SAME TYPE

AN ARRAY HAS IT'S ELEMENTS ARRANGED IN DIMENSIONS (ROW, COLUMNS ETC)

1-DIMENSIONAL ARRAYS

1
2
5

1	3	5	9

A 2-DIMENSIONAL ARRAY

IS A STACK OF 1 DIMENSIONAL ARRAYS

1	3	5	9
2	7	4	3
5	8	9	7

LIST
VECTOR
DATAFRAME
MATRIX

DATA STRUCTURES IN R

RECAP

ARRAY

LIKE A VECTOR, AN ARRAY CAN ONLY HAVE ELEMENTS OF THE SAME TYPE

AN ARRAY HAS IT'S ELEMENTS ARRANGED IN DIMENSIONS (ROW, COLUMNS ETC)

1-DIMENSIONAL ARRAYS

1
2
5

1	3	5	9

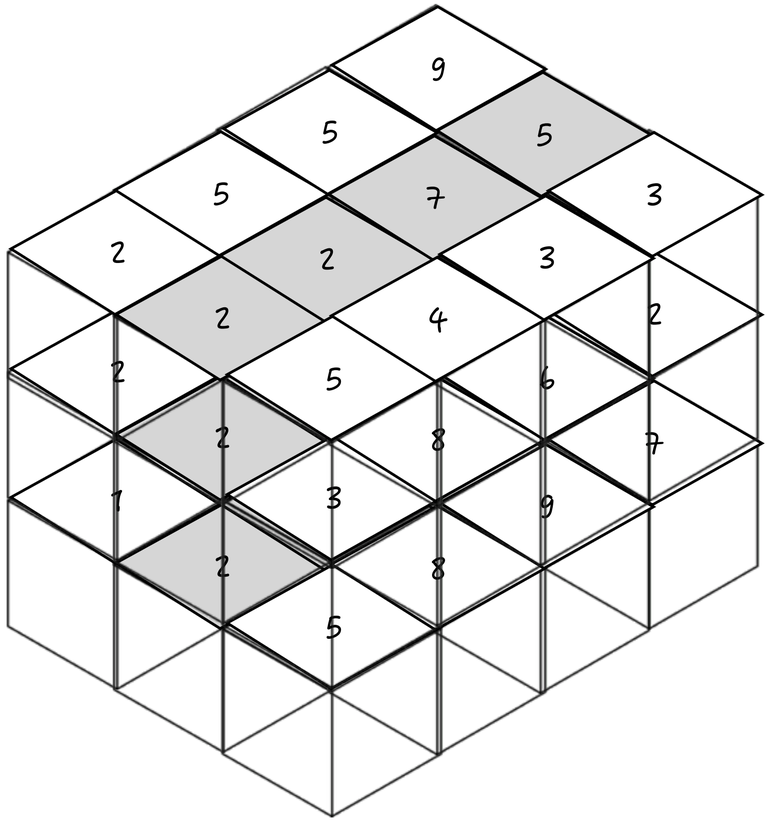
A 2-DIMENSIONAL ARRAY

IS A STACK OF 1 DIMENSIONAL ARRAYS

1	3	5	9
2	7	4	3
5	8	9	7

A 3-DIMENSIONAL ARRAY

IS A STACK OF 2 DIMENSIONAL ARRAYS



LIST
VECTOR
DATAFRAME
MATRIX

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

AN ARRAY IS A VECTOR, WITH AN ADDITIONAL ATTRIBUTE **DIMENSIONS**

A VECTOR IS A FLAT STRUCTURE, IT HAS NO ROWS, COLUMNS OR OTHER DIMENSIONS

```
myVector <- 1:10
myVector
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

BY ASSIGNING DIMENSIONS TO A VECTOR
YOU CAN TURN A VECTOR INTO AN ARRAY

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
[1]  1  2  3  4  5  6  7  8  9 10
```

```
myArray <- myVector
dim(myArray) <- c(5,2)
```

2 - DIMENSIONAL ARRAY

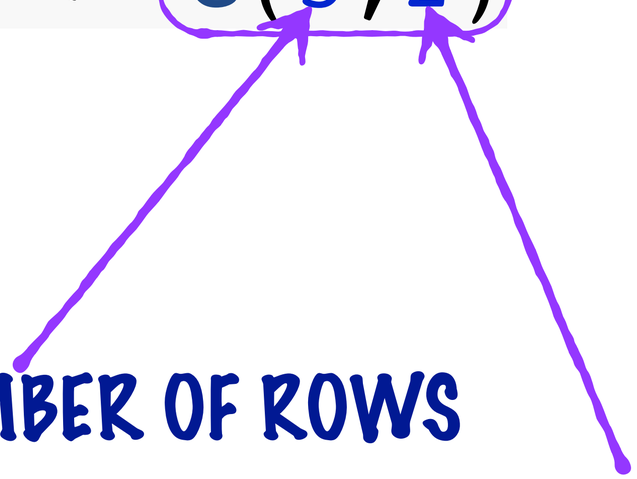
THE DIMENSIONS OF AN ARRAY ARE EXPRESSED BY AN INTEGER VECTOR

THE LENGTH OF THE DIMENSION VECTOR TELLS US THE NUMBER OF DIMENSIONS

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
[1]  1  2  3  4  5  6  7  8  9 10
```

```
myArray <- myVector
dim(myArray) <- c(5,2)
```



THE FIRST DIMENSION IS THE NUMBER OF ROWS

THE SECOND DIMENSION IS THE NUMBER OF COLUMNS

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
[1]  1  2  3  4  5  6  7  8  9 10
```

```
myArray <- myVector
dim(myArray) <- c(5,2)
```

AN ARRAY CAN HAVE AS MANY DIMENSIONS AS YOU LIKE...AS LONG AS

THE PRODUCT OF ALL THE DIMENSIONS = NUMBER OF ELEMENTS IN THE ARRAY

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
[1]  1  2  3  4  5  6  7  8  9 10
```

```
myArray <- myVector
dim(myArray) <- c(5,2)
```

WHEN YOU ASSIGN DIMENSIONS TO A VECTOR

THE ELEMENTS OF THE VECTOR ARE ARRANGED ALONG THE DIMENSIONS

THE ARRAY IS FILLED ALONG THE ROWS FIRST, THEN THE COLUMNS, AND SO ON

5 ROWS

2 COLUMNS

	,1	,2
1,	1	6
2,	2	7
3,	3	8
4,	4	9
5,	5	10

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
myArray <- myVector
dim(myArray) <- c(5,2)
myArray
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
myVector <- 1:10
myVector
 [1]  1  2  3  4  5  6  7  8  9 10
myArray <- myVector
dim(myArray) <- c(5,2)
myArray
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

THESE 2 COMMANDS TURNED THE VECTOR myVector INTO AN ARRAY

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
```

THE ARRAY FUNCTION DOES EXACTLY THIS IN JUST ONE STEP

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
```

THE VECTOR TO BE CONVERTED TO AN ARRAY

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
```

DIMENSIONS OF THE ARRAY

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
```

THIS IS A 3 DIMENSIONAL ARRAY

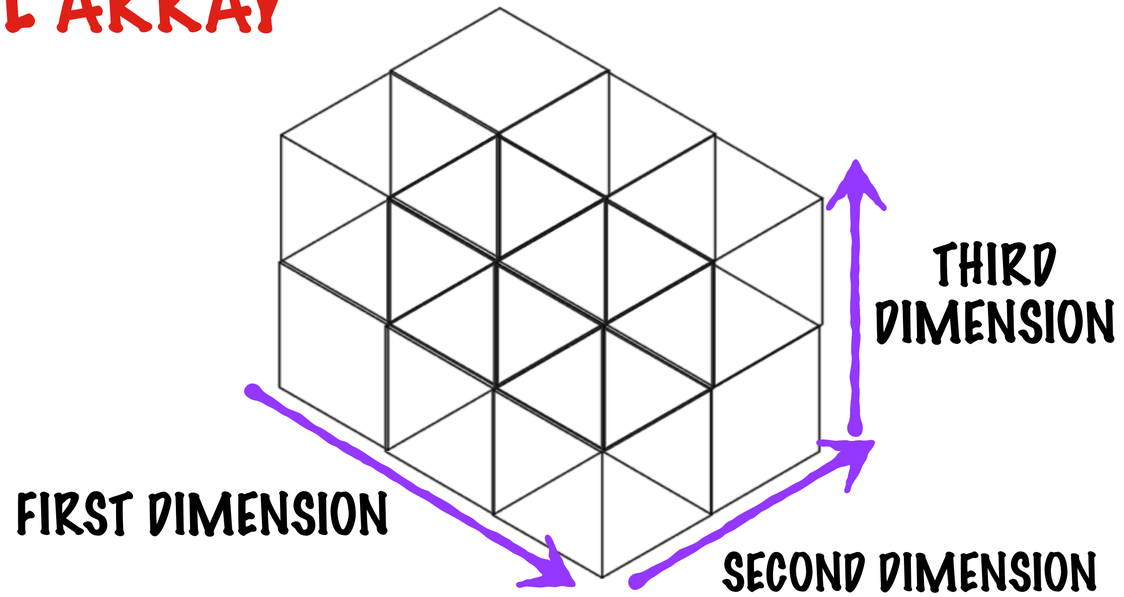
IT WILL FIRST BE FILLED ALONG THE ROWS, THEN COLUMNS
AND THEN THE 3RD DIMENSION

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
```

THIS IS A 3 DIMENSIONAL ARRAY

IT WILL FIRST BE FILLED ALONG THE ROWS, THEN COLUMNS AND THEN THE 3RD DIMENSION



EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

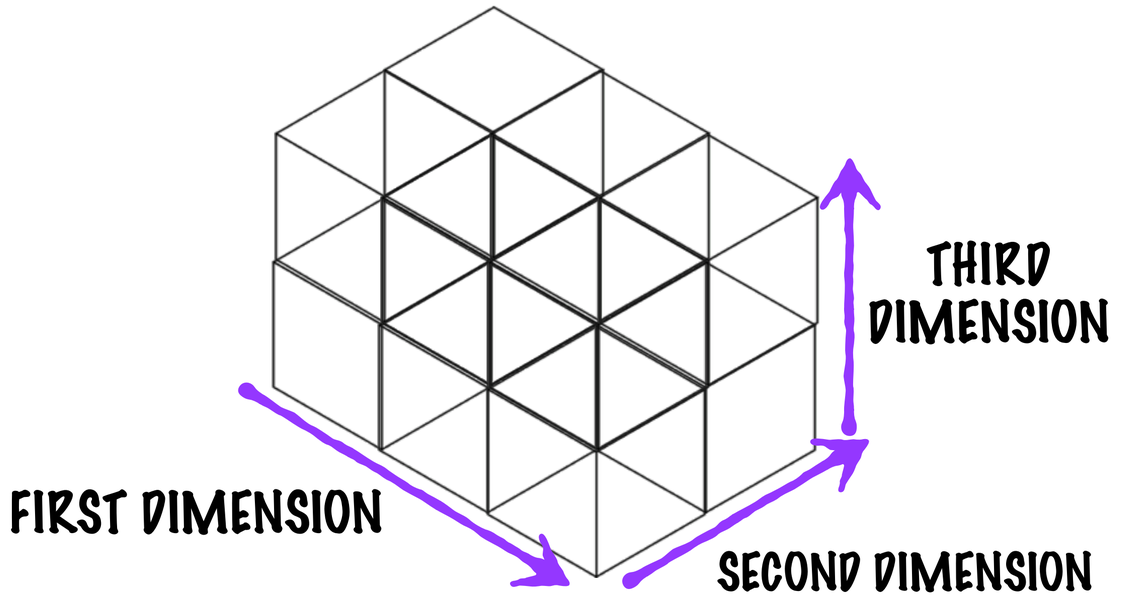
```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```



EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

BUT THE VECTOR ONLY HAS 2 ELEMENTS

THIS ARRAY EXPECTS TO HAVE 6 ELEMENTS

```
thirdArray <- array(c(1,0) , dim = c(2,3))
```

RECYCLING TO THE RESCUE :)

WHAT IF THE VECTOR IS YOU ARE CONVERTING IS TOO SHORT TO FILL THE ARRAY OF SPECIFIED DIMENSIONS?

EXAMPLE 20: CREATING AN ARRAY FROM A VECTOR

```
thirdArray <- array(c(1,0) , dim = c(2,3))
thirdArray
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    0    0    0
```

RECYCLING TO THE RESCUE :)

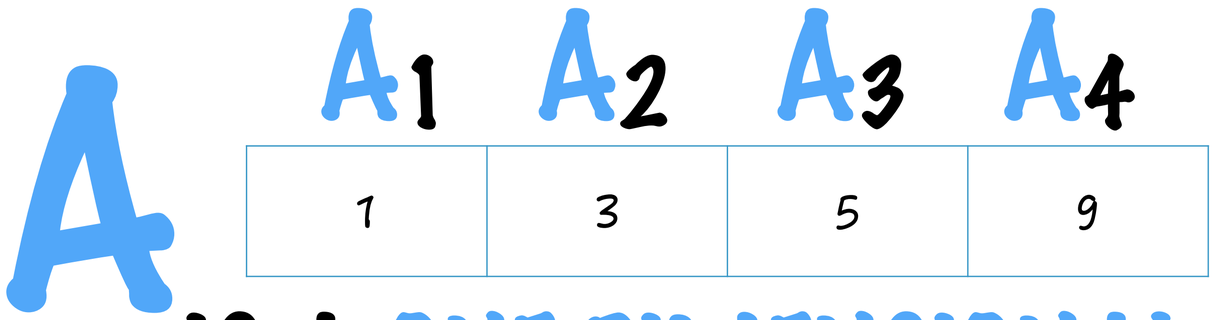
A COOL USE OF RECYCLING IS TO CREATE CONSTANT ARRAYS

```
arrayOfZeros <- array(0, dim = c(100,100))
```

EXAMPLE 21: INDEXING AN ARRAY

EXAMPLE 21: INDEXING AN ARRAY

TO INDEX AN ARRAY YOU NEED AS MANY SUBSCRIPTS AS THERE ARE DIMENSIONS



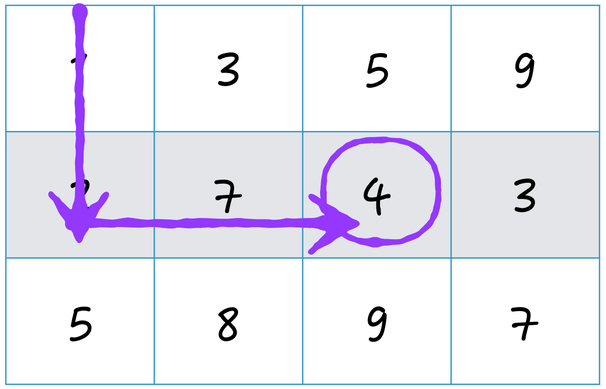
A IS A ONE DIMENSIONAL ARRAY

TO INDEX AN ELEMENT IN A YOU NEED ONLY 1 SUBSCRIPT

EXAMPLE 21: INDEXING AN ARRAY

TO INDEX AN ARRAY YOU NEED AS MANY SUBSCRIPTS AS THERE ARE DIMENSIONS

B IS A 2 DIMENSIONAL ARRAY



B_{23} IS THE ELEMENT IN THE SECOND ROW AND THE THIRD COLUMN

TO INDEX AN ELEMENT IN B YOU NEED 2 SUBSCRIPTS

EXAMPLE 21: INDEXING AN ARRAY

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

anotherArray IS A 3-DIMENSIONAL ARRAY

```
anotherArray[2,2,1]
```

TO INDEX AN ELEMENT IN anotherArray YOU NEED 3 SUBSCRIPTS

EXAMPLE 21: INDEXING AN ARRAY

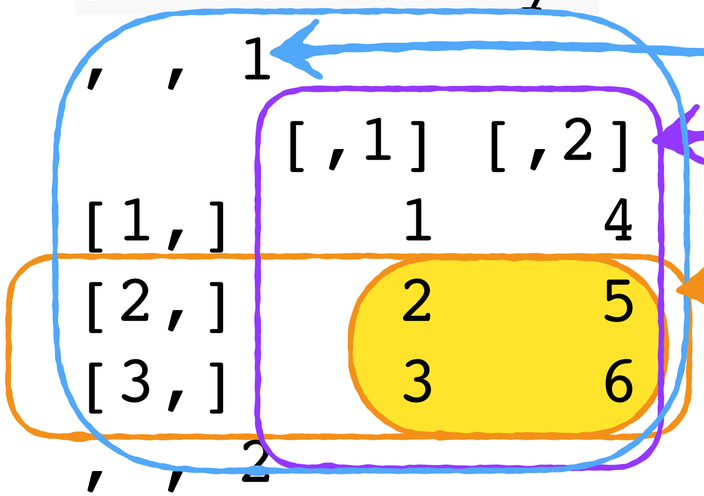
```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

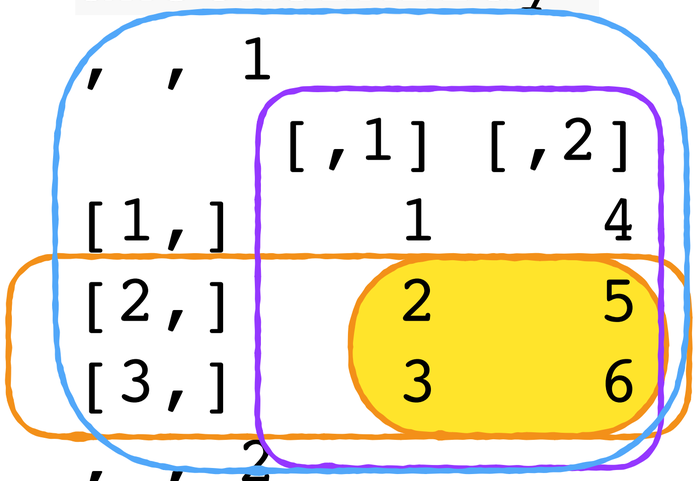


```
anotherArray[2:3,1:2,1]
```

TO INDEX A SUBSECTION OF AN ARRAY YOU CAN USE A VECTOR IN PLACE OF EACH SUBSCRIPT

EXAMPLE 21: INDEXING AN ARRAY

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```



```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

```
anotherArray[2:3,1:2,1]
```

```
anotherArray[2:3,,1]
```

IF YOU LEAVE A SUBSCRIPT BLANK, THAT IS EQUIVALENT TO SELECTING ALL THE VALUES IN THE CORRESPONDING DIMENSION

EXAMPLE 21: INDEXING AN ARRAY

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

```
indexArray <- array (c(1:2), dim=c(2,3))
indexArray
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

```
anotherArray[indexArray]
```

YOU CAN ALSO USE A 2 DIMENSIONAL ARRAY (A MATRIX) TO INDEX A SPECIFIC SET OF ELEMENTS

EXAMPLE 21: INDEXING AN ARRAY

```
anotherArray <- array(c(1:12), dim = c(3,2,2))
anotherArray
```

```
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
```

```
indexArray <- array (c(1:2), dim=c(2,3))
indexArray
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

```
anotherArray[indexArray]
[1]  1 11
```

EACH ROW IN THE INDEX ARRAY REFERS TO ONE ELEMENT THAT SHOULD BE SELECTED

EXAMPLE 22: OPERATIONS BETWEEN 2 ARRAYS

EXAMPLE 22: OPERATIONS BETWEEN 2 ARRAYS

```
a <- array(1:6, dim = c(2,3))
b <- array(7:12, dim = c(2,3))
a * b
```

FOR AN OPERATION INVOLVING 2 ARRAYS, BOTH ARRAYS MUST BE OF THE SAME DIMENSION

EXAMPLE 22: OPERATIONS BETWEEN 2 ARRAYS

```r
a <- array(1:6, dim = c(2,3))
b <- array(7:12, dim = c(2,3))
a * b
```

```
     [,1] [,2] [,3]
[1,]    7   27   55
[2,]   16   40   72
```

THE OPERATION IS PERFORMED ELEMENT-WISE I.E. BETWEEN CORRESPONDING ELEMENTS OF BOTH ARRAYS

THE RESULT IS AN ARRAY WITH SAME DIMENSIONS AS THE INPUT ARRAYS

EXAMPLE 22: OPERATIONS BETWEEN 2 ARRAYS

```
a <- array(1:6, dim = c(2,3))
b <- array(7:12, dim = c(2,3))
a * b
```

```
[,1] [,2] [,3]
[1,]    7   27   55
[2,]   16   40   72
```

```
c <- array(13:18, dim = c(3,2))
a - c
```

```
Error in a - c : non-conformable arrays
```

IF THE ARRAYS ARE OF DIFFERENT DIMENSIONS AN ERROR WILL BE THROWN

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

FOR AN OPERATION BETWEEN
AN ARRAY AND A VECTOR,

```
a * b
```

```
a - c
```

LENGTH OF THE VECTOR
MUST BE <=
SIZE OF THE ARRAY

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

```
a
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
sameLength <- 1:6
shorterVector <- 1:2
longerVector <- 1:10
```

a IS AN ARRAY OF SIZE 6

I.E. IT HAS 6 ELEMENTS

LET'S LOOK AT 3 CASES

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

```
a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
sameLength <- 1:6
shorterVector <- 1:2
longerVector <- 1:10
a + sameLength
[,1] [,2] [,3]
[1,]    2    6   10
[2,]    4    8   12
```

```
a * b
```

```
a - c
```

IF THE VECTOR'S LENGTH IS EQUAL TO THE ARRAY SIZE, THEN THE OPERATION IS APPLIED ELEMENT-WISE

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

```
a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
sameLength <- 1:6
shorterVector <- 1:2
longerVector <- 1:10

a - shorterVector
     [,1] [,2] [,3]
[1,]    0    2    4
[2,]    0    2    4
```

a

1	3	5
2	4	6

shorterVector

1	1	1
2	2	2

IF THE VECTOR IS SHORTER, IT IS RECYCLED AS MANY TIMES AS NEEDED

EXAMPLE 23: OPERATIONS BETWEEN AN ARRAY AND A VECTOR

```
a
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
sameLength <- 1:6
shorterVector <- 1:2
longerVector <- 1:10
```

```
a / longerVector
```

```
dims [product 6] do not match the length of object [10]
```

IF THE VECTOR IS LONGER, AN ERROR IS THROWN

EXAMPLE 24 : OUTER PRODUCTS

EXAMPLE 24 : OUTER PRODUCTS

THE OUTER PRODUCT OF 2 ARRAYS IS AN ARRAY WITH ELEMENT-WISE PRODUCTS OF EVERY PAIR OF ELEMENTS FROM BOTH ARRAYS

LET'S TAKE AN EXAMPLE

ONE OR BOTH OF THE ARRAYS COULD ALSO BE VECTORS
TREATING VECTORS AS 1 DIMENSIONAL ARRAYS

EXAMPLE 24 : OUTER PRODUCTS

ONE OR BOTH OF THE ARRAYS COULD ALSO BE VECTORS

TREATING VECTORS AS 1 DIMENSIONAL ARRAYS

LET'S TAKE AN EXAMPLE

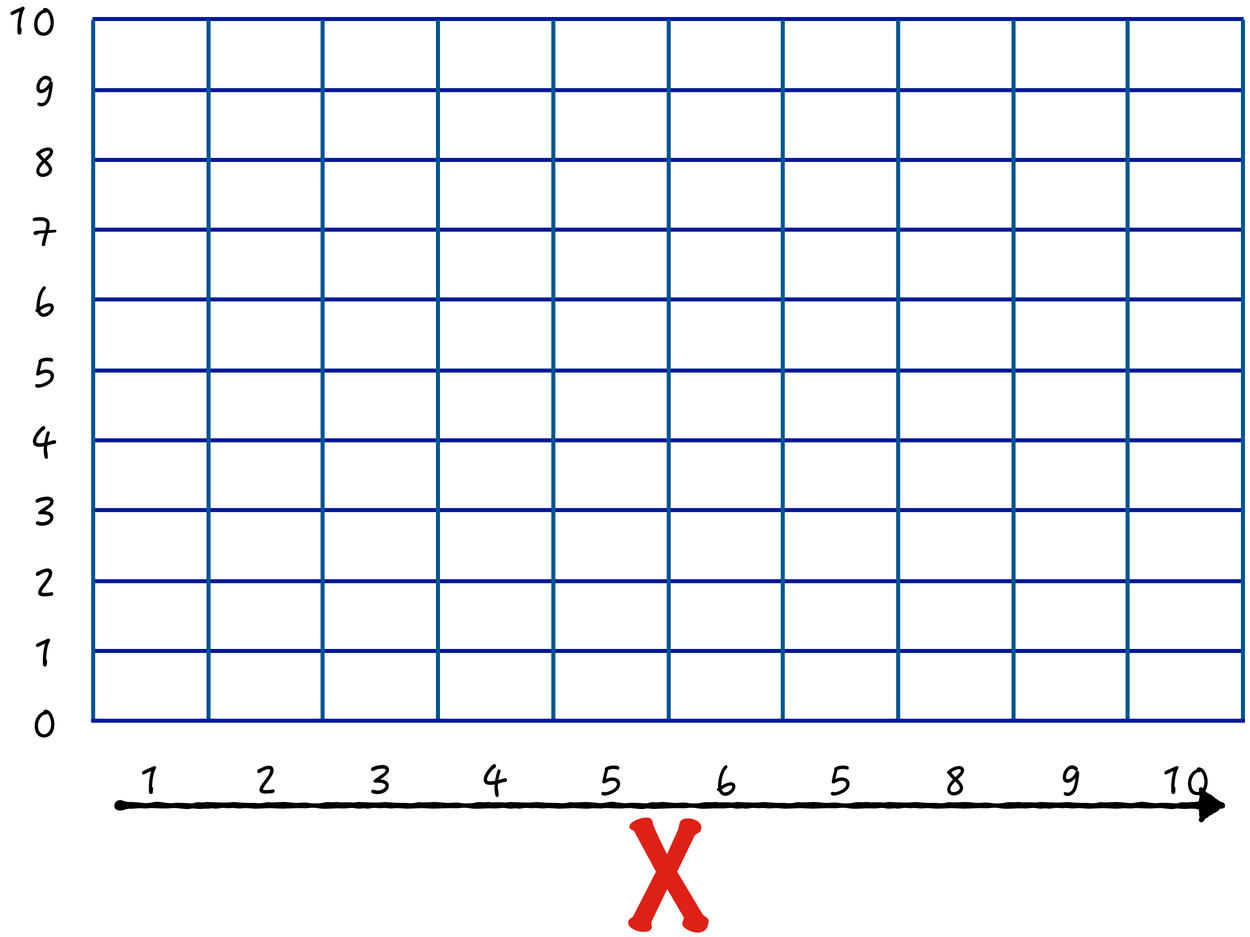
EXAMPLE 24 : OUTER PRODUCTS

LET'S SAY WE HAVE A 10x10 GRID

```
x <- array(1:10, dim = 10)
y <- 1:10
```

X AND Y REPRESENT THE GRID

(WE'VE MADE ONE OF THEM A VECTOR JUST FOR KICKS)



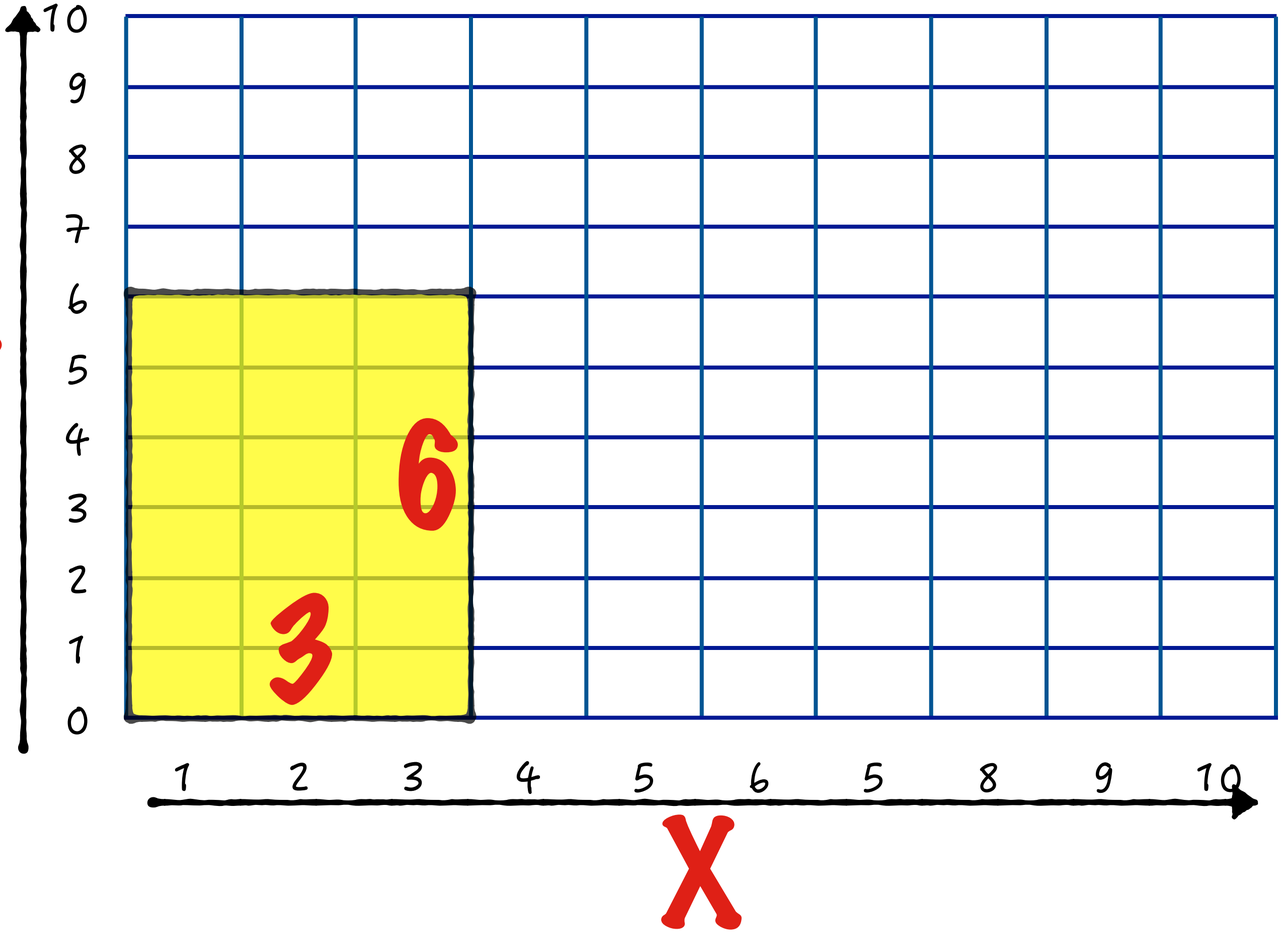
EXAMPLE 24 : OUTER PRODUCTS

LET'S SAY WE HAVE A 10x10 GRID

```
x <- array(1:10, dim = 10)
y <- 1:10
```

WE CAN DRAW RECTANGLES WITH ONE CORNER AT THE ORIGIN AND

COMPUTE THE AREA



AREA = 18

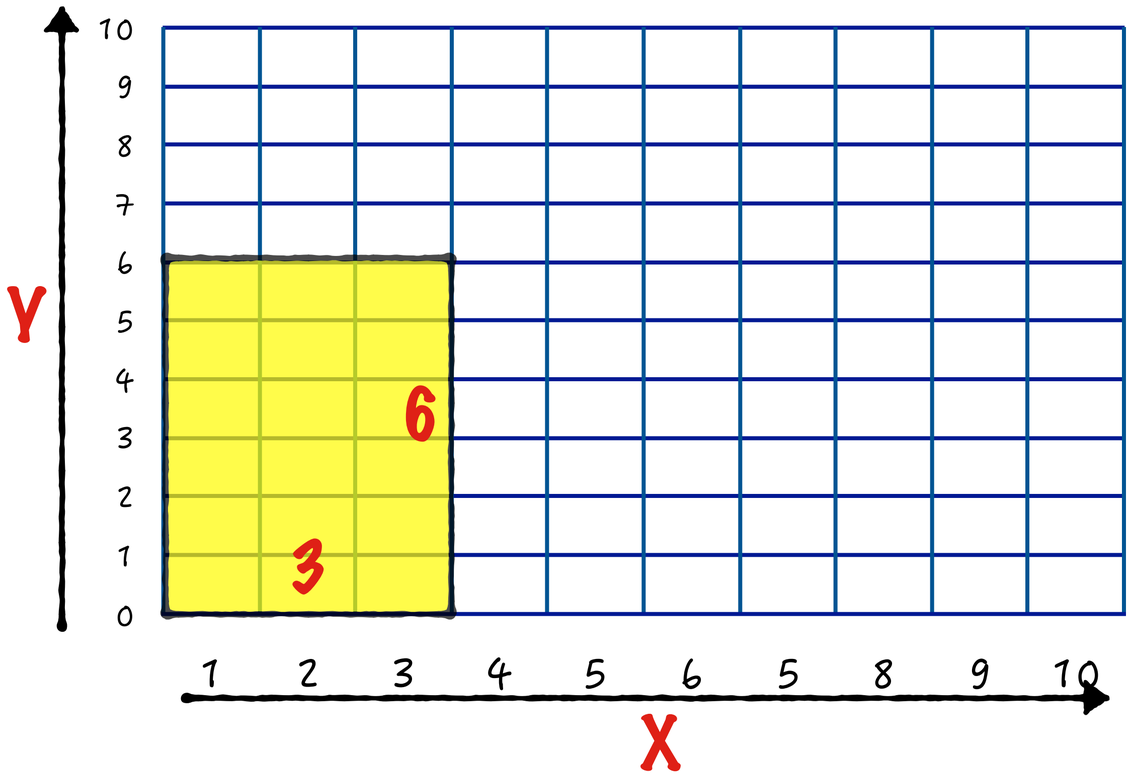
EXAMPLE 24 : OUTER PRODUCTS

```
x <- array(1:10, dim = 10)
y <- 1:10
```

HOW DO WE COMPUTE THE AREAS OF EVERY POSSIBLE SUCH RECTANGLE?

OUTER PRODUCT OF X AND Y

```
x %o% y
```

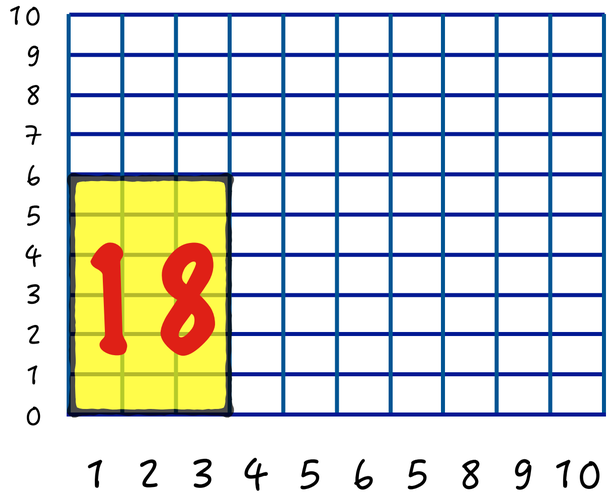


EXAMPLE 24 : OUTER PRODUCTS

LET'S SAY WE HAVE A 10x10 GRID

```
x <- array(1:10, dim = 10)
y <- 1:10
x %o% y
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    2    3    4    5    6    7    8    9    10
 [2,]    2    4    6    8   10   12   14   16   18    20
 [3,]    3    6    9   12   15   18   21   24   27    30
 [4,]    4    8   12   16   20   24   28   32   36    40
 [5,]    5   10   15   20   25   30   35   40   45    50
 [6,]    6   12   18   24   30   36   42   48   54    60
 [7,]    7   14   21   28   35   42   49   56   63    70
 [8,]    8   16   24   32   40   48   56   64   72    80
 [9,]    9   18   27   36   45   54   63   72   81    90
[10,]   10   20   30   40   50   60   70   80   90   100
```



THE OUTER PRODUCT IS A 10x10 ARRAY

EACH ELEMENT IS THE AREA OF ONE SUCH RECTANGLE

EXAMPLE 24 : OUTER PRODUCTS

AB IS THE OUTER PRODUCT OF 2 ARRAYS A AND B

```
A <- array(1:18, dim = c(3,2,3))
B <- array(19:36, dim = c(2,3,3))
AB <- A %o% B
dim(AB)
[1] 3 2 3 2 3 3
```

DIMENSIONS OF AB ARE FOUND BY CONCATENATING THE DIMENSIONS OF A AND B

EXAMPLE 24 : OUTER PRODUCTS

```
A <- array(1:18, dim = c(3,2,3))
B <- array(19:36, dim = c(2,3,3))
AB <- A %o% B
dim(AB)
[1] 3 2 3 2 3 3
```

```
AB[1,2,1,2,1,1]
[1] 80
A[1,2,1] * B[2,1,1]
[1] 80
```

EACH ELEMENT IN AB IS THE PRODUCT OF CORRESPONDING ELEMENTS IN A AND B

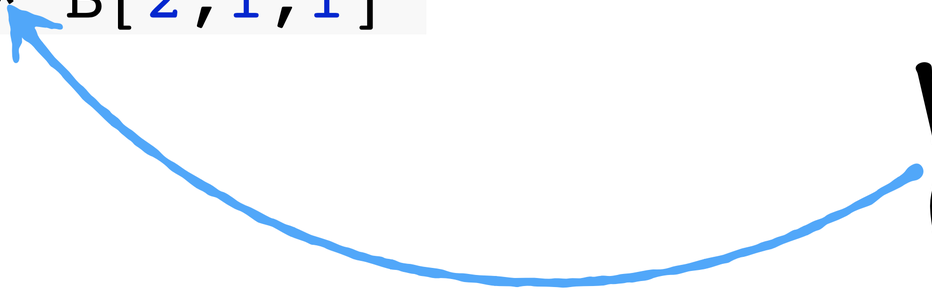
EXAMPLE 24 : OUTER PRODUCTS

```
x <- array(1:10, dim = 10)
y <- 1:10
x %o% y
A <- array(1:18, dim = c(3,2,3))
B <- array(19:36, dim = c(2,3,3))
AB <- A %o% B
dim(AB)
[1] 3 2 3 2 3 3
AB[1,2,1,2,1,1]
[1] 80
A[1,2,1] * B[2,1,1]
[1] 80
```

EACH ELEMENT IN AB IS THE PRODUCT OF ONE PAIR OF ELEMENTS IN A AND B

EXAMPLE 24 : OUTER PRODUCTS

```
x <- array(1:10, dim = 10)
y <- 1:10
x %o% y
A <- array(1:18, dim = c(3,2,3))
B <- array(19:36, dim = c(2,3,3))
AB <- A %o% B
dim(AB)
[1] 3 2 3 2 3 3
AB[1,2,1,2,1,1]
[1] 80
A[1,2,1] * B[2,1,1]
[1] 80
```



YOU CAN REPLACE THE * WITH ANY OTHER OPERATOR OR FUNCTION USING OUTER()

EXAMPLE 24 : OUTER PRODUCTS

```
x <- array(1:10, dim = 10)
y <- 1:10
x %o% y
A <- array(1:18, dim = c(3,2,3))
B <- array(19:36, dim = c(2,3,3))
AB <- A %o% B
dim(AB)
[1] 3 2 3 2 3 3
AB[1,2,1,2,1,1]
[1] 80
A[1,2,1] * B[2,1,1]
[1] 80
```

```
outerSumAB <- outer(A, B, "+")
```

YOU CAN REPLACE THE * WITH ANY OTHER OPERATOR OR FUNCTION USING OUTER()