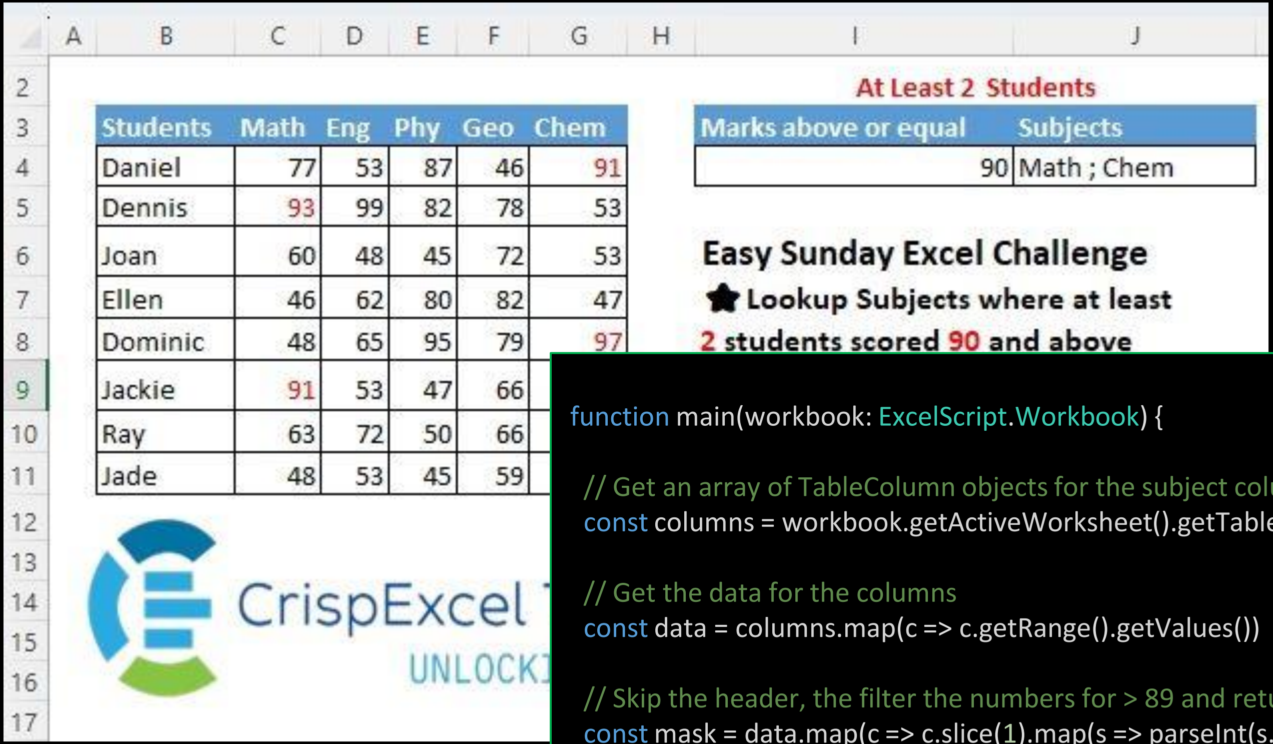
First steps with Office Scripts: Solving a data challenge - (follow-up)

Owen Price
flexyourdata.com, @flexyourdata

I posted an Office Scripts solution to a data challenge from Crispo Mwangi



Students	Math	Eng	Phy	Geo	Chem
Daniel	77	53	87	46	91
Dennis	93	99	82	78	53
Joan	60	48	45	72	53
Ellen	46	62	80	82	47
Dominic	48	65	95	79	97
Jackie	91	53	47	66	
Ray	63	72	50	66	
Jade	48	53	45	59	

At Least 2 Students

Marks above or equal	Subjects
90	Math ; Chem

Easy Sunday Excel Challenge

Lookup Subjects where at least 2 students scored 90 and above

```
function main(workbook: ExcelScript.Workbook) {

  // Get an array of TableColumn objects for the subject columns
  const columns = workbook.getActiveWorksheet().getTable("Marks").getColumns().slice(1)

  // Get the data for the columns
  const data = columns.map(c => c.getRange().getValues())

  // Skip the header, the filter the numbers for > 89 and return true if more than one
  const mask = data.map(c => c.slice(1).map(s => parseInt(s.toString())).filter(v => v > 89).length > 1 )

  // Filter the data for where the mask is true, then slice for the header
  const headers = data.filter((_, i) => mask[i]).map(c => c.slice(0, 1))

// join and print the output
  console.log(headers.join(" ; "))
}
```

Owen Price
flexyourdata.com, @flexyourdata

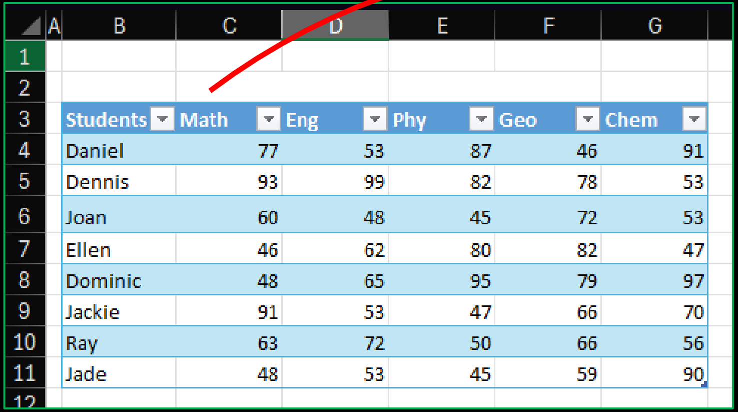
But I wasn’t happy with this part of the code

```
function main(workbook: ExcelScript.Workbook) {

  // Get an array of TableColumn objects for the subject columns
  const columns = workbook.getActiveWorksheet().getTable("Marks").getColumns().slice(1)

  // Get the data for the columns
  const data = columns.map(c => c.getRange().getValues())

  // Skip the header, the filter the numbers for > 89 and return true if more than one
  const mask = data.map(c => c.slice(1).map(s => parseInt(s.toString())).filter(v => v > 89).length > 1 )

  // Filter the data for where the mask is true, then slice for the header
  const headers = data.filter((_, i) => mask[i]).map(c => c.slice(0, 1))

// join and print the output
  console.log(headers.join(" ; "))
}
```

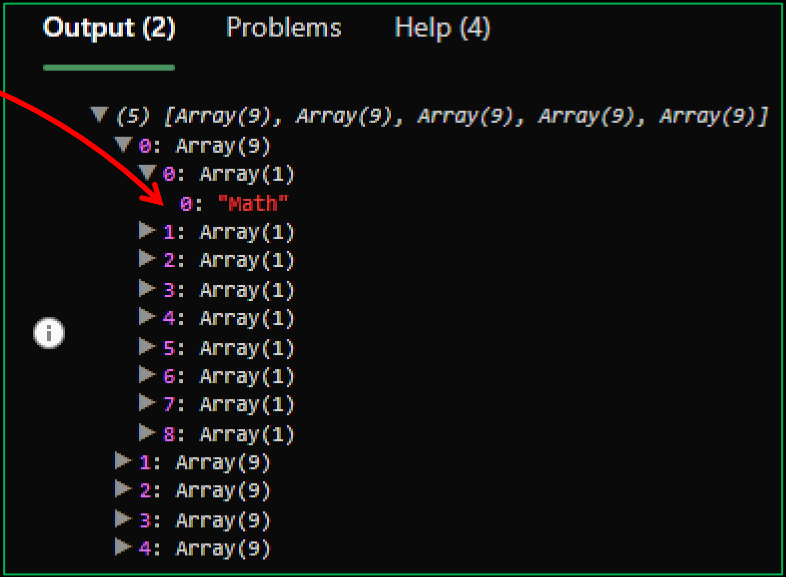
Specifically, why did I have to convert to string, then convert to integer, and why did I have to use .map to do it, all so I could filter the array for numbers greater than 89?

Some background on array types and depth

```
function main(workbook: ExcelScript.Workbook) {
  // Get an array of TableColumn objects for the subject columns
  const columns = workbook.getActiveWorksheet().getTable("Marks").getColumns().slice(1)

  /* columns is an array of TableColumn, i.e. TableColumn[]
   getValues() always returns (string | number | boolean)[][] - a 2D array of mixed-type values
   since getValues() is applied on each TableColumn, this results in 'data' being
   (string | number | boolean)[][][] : a 3D array of mixed-type values
   The first level is each column, the second level is the rows in that column
   and the third level is the values in the rows in that column,
   even though the third level is always length 1 */
  const data = columns.map(c => c.getRange().getValues())
  console.log(data)
```



	A	B	C	D	E	F	G
1							
2							
3		Students	Math	Eng	Phy	Geo	Chem
4		Daniel	77	53	87	46	91
5		Dennis	93	99	82	78	53
6		Joan	60	48	45	72	53
7		Ellen	46	62	80	82	47
8		Dominic	48	65	95	79	97
9		Jackie	91	53	47	66	70
10		Ray	63	72	50	66	56
11		Jade	48	53	45	59	90
12							

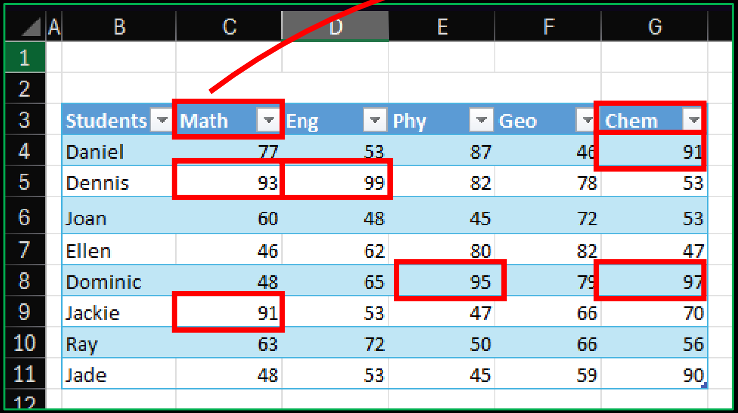


```
▼ (5) [Array(9), Array(9), Array(9), Array(9), Array(9)]
  ▼ 0: Array(9)
    ▼ 0: Array(1)
        0: "Math"
    ▶ 1: Array(1)
    ▶ 2: Array(1)
    ▶ 3: Array(1)
    ▶ 4: Array(1)
    ▶ 5: Array(1)
    ▶ 6: Array(1)
    ▶ 7: Array(1)
    ▶ 8: Array(1)
  ▶ 1: Array(9)
  ▶ 2: Array(9)
  ▶ 3: Array(9)
  ▶ 4: Array(9)
```

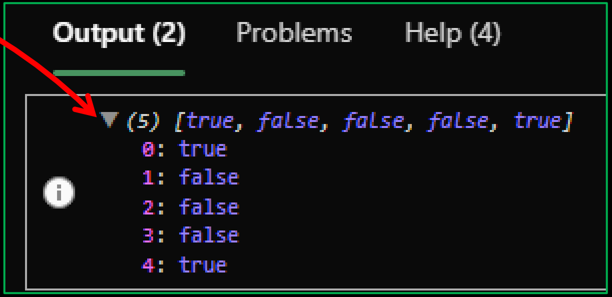
Owen Price
flexyourdata.com, @flexyourdata

Create a boolean array of columns that meet the criteria – original version

```
/* In the original version of the mask line, data.map(c => etc) is applying the logic 'etc' to each column c.
c.slice removes the header, but the remaining items are still of type (string | number | boolean)[]
.map(s => parseInt(s.toString())) is taking each value in each column, converting it to a string
then parsing that string into an integer. After that, the array of integers is filtered as shown */
const mask = data.map(c => c.slice(1).map(s => parseInt(s.toString())).filter(v => v > 89).length > 1 )
console.log(mask)
```



Students	Math	Eng	Phy	Geo	Chem
Daniel	77	53	87	46	91
Dennis	93	99	82	78	53
Joan	60	48	45	72	53
Ellen	46	62	80	82	47
Dominic	48	65	95	79	97
Jackie	91	53	47	66	70
Ray	63	72	50	66	56
Jade	48	53	45	59	90

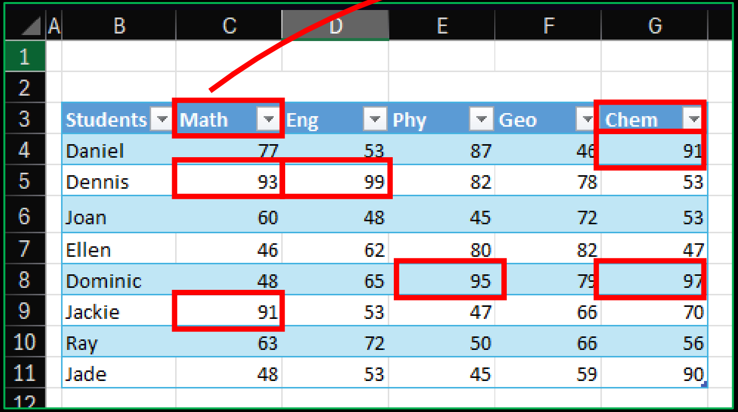


```
▼ (5) [true, false, false, false, true]
    0: true
    1: false
    2: false
    3: false
    4: true
```

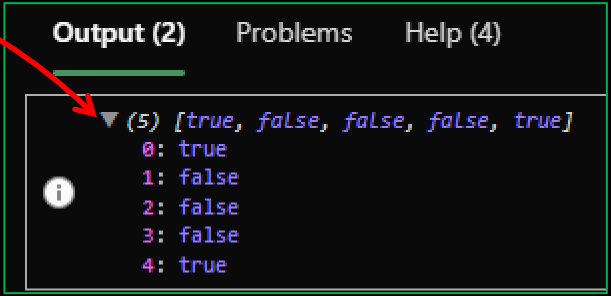
Owen Price
flexyourdata.com, @flexyourdata

Create a boolean array of columns that meet the criteria – second version

```
/* In this second version, the parseInt(s.toString()) step is replaced with Number(s)
This works because the argument type for the number function is 'any', meaning we can safely
pass a value of type (string | number | boolean) to it. In contrast the argument type of
parseInt is string, meaning s first had to be converted to a string with toString() */
const mask = data.map(c => c.slice(1).map(s => Number(s)).filter(v => v > 89).length > 1)
console.log(mask)
```



	A	B	C	D	E	F	G
1							
2							
3		Students	Math	Eng	Phy	Geo	Chem
4		Daniel	77	53	87	46	91
5		Dennis	93	99	82	78	53
6		Joan	60	48	45	72	53
7		Ellen	46	62	80	82	47
8		Dominic	48	65	95	79	97
9		Jackie	91	53	47	66	70
10		Ray	63	72	50	66	56
11		Jade	48	53	45	59	90
12							



```
▼ (5) [true, false, false, false, true]
    0: true
    1: false
    2: false
    3: false
    4: true
```

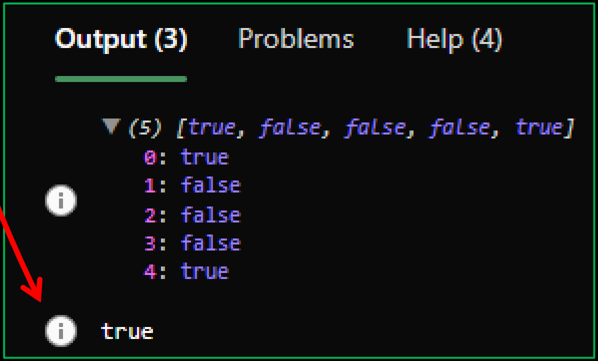
Owen Price
flexyourdata.com, @flexyourdata

Create a boolean array of columns that meet the criteria – third version

```
/* In this third version, map is removed, and Number(v) is used inside the filter function.
Both map and filter have to iterate through the rows, so removing map means more efficient code. */
const mask = data.map(c => c.slice(1).filter(v => Number(v) > 89).length > 1)
console.log(mask)
console.log(Number(["123"]) === Number("123"))
```

v is of type (string | number | boolean)[] – it's an array, but as we saw earlier: it always has only one element. JavaScript allows this Number() function to convert an array of one element to a single value. So, for length 1 arrays:

Number(v[0]) === Number(v)



```
▼ (5) [true, false, false, false, true]
    0: true
    1: false
    2: false
    3: false
    4: true

true
```

Owen Price
flexyourdata.com, @flexyourdata

Takeaways:

1. The getValues() method returns a 2D array of (string | number | boolean)
2. To convert numeric cells of mixed-type to numbers in an array, use the Number() function
3. Office Scripts will often coerce single-element arrays to their primitive value equivalents:
   `Number(["123"]) === Number("123"))`
4. Revisiting your assumptions can lead to better code!