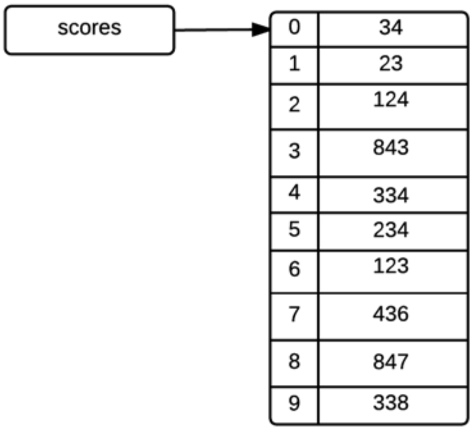
R 20

Arrays - Reading

If we had a two-player game with two scores we could store these as two **variables**, score1 and score2. There would be a problem if we wanted to have hundreds of players though, as we would need to create a variable for every one of them.

Arrays allow programmers to store a set of values under one **identifier**. Each value will have an **index number**. Index numbers usually start at 0 and this is the case for C, C++, C#, Java and Python.

```
scores ← [34, 23, 124, 843,
334, 234, 123, 436, 847, 338]
```



0	34
1	23
2	124
3	843
4	334
5	234
6	123
7	436
8	847
9	338

The pseudocode above will create an array with 10 **elements** (spaces). Each element will then be initialised with the values given. A diagram of how this is stored in memory is shown above on the right. To alter one element, for example the first element, we can use the following code:
`scores[0] ← 23`

Arrays are often used together with **FOR loops** to access them. The code to the right shows how to print out all the scores. Notice how length(scores) is used rather than the number 10. This means that if we add another score the program will still work.

```
FOR i ← 0 TO 9
    OUTPUT scores[i]
ENDFOR
```

Arrays make it easier to work on each element. We only need to tell the computer what to do to one element and we can then use a FOR loop to do the rest. The example on the right will multiply all the scores by 100.

```
FOR i ← 0 TO 9
    scores[i] ← scores[i] * 100
ENDFOR
```

More complicated algorithms can be built with arrays. The example on the right shows how the average of all the numbers in the array is found. The totalScore variable is created outside the FOR loop. If it were created inside the loop it would keep on resetting to zero every time the loop repeated.

```
totalScore ← 0
FOR i ← 1 TO 9
    totalScore ← totalScore + scores[i]
ENDFOR
average ← totalScore / 10
```