Learning Objectives

- Be familiar with the concept of a data structure
- Be familiar with arrays of up to 3 dimensions, tuples and records

Arrays, Lists and Tuples

0	1	2	3	4	5
blue	green	purple	cyan	magenta	violet

Allows multiple items of data to be stored under one identifier

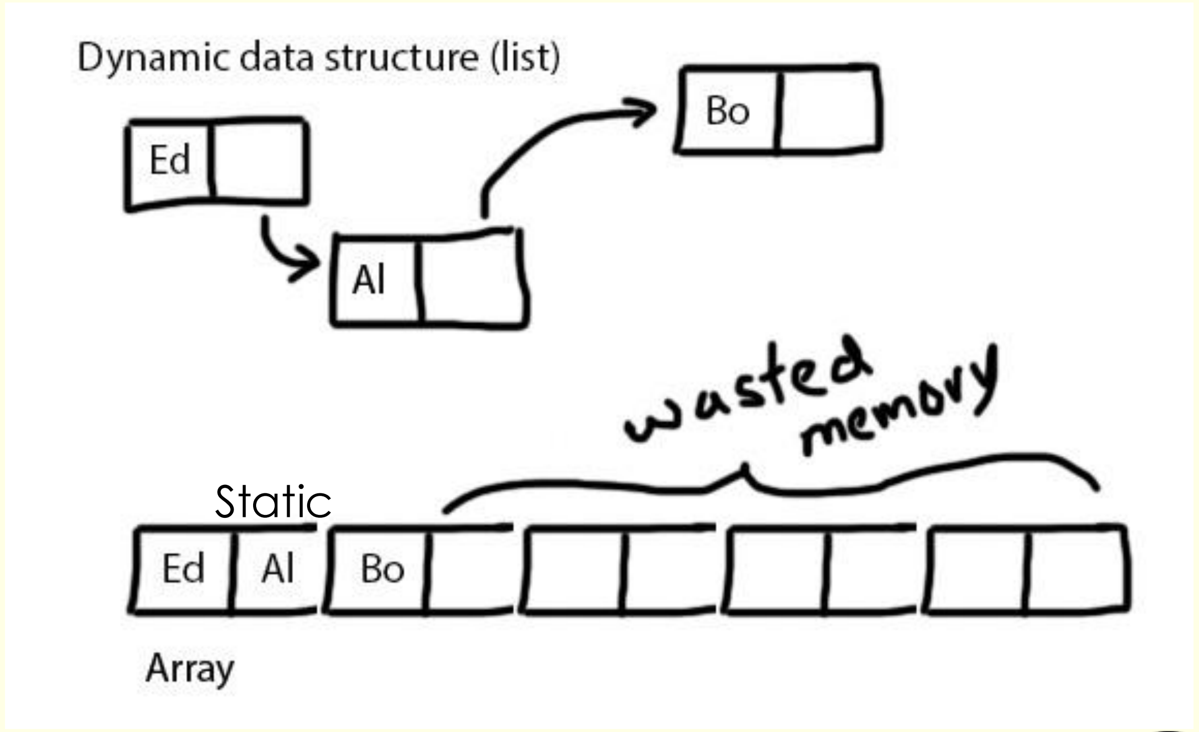
Can store a table structure

Reduces need for multiple variables

Array … must be of the **same data type** and fixed size (**static**)

List … A **list** is a sequence of values, can be **different data types**. It is **mutable** – so you can add, remove and change items in it. (**dynamic**)

Tuple…is **immutable**, so once it's set up, it can't be changed.



1-dimensional arrays

An array is defined as a finite, ordered set of elements of the same type, such as integer, real or char.

Finite means that there is a specific number of elements in the array.

Ordered implies that there is a first, second, third etc. element of the array.
For example, (assuming the first element of the array is myArray[0]:

```
myArray = [51, 72, 35, 37, 0, 3]
x = myArray[2] #assigns 35 to x
```

Worked Example

Every year the RSPB organises a Big Garden Birdwatch to involve the public in counting the number of birds of different types that they see in their gardens on a particular weekend. During 30-31 January 2016, more than 8 million birds were counted and reported.

The scientists add all the sightings together, and once the data has been analysed, they can discover trends and understand how different birds and other wildlife are faring.

An array of strings could be used to hold the names of the birds, and an array of integers to hold the results as they come in. As a simple example we will hold the names of 8 birds in an array:

```
birdName = ["robin", "blackbird", "pigeon", "magpie", "bluetit","thrush", "wren", "starling"]
```

We can reference each element of the array using an index. For example:

```
birdName[2] = "pigeon" #the index here is 2
```

Most languages have a function which will return the length of an array, so that

```
numSpecies = len(birdName)
```

will assign 8 to numSpecies.

Worked Example

- To find at which position of the array a particular bird is, we could use the following pseudocode algorithm:

```
bird = input("Enter bird name: ")
birdFound = False
numSpecies = len(birdName)
for count = 0 to numSpecies - 1
   if bird == birdName[count] then
      birdIndex = count
      birdFound = True
   endif
next count
if birdFound == False then
   print("Bird species not in array")
else
   print("Bird found at",birdIndex)
endif
```

We need a second array of integers to accumulate the totals of each bird species observed.
We can initialise each element to zero.

birdCount = [0,0,0,0,0,0,0,0]

To add 5 to the blackbird count (the second element in the list) we can write a statement

birdCount[1] = birdCount[1] + 5

Worked Example

The following algorithm enables a member of the Birdwatch team to enter results as they come in from members of the public.

```
birdName = ["robin", "blackbird", "pigeon", "magpie", "bluetit",
                                    "thrush", "wren", "starling"]
birdCount = [0,0,0,0,0,0,0,0]
bird = input("Please input name of bird (x to end): ")
while bird != "x"
   birdFound = False
   for count = 0 to 7
      if bird == birdName[count] then
         birdFound = True
         birdsObserved = input("number observed: ")
         birdCount[count] = birdCount[count] + birdsObserved
      endif
   next count
   if birdFound == False then
      print("Bird species not in array")
   endif
   bird = input("Please input name of bird (x to end): ")
endwhile
#now print out the totals for each bird
for count = 0 to 7
   print(birdName[count], birdCount[count])
next count
```

Useful List Functions

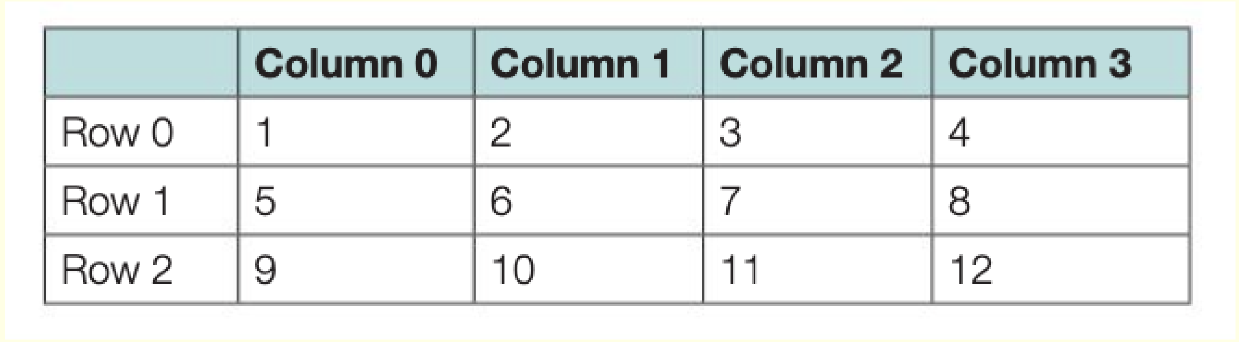
- **len(my_list)** length of list
- **my_list.append(item)** adds an item to the end of the list
- **my_list.extend(list)** adds a list to the end of a list
- **my_list.insert(index, item)** adds an item to the list at the specified index
- **my_list.remove(item)** removes the first occurrence of the item from the list
- **del my_list[index]** delete from a specific index
- **my_list.pop([index])** removes and returns an item from the list (if index is omitted it removes the last item)
- **my_list.index(item)** returns a number indicating the position of the item in the list
- **my_list.count(item)** counts how many times the item appears in the list
- **my_list.sort()** sorts list items into order
- **my_list.reverse()** reverses the list

2-dimensional arrays

An array can have two or more dimensions. A two-dimensional array can be visualised as a table, rather like a spreadsheet.

Imagine a 2-dimensional array called numbers, with 3 rows and 4 columns.

Elements in the array can be referred to by their row and column number, so that numbers[1,3] = 8 in the example below.



	Column 0	Column 1	Column 2	Column 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

What is the value of numbers[2,1]?

Worked Example

Write a pseudocode algorithm for a module which prints out the quarterly sales figures (given in integers) for each of 3 sales staff named Anna, Bob and Carol, together with their total annual sales. Assume that the sales figures are already in the 2-dimensional array quarterSales. The staff names are held in a 1-dimensional array staff.

```
staff = ["Anna","Bob","Carol"]
quarterSales = [[100,110,120,110],
                [350,355,360,360],
                [200,210,220,220]]
for s = 0 to 2
   annualSales = 0
   #output staff name
   (insert statement here)

   for q = 0 to 3
      print("Quarter ", q, quarterSales[s,q])
      annualSales = annualSales + quarterSales[s,q]
   next q
   print("Annual sales: ", annualSales)
next s
```

What statement needs to be inserted after the comment #output staff name in order to output the staff name?

print (staff[s])

Arrays of three dimensions

- Arrays may have more than two dimensions. An n-dimensional array is a set of elements of the same type, indexed by n integers.
- In a 3-dimensional array x, a particular element may be referred to as x[4,5,2], for example.
- The first element would be referred to as x[0,0,0].

Tuples

- A tuple is an ordered set of values, which could be elements of any type such as strings, integers or real numbers, or even graphic images, sound files or arrays.
- Unlike arrays, the elements do not all have to be of the same type. However, a tuple, like a string, is immutable, which means that its elements cannot be changed, and you cannot dynamically add elements to or delete elements from a tuple.

In Python a tuple is written in parentheses, for example:

```
pupil = ("John", 78, "a")
```

You can refer to individual elements of a tuple, for example:

```
name = pupil[0]
```

but the following statement is invalid:

```
pupil[0] = "Mary"
```

Example: Write a pseudocode algorithm to perform the following task:

Create a function that accepts a 1D array of integers and returns the index of the largest number in the array.

Example Solution for the Algorithm

```
FUNCTION findLargestIndex(arr)
    maxIndex = 0
    maxValue = arr[0]
    FOR i = 1 TO len(arr) - 1
        IF arr[i] > maxValue then
            maxValue = arr[i]
    return maxValue
End function
```