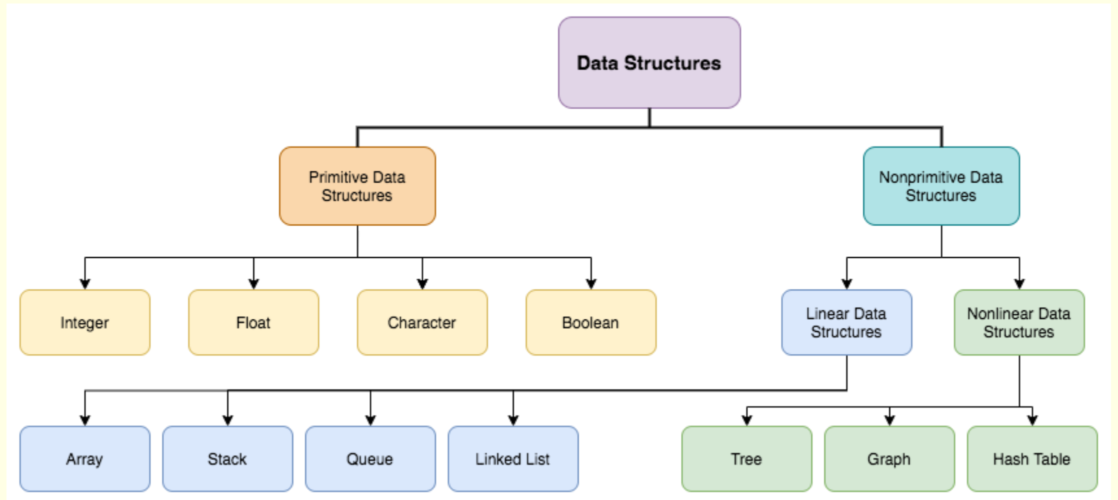
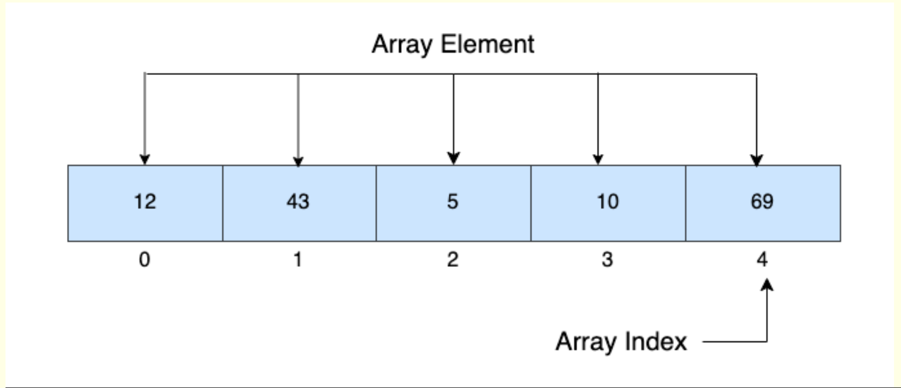
**Data Structures KO: Arrays, Lists, Tuples and records**



| **Static Data Structure** | **Dynamic Data Structure** |
|---|---|
| A static data structure has a fixed size that is determined at the time of creation and cannot change during execution.<br><br>For example:<br>Array (mutable – can be changed)<br>Tuple (Immutable – can not be changed)<br><br>**Advantages**:<br>• The program/memory management system can allocate a fixed amount of memory at compile time. No overhead for resizing.<br>• **Faster access:** Direct access to elements via index.<br><br>**Disadvantages:**<br>• **Fixed size:** Wastes memory if not fully used, or may not be large enough.<br>• **No flexibility:** Cannot grow or shrink as needed during runtime. | A dynamic data structure can grow or shrink at runtime, allocating and deallocating memory as needed.<br><br>For example:<br>List, Linked List, Stack, Queue, Binary tree, Graph<br><br>**Advantages:**<br>• **Flexible size:** Adjusts to the program's needs.<br>• **Efficient memory use:** Only uses memory when needed.<br>• Inserting, merging and deleting of items is very easy and requires little processing power.<br><br>**Disadvantages:**<br>• **More complex to implement** (especially pointer management).<br>• **Slower access:** No direct access like with arrays.<br>• **Overhead:** Extra memory needed for pointers or management. |

*"No overhead for resizing."*

This means that with **static data structures**, since the size is fixed and set at **compile time**, the program doesn't have to do any **extra work** to change the size during runtime.



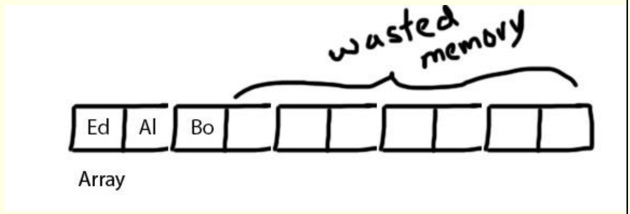
Allows multiple items of data to be stored under one identifier

Can store a table structure

Reduces need for multiple variables

Array

- **Static** (Fixed size)
- **Same data type** for all elements
- Stores data linearly
- Elements are stored in **contiguous memory locations**, allowing fast sequential access.
- Contents are mutable - The values of elements can be changed,
- Memory allocated at compile time



**Example in pseudocode:**

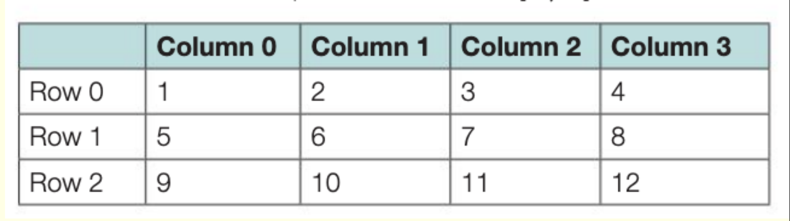
```
Array birdName(4)

birdName[0] = "pigeon" #the index here is 0
birdName[1] = "robin" #the index here is 1
birdName[2] = "blackbird" #the index here is 2

Result: birdName = [“pigeon”, “robin”, “blackbird”, “”]

numSpecies = len(birdName) #will assign 4 to numSpecies.
```

**2-dimensional arrays**

An array can have two or more dimensions.
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:

**Example in pseudocode:**

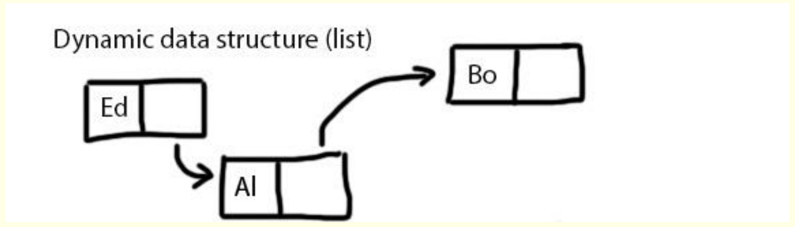
```
Array numbers (3,4)

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 |

List

- **Dynamic size** (can grow and shrink)
- Can store **different data types** in the same list
- Mutable (elements can be changed, added, removed)
- Memory allocated at run time
- List values are stored **non-contiguously**. This means they do not have to be stored next to each other in memory.



Example showing the use of a dynamic list structure in Python:

```
numbers = [] #Initalise list
numbers.apprear(12) #Add a value
numbers.apprear(8)
numbers.apprear(5)

Result: numbers = [12,8,5]

print(numbers.pop()) #Remove item

Result: numbers = [12,8]
```

Tuple

- A tuple cannot be changed at runtime // a tuple is immutable
- Can store different data types
- Cannot be modified (no adding, removing, or changing elements)

Example:

```
colour = (255, 0, 0)  # Red color
```

**Records**

A record is a data structure that groups together related items of data
You can store more than one type of data together
A record is an unordered data structure
Can have multiple instances

Player records:

Player1 record

| |
|---|
| Olivia<br>35 |

Player 2 record

| |
|---|
| Luke<br>40 |

Creating a record structure

```
recordStructure recordstructurename

  fieldname : datatype
  …
rearRecordStructure
```

The pseudocode to define a new complex data structure called player:

```
RECORD player
  name: String
  score: int
REARRECORD
```

*Adding data to the record*

```
recordidentifier : recordstructurename
recordidentifier.fieldname = data
```

The pseudocode to define a new complex data structure called Player:

```
Player1: player
Player1.player.name = 'Olivia'
Player1.player.score = 35
```

Array of Records

Records are treated as data types, so they can be held within a single array.

This allows for storage of more than one record within the same structure. This structure is essentially an array of records.

The table below simplifies the way that this data would be held in memory.

- The records for the players could be stored in a 1D array.
- It allows easy access/indexing/manipulation of each data item in turn
- 1D Array can hold multiple items of same data type – record
- Maximum number of array elements is known

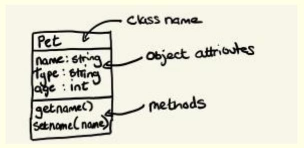
| 0 | 1 | 2 | 3 |
|---|---|---|---|
| Olivia | Luke | Adam | Alex |
| 35 | 40 | 25 | 35 |

Pseudocode of Arrays of records:

```
Players(100) As player
```

We can then reference any element of the array:

```
Players(3).name = "Jane"
```

| | Record | Class |
|---|---|---|
| Similarities | | |
| Data structure | A record is a data structure that stores data together, organised by attributes. | A *class* is a record with associated methods. Each object is a data structure with attributes stored together |
| Set up in advance | Attributes and structure for the record are set up. Meaning that it is created by the programmer for a particular purpose. | Constructor method defines the class object |
| Store data of different types | recordStructure pets<br>    name : String<br>    type : String<br>    age : Int<br>rearRecordStructure |  Pet<br>name: string<br>type : string<br>age : int<br>getname()<br>setname(name)<br>Class name<br>Object attributes<br>methods |
| Both can have multiple instances | Yes | Yes |
| Accessed by their names | Yes | Yes |
| Differences | | |
| Class also has methods<br><br>Class can include visibility of properties / private | | |