Simple data type: contain a single value eg- float, int, char

Complex data type: has multiple parts. Multiple simple data type joined into a single entity. EG- array

Array: is a complex data type that group together a number of different values of the same type in a single structure (array of char, array of int)

- Array is a contiguous block (joined block of memory) of memory holding "n" elements of given type.
    - Index- Are numbers that identify an element eg: 0, 1,2 3, are index numbers
    - Element- values stored in memory
    - Size- how many elements stored int myNums**[50] -1 = 49 elements** total number of elements
    - Advantages: Instead of having lots of separate variables, each storing a value, we can have a single variable storing many pieces of data
    - Don't have to declare many variables individually
    - When passing array into a function as parameter
    - Useful: scalability for when more

Static array vs dynamic array: Create array, declare, and initialize size and it is fixed. Dynamic array needs an array and calculate size of array at runtime and creates size that is needed (decides when program created how big).

Strings: Strings are data types that are simply a sequence or string of characters of a given length. So a series of characters related with each other. (like *Array* of characters). Array size is static while length of array changes. **Eg: size is 10 so the length may be 1 2 or 9.**

Two types of algorithmic approach to dealing with lots of data

Multiple loaded records:

- This involves loading multiple records from the file into memory.
- This will involve allocating sufficient memory to hold all of these records.
- This will usually be done with an array.

Single batch processing:

- The alternative to loading multiple records is to only load as little of the file into memory at one time as possible.
- involves reading only a single record in, processing it in memory and then reading in the next record.

<u>Principle of Object Orientated programming:</u>

Abstraction/Encapsulation:

- Is where people who are using classes do not need to know how those classes work, just what they do (*encapsulation*).
- Access to either the code or data inside of an object is often restricted.

Code-reuse/Inheritance:

- Classes created must be general "blueprints".
- Objects can then be created as instances of these classes facilitating code re-use.

High-Cohesion:

- Applies to both the design of the class and their code/modules
    - Classes only store data that are relevant to their task
    - Methods are narrowed focused in terms of the task they perform

Low coupling:

- It is important data is completely inaccessible outside of the object
- In some ways it is slightly less significant since less data is passed around between modules.

Classes vs Objects:

- A class therefore is a general specification of an object, a little like a *blueprint* or a design eg ucoz. Objects are contained within a class they use the class as a blueprint/backdrop they can inherit from original class.
- Classes are therefore effectively complex data types and objects are like variables declared from that type.
- So Class relates to the properties of object from a class Object

Object Orientated Programming vs Structured Model/Modular Programming:

- Structured Model: involves taking the high-level algorithm which solves a problem and dividing it up into the individual steps
- OOP: OO paradigm, the program is divided up into objects

- OO is much more data-centric in terms of developing a solution
- Traditional approach is much more concerned with the steps involved in solving the problem