A **Variable** is a named placeholder for storing of data.

A **Data Type** in a programming language is a set of data with predefined values, such as an integer, floating point, string, etc. Data Types are separated typically by type of data (numerical, character string) and amount of storage used (Integer is often 2 bytes, floats are often 4).

There are two main categories of data types, namely System-defined/**Primitive data types**; and **User-defined data types**.

Strongly typed languages like C# often provide Primitive data types like int, float, char, bool, etc. The distribution of storage space and the specific types often differ based on the programming language used.

These languages also often provide tools for users to create their own data types, when a primitive one does not match their requirements. This often involves combining/grouping primitive data types together.



A **Data Structure** is a particular way of storing and organizing data such that it can be used efficiently. General Data structure types include arrays, files, linked lists, stacks, queues, trees, graphs, etc.

Primitive data types come with operations that can be performed against them (addition of integers, concatenation of strings). For user-defined data types we also need to define their operations. We combine the data structures with their operations, and we call this **Abstract Data Types** (**ADTs**).

Commonly used ADTs include Linked Lists; Stacks; Queues; Priority Queue; Binary Trees; Dictionaries; Disjoint Sets (Union and Field); Hash Tables; Graphs; and many others.

Consider a Stack, which uses a Last-In-First-Out mechanism for storing data. Common operations for a stack would include:

* Pushing an item to the stack,
* Creating a stack,
* Popping an element from the stack,
* Finding the top of a stack,
* Finding the length of the stack,
* Etc.

An **algorithm** is the step-by-step unambiguous instructions to solving a problem. There are two main ways to judge the merits of algorithm i.e., correctness – does the algorithm solve the problem for the range of inputs and in a finite number of steps; and efficiency – how much resources, both time and space, is required for the algorithm to solve the problems.

For any problem, there are often multiple ways to solve them. Working smart and efficiently is thus valued, as solving the problem efficiently is overall goal. Algorithm analysis helps us determine which algorithm is more efficient in terms of time and space consumed.

STOPPED 1.7