Data types and data structures

A data type (also called abstract data type or ADT) defines the operations and behavior supported by an object.

A data type is a concept, similar to mathematical concepts such as function, set, or sequence.

Examples of data types are Stack, Queue, Set, Dictionary.

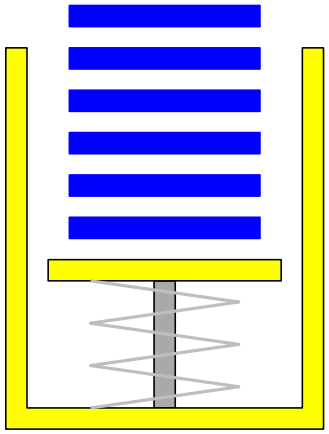
A data structure is an implementation of a data type: An object that provides all the operations defined by the data type, with the correct behavior.

We often have multiple, different implementations for the same data type: Stacks can be implemented with arrays or with linked lists, sets can be implemented with search trees or with hash tables.

Stacks

Think about a stack of books, or dishes.

One can only access the top of the stack.



Operations on a stack:

- push something on top,
- look at the top,
- pop something off the stack.

Book definition

An abstract data type is a programmer-defined data type that specifies a set of data values and a collection of well-defined operations that can be performed on those values.

Abstract data types are defined independent of their implementation.

- We can focus on solving the problem instead of the implementation details.
- Reduce logical errors by preventing direct access to the implementation.
- Implementation can be changed.
- Easier to manage and divide larger programs into smaller modules.

The Stack trait

```
trait Stack[T] {
  def push(el: T): Stack[T]
  def top: T
  def pop(): T
  def isEmpty: Boolean
}
```

This trait shows the specification of a stack as an Abstract Data Type (ADT). Traits are also called interfaces (in Java) or abstract base classes (in C++).

An abstract data type is defined by

- what operations are allowed on it, and
- the semantics (meaning) of these operations.

Using a stack:

```
val stack: Stack[Int] =
    new scala.collection.mutable.Stack[Int]
    with Stack[Int]

stack.push(5)
stack push 7
val k: Int = stack.top
val p = stack.pop()
```

To create an object with Trait `Stack`, we need to create a specific implementation of stacks.
Here we use a Scala collection class.

Balanced symbol checker

`(){[(){}](())}` is correct,
but `(){[({)}](())}` is not correct.

How to check whether a string is balanced:

1. Make an empty stack,
2. For each symbol in the string:
   (a) If the symbol is an opening symbol, push it on the stack.
   (b) If it is a closing symbol, then
      i. If the stack is empty, return false.
      ii. If the top of the stack does not match the closing symbol, return false.
      iii. Pop the stack.
3. Return true if the stack is empty, otherwise false.