
🧩 Abstract Data Type (ADT) – Complete Notes

---

1. Understanding ADT

Meaning of ADT:

ADT (Abstract Data Type) = *A data type defined by its behavior (operations) from the point of view of a user, while hiding how it's implemented internally.*

In short:
An ADT defines **what** operations you can perform, **not how** they're performed.

---

2. Breaking Down the Term

Data Type

A **data type** is defined by:

1. **Representation of data** – how it's stored in memory.
2. **Operations on data** – what you can do with it.

Example:
Integer data type in C/C++

- Representation: 2 bytes (16 bits) in memory
  - 1 bit → sign (positive/negative)
  - 15 bits → value
- Operations allowed:
  - Arithmetic: +, -, *, /, %
  - Relational: <, >, ==, !=
  - Increment/decrement: ++, --

So, a data type = representation + operations.

Abstract

- *Abstract* = Hiding internal details (implementation).
- Example:
  You can use integers (int a = 5; a++) without knowing how increment is actually done in binary.
  Internal binary operations are **abstracted away**.

Therefore:

An **Abstract Data Type** hides *how* data and operations are implemented and exposes only *what* operations you can perform.

---

3. Why ADT Exists

- Originated with **Object-Oriented Programming (OOP)**.
- In OOP, you can define your own data types (classes) that:
  - Contain both **data representation** and **operations**.
  - Hide implementation details.
- These user-defined classes → are **ADTs**.

---

4. Example: List ADT

We'll use a **list** (collection of elements) to understand ADT.

Example list:

[5, 8, 9, 4, 10, 12, 14]

- Indices: start from 0 → 6
- It's a **collection of elements** with a certain **order**.

Representation of List

To represent a list, we need:

1. **Space** to store elements
2. **Capacity** – maximum number of elements the list can hold
3. **Size** – number of elements currently in the list

Representations:

- Using an **Array**
- Using a **Linked List**

Both are valid ways to *implement* the same ADT.

Operations on List ADT

These are the operations that define the *behavior* of the list (the "interface" of the ADT):

Operation	**Meaning**	**Example**
add(x) / append(x)	Add an element at the end	[5,8,9] → append(10) → [5,8,9,10]
insert(index, x)	Insert element x at a given index (shift elements right)	insert(1, 7) → [5,7,8,9]
remove(index)	Remove element at given index (shift elements left)	remove(2) → [5,7,9]
set(index, x) / replace(index, x)	Replace element at given index	set(1, 25) → [5,25,9]
get(index)	Retrieve element at given index	get(0) → 5
search(x) / contains(x)	Find whether element exists (and return index)	search(9) → index 2
sort()	Arrange list elements in ascending/descending order	[5,25,9] → sort() → [5,9,25]
reverse()	Reverse the list	[5,9,25] → [25,9,5]
merge(list2)	Combine two lists	[1,2,3] + [4,5] → [1,2,3,4,5]
split()	Divide list into two sublists	[1,2,3,4,5,6] → [1,2,3], [4,5,6]

So, **List ADT** = Representation (array/linked list) + Operations (add, remove, search, etc.)

Abstract Nature

When we use a list in programming:

- We don't care *how* elements are stored or moved.
- We only care *what* we can do — like append, insert, delete, etc.

That's what makes it **abstract** — internal details are hidden.

---

5. Defining ADT in Object-Oriented Terms

In C++:

- A **class** can define both:
  - Data (representation)
  - Operations (functions)

- When you create a class with both data + operations and hide its internal details (using private members), it's an **ADT**.

Example:

```
class List {

private:

  int arr[100];   // data representation

  int size;

public:

  void add(int x);    // operation

  void remove(int index);

  int get(int index);

};
```

The user can use the List class methods without knowing how arr is managed internally.
That's the essence of **abstraction**.

---

6. How ADT Works in Python

Python handles **Abstract Data Types** naturally using **classes** and **built-in data structures**.

Example 1: Python's Built-in List

```
my_list = [5, 8, 9, 4, 10]

my_list.append(15)

my_list.insert(2, 99)

my_list.remove(4)

print(my_list[1])
```

- append(), insert(), remove(), sort() are operations.
- Python internally manages how the list grows and shrinks in memory.
- You use these methods **without knowing** how data is stored → **Abstract Data Type** in action.

Internally, Python lists are implemented as **dynamic arrays** in heap memory:

- When the list runs out of capacity, it **automatically resizes**.
- You don't need to manage memory manually (unlike C/C++).

Example 2: Custom ADT in Python

You can define your own ADT using classes — exactly how it's done in C++.

```
class ListADT:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def remove(self, x):
        self.data.remove(x)

    def get(self, index):
        return self.data[index]

    def search(self, key):
        return key in self.data
```

- data is the **representation**.
- add, remove, get, search are the **operations**.
- User can interact only with methods → **internal details are abstracted**.

Usage:

```
lst = ListADT()
lst.add(5)
lst.add(10)
lst.remove(5)
print(lst.get(0))   # 10
```

So yes, every Python class that hides data and exposes methods = an **ADT**.

7. Key Points Summary

Concept	**Meaning**
Data Type	Defines data representation + operations
Abstract	Hides internal details
ADT	Combines data + operations while hiding implementation
List ADT Example	Defines how list behaves, not how it’s stored
Representation of List	Array or Linked List
Operations on List	Add, Insert, Remove, Search, Sort, etc.
In Python	Built-in lists and user-defined classes are ADTs
In C++	ADTs implemented through classes and encapsulation

8. How ADT Differs from Implementation

ADT (Abstract)	**Implementation (Concrete)**
Defines *what* operations can be done	Defines *how* operations are performed
Example: “List supports add, remove, get”	Example: “List implemented using array or linked list”
Language-independent concept	Language-specific coding logic
User-facing interface	Internal structure

9. ADT in Data Structures Subject

Every data structure (array, stack, queue, linked list, tree, graph, hash table) is studied as an **ADT**:

- You’ll define its **data representation**
- Specify its **operations**
- Implement it using C/C++/Python

So in your data structures course:

“Implement Stack ADT using Array”
means — represent stack data using array and define operations: push, pop, peek, etc.

🧠 Quick Recap

Term	Description
Abstract Data Type (ADT)	Data + operations, hiding implementation
Data Representation	Defines how data is stored
Operations	Defines what you can do with data
List ADT	Example combining data and common operations
Python Implementation	Done using classes or built-in list
Purpose	Encapsulation, abstraction, modular design

🪄 In One Line

ADT = “What” you can do with data, not “How” you do it.