Data Structures in Python
Chapter 2

1. Abstract Data Type(ADT)
2. Performance Analysis
3. Big-O Notation
4. Growth Rates

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

나는 인애를 원하고 제사를 원하지 아니하며 번제보다 하나님을 아는 것을 원하노라 (호6:6)
하나님은 모든 사람이 구원을 받으며 진리를 아는데에 이르기를 원하시느니라 (딤전2:4)

그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전10:31)

Agenda & Readings

- Some Software Design Principles
- Abstract Data Type (ADT)
  - What is an ADT?
  - What is a Data Structure?
- Examples on ADT:
  - Integer, Set, and List

- Resources:
  - Textbook: Problem Solving with Algorithms and Data Structures
    - Chapter 1.5 Why Study Data Structures and Abstract Data Types?
  - Textbook: www.github.com/idebtor/DSpy:
    - Ch1-1: Introduction

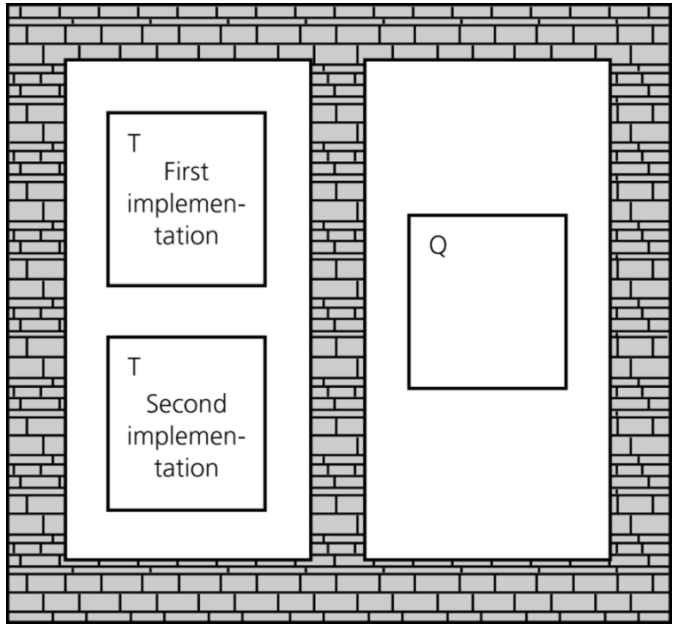
Software Design Principles - Modularity

- Modularity divides a program into manageable parts.
  - Keeps the complexity of a large program manageable.
  - Isolates errors.
  - Eliminates redundancies.
  - Encourages reuse (write libraries).

- A modular program is
    - Easier to write.
    - Easier to read.
    - Easier to modify (or maintain).

Software Design Principles - Information Hiding

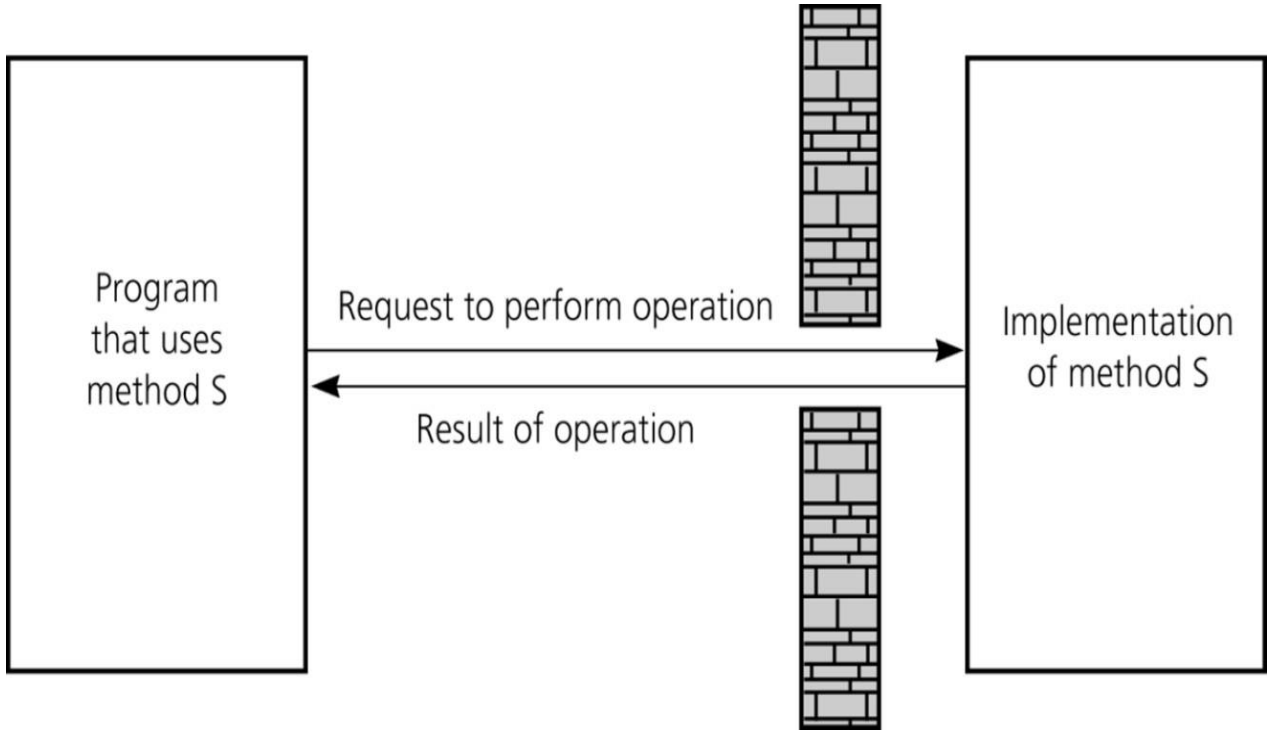
- Hides certain implementation details within a module.
- Makes these details inaccessible from outside the module.
- Isolates the implementation details of a module from other modules.



Isolated tasks: the implementation of task T does **not** affect task Q

Software Design Principles - Isolation of Modules

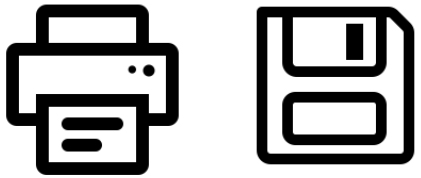
- The isolation of modules is not the end. (otherwise module would be useless)
- Methods' specifications, or contracts, govern how they interact with each other.
- Similar to having a wall hiding details, but being able to access through "hole in the wall"



Software Design Principles - Abstraction

- What does ‘abstract’ mean?
  - From Latin: to ‘pull out’—the essentials
  - To defer or hide the details
  - Abstraction emphasizes **essentials** and defers the details, making engineering artifacts easier to use.

- Example:
  - I don’t need a mechanic’s understanding of what’s under a car’s hood in order to drive it.
    - What’s the car’s interface?
    - What’s the implementation?



Software Design Principles - Abstraction

- Abstraction
  - The principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate solely on those aspects which are **relevant.**

- How do we achieve:
  - Modularity
  - Information hiding
  - Isolation of modules
    - i.e. The abstraction of **what** from the **how**

Software Design Principles - Abstraction

- Abstraction separates the purpose and use of a module **from its implementation**.

- A module's specifications should
  - Detail how the module behaves.
  - Identify details that can be hidden within the module.

- Advantage:
  - Hides details (easier to use).
  - Allows us to think about the general framework overall solution) & postpone details for later.
  - Can easily replace implementations by better ones and it **does not affect the users of the modules (or library).**

Software Design Principles - Data and Operations

- What do we need in order to achieve the above Software Design principles?

- **For example,** what are the typical operations on data? (example: database)
  - Add data to the database
  - Remove data from the database
  - Find data (or determine that it is not in the data base)
  - Ask questions about the data in a data collection
    (e.g. how many CS50 students take Linear Algebra course?)

- **Question:** Do we need to know what data structures used?
  - No, better make implementation independent of it!

Abstraction - Example

- Programming Language
  - Programming:
    - For instructing a computer to perform a computing task
    - But more than just a means of computing …
      - a framework within which we **organize our ideas**
      - a means of communicating ideas in the community
  - Language:
    - Provides for combining simple ideas to form more complex ideas.

- Example

羊 手 戈 ⇨ 義

[sheep] [hand] [spear head] [righteousness]

손(손 수手)으로 양(羊)를 쳐서(창 과戈) 드림. 손의 창으로 양을 잡아 그 피로 나를 덮는 것이 옳을 의
손의 창으로 나를 쳐서 어린 양 예수님께 복종하는 것이 옳은 의

Abstraction - Example

- Programming Language
  - Programming:
    - For instructing a computer to perform a computing task
    - But more than just a means of computing …
      - a framework within which we **organize our ideas**
      - a means of communicating ideas in the community
  - Language:
    - Provides for combining simple ideas to form more complex ideas.

- Example

[ox] [sheep] [excellent] [spear head] **[sacrifice]**

흠 없는(秀) 양(羊)과 소(牛)를 창(칼)로 잡아(戈) **희생**시킴으로써(犧) 하나님께 제물로 드렸다 (레1:2-4)

Software Design Principles - Data and Operations

- Asks you to think what you can do to a collection of data independently of how you do it.
- Allows you to develop each data structure in relative isolation from the rest of the solution.
- A natural extension of procedural abstraction.

```
CAR:
Start()
Stop()
TurnLeft()
TurnRight()
```

```
CAR:
Start(){
  Select first gear
  Release parking brake
  Bring clutch up to the friction point
  Press gas pedal
  Release clutch
}
```

NOTE: Implementation can be different for different cars, e.g. automatic transmission.

Abstract Data Types

- An Abstract Data Types (ADT) is composed of
  - A collection of **data**
  - A set of **operations** on that data
- Specifications of an ADT indicate **what** the ADT operations do, **not how** to implement them.
- Implementation of an ADT includes choosing a particular data structure. This let the coders enjoy the freedom of choices in terms of how they code.

Abstract Data Types - Properties and Behaviors

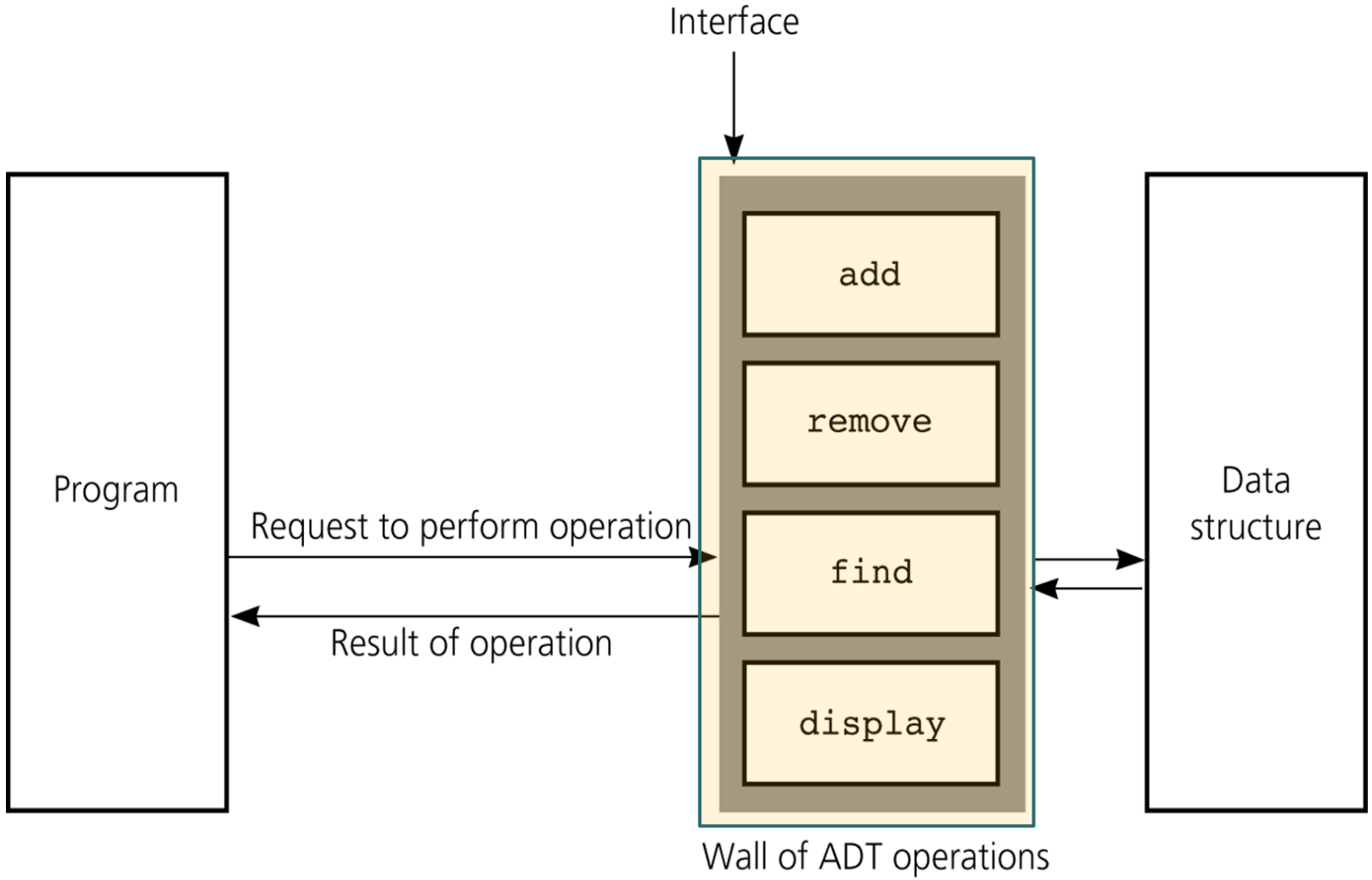
- **The properties** of the ADT are described by the collection of data.
  - The data can be in terms of simple data types and/or complex data types.
  - Simple data types
    - Integer
    - Floating point
    - Character
  - Complex data types
    - Multimedia
- **The behaviors** of the ADT are its operations or functions.

Abstract Data Types - Data structure vs. ADT

- An ADT is not the same with a Data Structure
- Data structure
  - A construct that is defined within a programming language to store a collection of data.
  - Example: arrays
- ADT provides “data abstraction”
  - Results in a wall of ADT operations between data structures and the program that accesses the data within these data structures.
  - Results in the data structure being hidden.
  - Can access data using ADT operations.
  - Can change data structure without changing functionality of methods accessing it.

Abstract Data Types

- A wall of ADT operations isolates a data structure from the program that uses it.



Abstract Data Types - Disadvantages & Advantages

- Disadvantages of Using ADTs
  - Initially, there is more to consider
    - Design issues
    - Code to write and maintain
    - Overhead of calling a method to access ADT information
    - Greater initial time investment
- Advantages of Using ADTs
  - A client (the application using the ADT) doesn't need to know about the implementation.
  - Maintenance of the application is easier.
  - The coder can focus on problem solving and not worry about the implementation.

ADT Examples - Designing an ADT

- An abstract data type (ADT) is a specification of a set of **data** and the set of **operations** that can be performed on the data.
- Such a data type is **abstract** in the sense that it is **independent** of various concrete implementations.
  - Questions to ask when designing an ADT
    - What data does a problem require?
    - What operations does a problem require?
  - Examples:
    - Integers, floating-point
    - Sets
    - Lists
    - Stacks, Queues, Trees …

ADT Examples - Integers

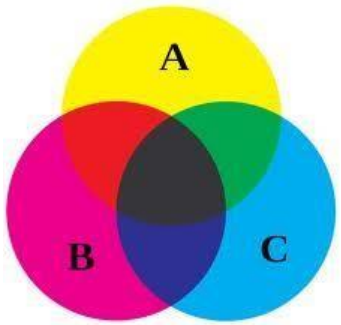
- Data
  - Containing the positive and negative whole numbers and 0
- Operations which manipulate the data:
  - Such as addition, subtraction, multiplication, equality comparison, and order comparison
- Methods for data conversion, output etc.

ADT Examples - Floating-point

- Data
  - Containing the whole possible positive and negative values
  - Precision is limited by number of digits
  - Arrays, strings, lists…
- Operations which manipulate the data:
  - Such as addition, subtraction, multiplication, equality comparison, and order comparison
  - Complex mathematical functions such as exponential, logarithm, triangular functions etc.
  - Rounding's
  - Methods for data conversion, output etc.

ADT Examples - Sets

- Data
  - An unordered collection of unique elements
- Operations which manipulate the data:
  - add
    - Add an element to a set
  - remove
    - Remove an element from the set,
  - Others
    - Get the union, intersection, complement of two Set objects

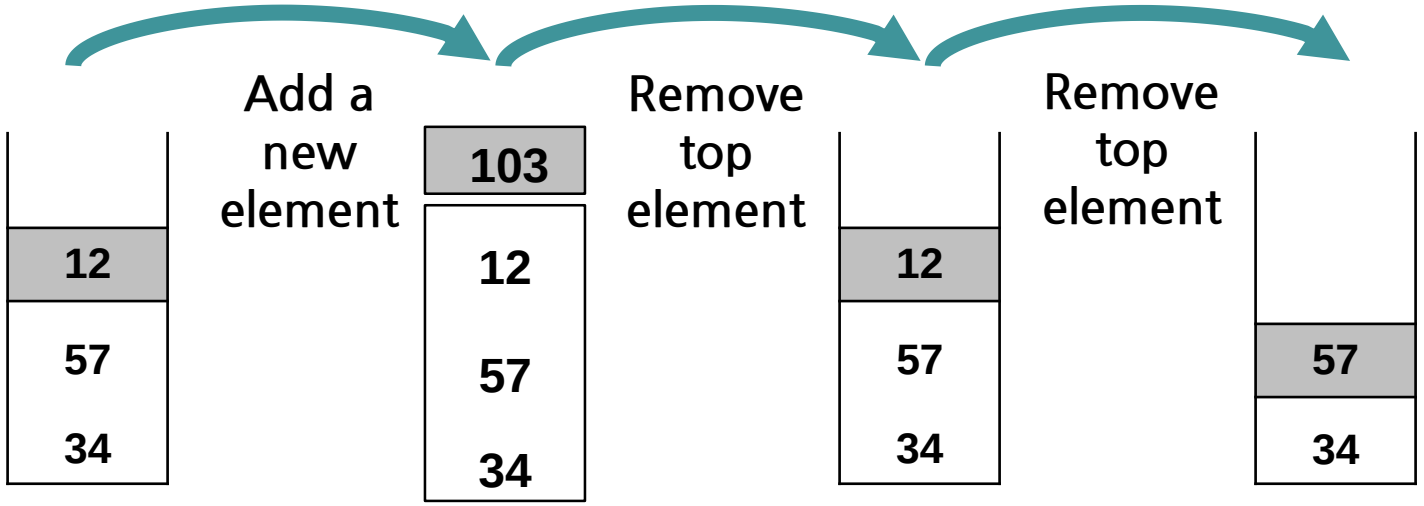


ADT Examples - Linear Structures

- **Linear structures** are data collections whose items are ordered depending on how they are added or removed from the structure.
- Once an item is added, it stays in that position relative to the other elements that came before and came after it.
- Linear structures can be thought of as having two **ends**, **top** and **bottom**, (or front and end or front and back)
- What distinguishes one linear structure from another is the way in which items are added and removed, in particular the location where these additions and removals occur, e.g., add only to one end, add to both, etc.

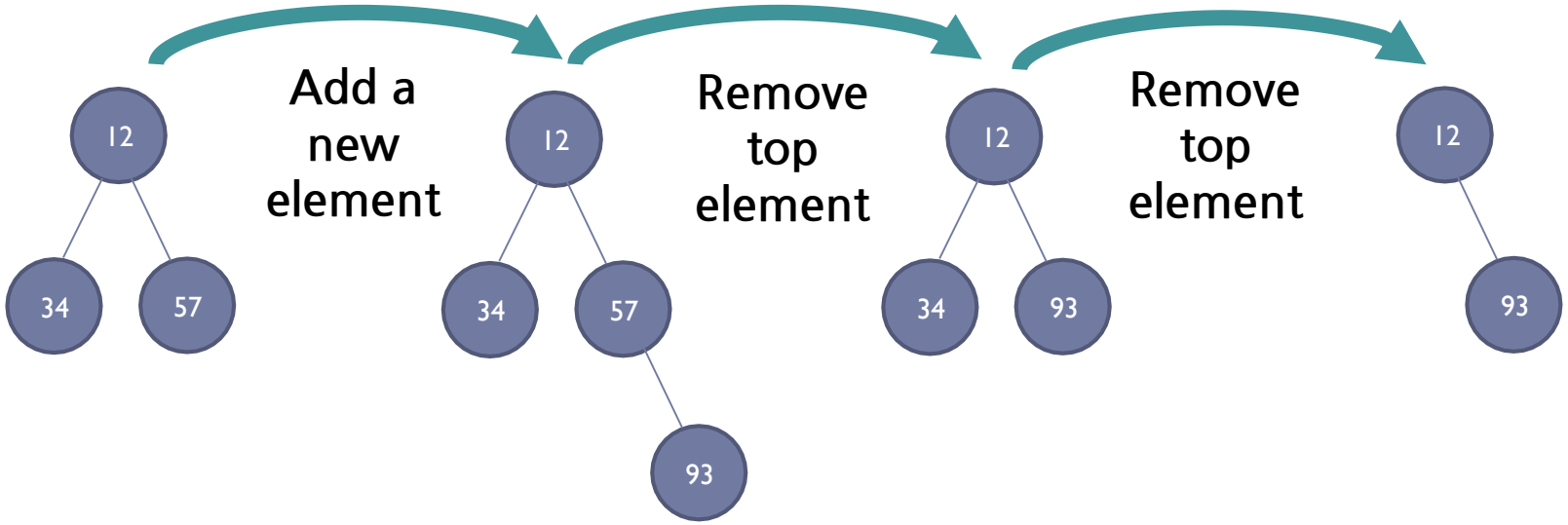
ADT Examples - Linear Structures

- Examples: Stack, Queue, Linked-list, etc.



ADT Examples - Non-Linear Structures

- Every data item is attached to several other data items in a way that is specific for reflecting relationships.
- The data items are not arranged in a sequential structure
- Examples: Tree, Graph, Heap, etc.



Summary

- Solving a problem becomes easier by considering only relevant data and operations and ignoring the implementations (“abstraction”)
- Abstract Data Types (ADTs) enable you to think -
  - **What** you can do with the data independently of **how** you do it
- An abstract data type can be accessed by a limited number of **public methods** -
  - They are often defined by using an interface
- The implementation of the ADT (data structures and algorithms) can be changed without influencing the behavior of the ADT.