Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 13
Abstract Data Types

FIT 1008
Introduction to Computer Science

MONASH University
Information Technology

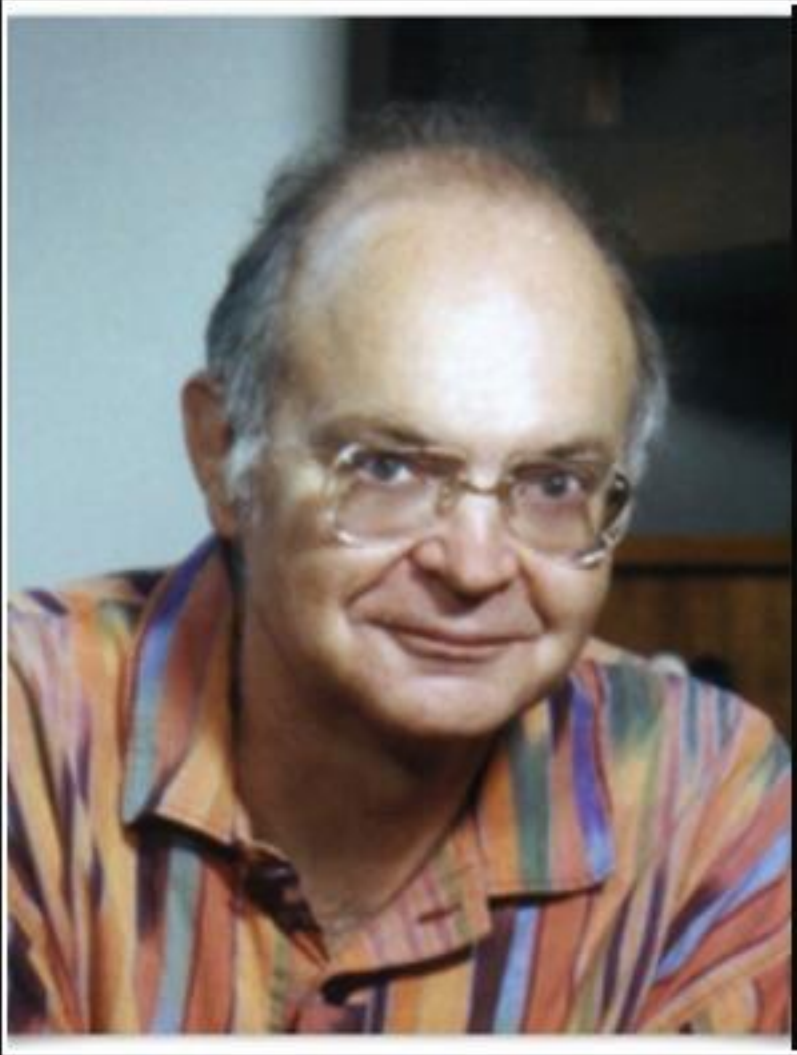
Where are we at?

- We are now familiar with Python basics
- Have learnt how to implement in Python:
  - ➞Bubble Sort
  - ➞Selection Sort
  - ➞Insertion Sort
- Have re-familiarised ourselves with time complexity
- Have started to become accustomed to think
  - ➞The use of invariants for improving our code
  - ➞The properties of our algorithms (e.g., stable?)
  - ➞Their Big O complexity

Objectives for this lecture

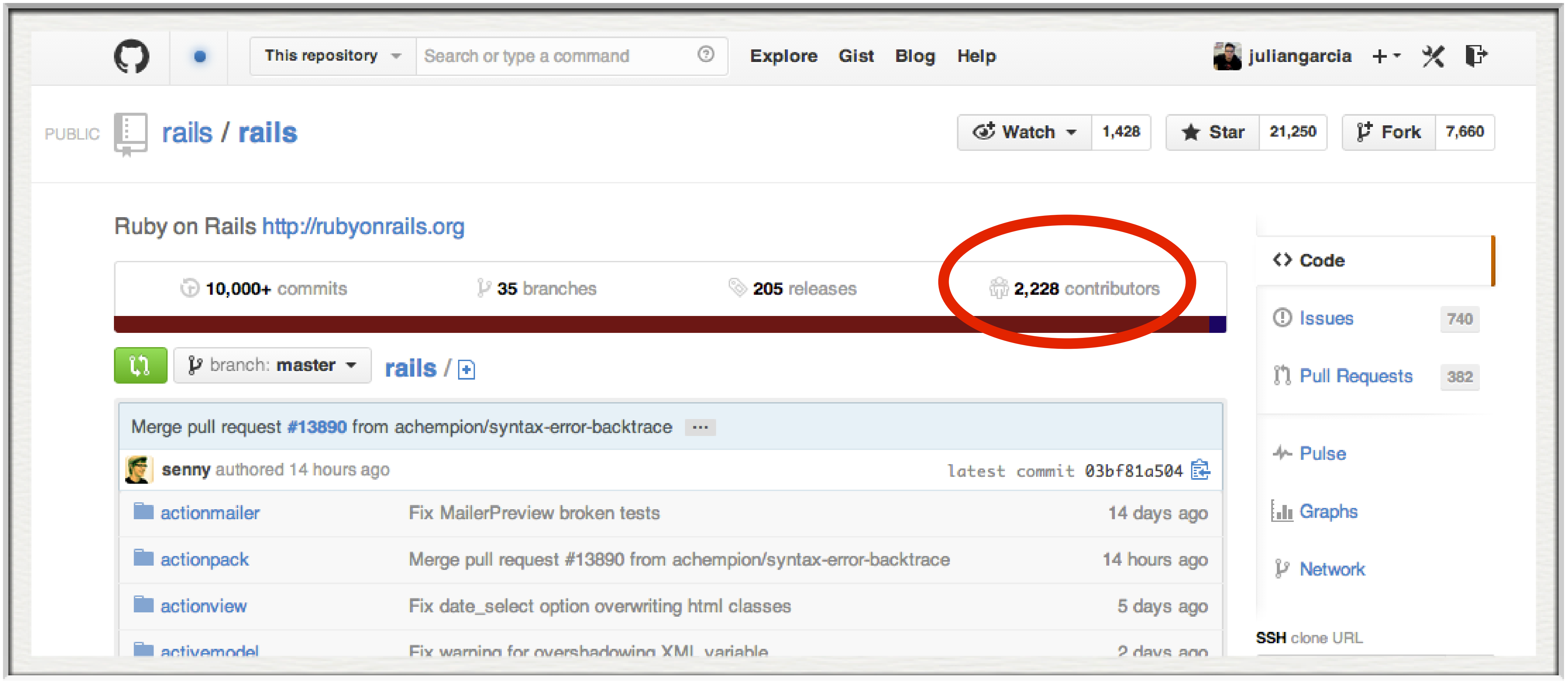
Understand the concepts of:

- Abstraction
- Encapsulation
- Information Hiding
- Data Types
- Data Structures
- Abstract Data Types (ADTs)



In fact, my main conclusion after spending ten years of my life working on the T E X project is that software is hard. It's harder than anything else I've ever had to do.

— Donald Knuth —



Ruby on Rails http://rubyonrails.org

10,000+ commits | 35 branches | 205 releases | 2,228 contributors

Merge pull request #13890 from achempion/syntax-error-backtrace …

senny authored 14 hours ago   latest commit 03bf81a504

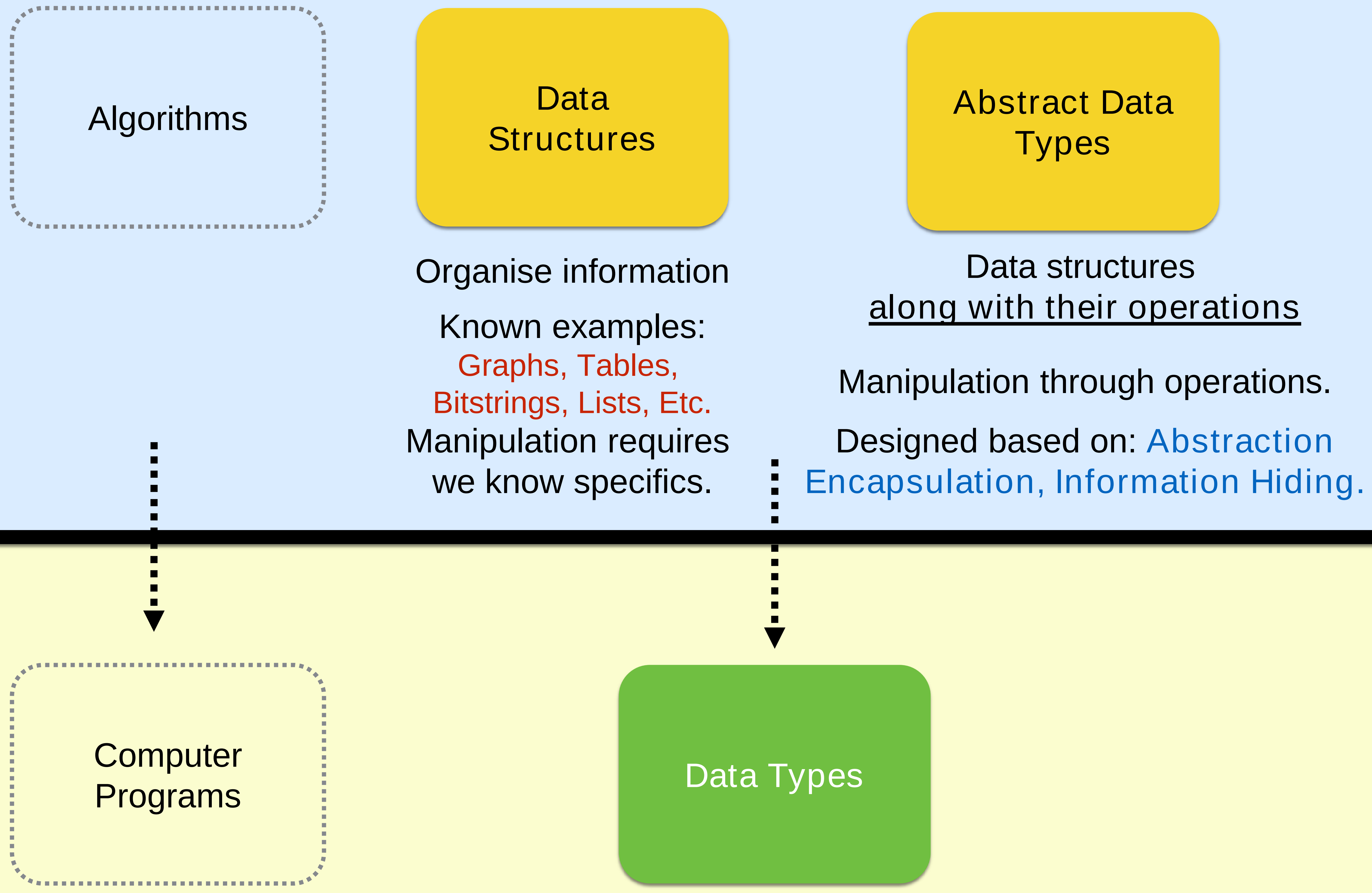
actionmailer	Fix MailerPreview broken tests	14 days ago
actionpack	Merge pull request #13890 from achempion/syntax-error-backtrace	14 hours ago
actionview	Fix date_select option overwriting html classes	5 days ago
activemodel	Fix warning for overshadowing XML variable	2 days ago

2228 contributors

(actually it's over 4400 now)

Abstraction

Concepts & Ideas



Organise information

Known examples:
Graphs, Tables,
Bitstrings, Lists, Etc.
Manipulation requires
we know specifics.

Data structures
<u>along with their operations</u>

Manipulation through operations.

Designed based on: Abstraction
Encapsulation, Information Hiding.

Tangible things.

Abstraction

- Extracting essential information while ignoring the rest.
- Examples:
  - maps
  - football highlights
  - function’s header
- Aim: reduce complexity

Richardson Hall
CSIRO - Clayton Site
Research Way
Beddoe Ave
Boundary Rd
Alliance Ln
College Walk
Jock Marshall Reserve
Monash University Clayton Campus
Chancellors Walk
Scenic Blvd
Ancora Imparo Way
Exhibition Walk
The Alexander Theatre
Oval Rd
Monash University Baseball Club
Monash College Clayton Campus
Rainforest Walk
Wellington Rd
Cobain St
Parker St
Mannix College
Normanby Rd
Monash Sport

It ignores, not hides

Other fields perform abstraction

Cue spherical cow in a vacuum joke

https://upload.wikimedia.org/wikipedia/commons/2/23/SphericalCow2.gif

Encapsulation

- Enclosing items within a container.
  - Containers: bag, loop, function’s body
  - Compartmentalise: Things that belong together go together.

- Aim: defining scope, modularisation and reusability

Encapsulated items do not need to be hidden.

Information Hiding

- Blocking access to information.
- Example:
  - implementation details
- Aim: protect the user, the information itself, and/or its owner

Data Types

- Common concept in lower level languages (C, Java, etc.)
- Refers to a classification that determines:
  - The possible values for that type
  - The meaning of those values
  - The operations that can be done on them
  - The way those values are implemented
- Example: In Java if variable has type int
  - It can take values from -2,147,483,648 to 2,147,483,647
  - Its meaning is that of an integer number
  - Can be used in all integer operations (add, subtract, etc)
  - Implemented using 32 bits and specific bytecode operations
  - There are other kinds of whole numbers (short, long , etc.)

Data Types

- Knowing the implementation can have advantages, e.g.
  - Extra functionality
  - Speed
- Example:
  - Some languages implement False/True as 0/ (>=1)
  - Allow programs to use arithmetic with booleans
- Using the implementation can have disadvantages, e.g.
  - Lack of portability: tied to one implementation.
  - Poor maintenance: e.g., adding functionality hard.

Data Structures

Organise information

Known examples:
Graphs, Tables,
Bitstrings, Lists, Etc.
Manipulation requires
we know specifics.

Abstract Data Types

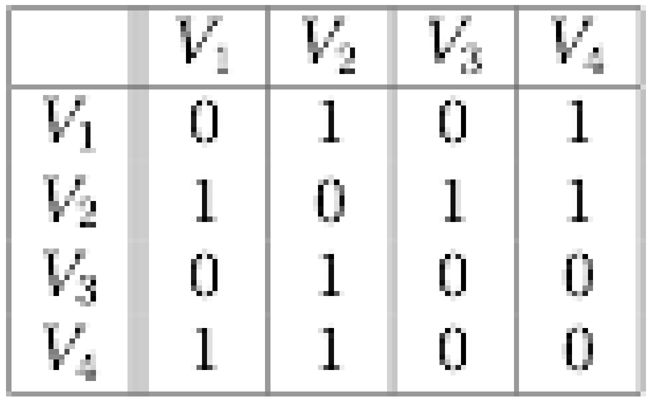
Data structures
→ <u>along with their operations</u> ←

Manipulation through operations.

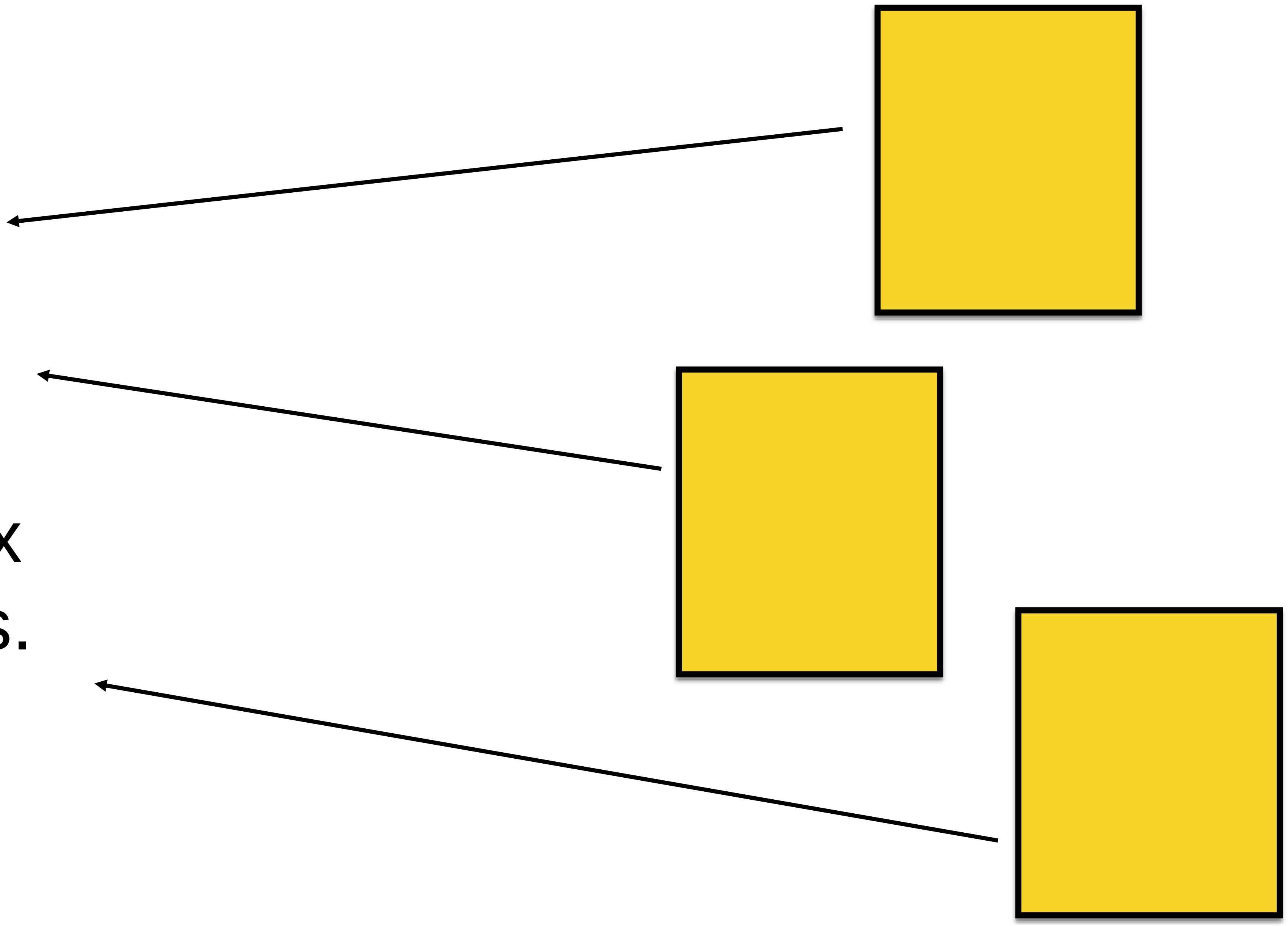
Designed based on: Abstraction
Encapsulation, Information Hiding.

used in different parts of a program

Graph



	V_1	V_2	V_3	V_4
V_1	0	1	0	1
V_2	1	0	1	1
V_3	0	1	0	0
V_4	1	1	0	0

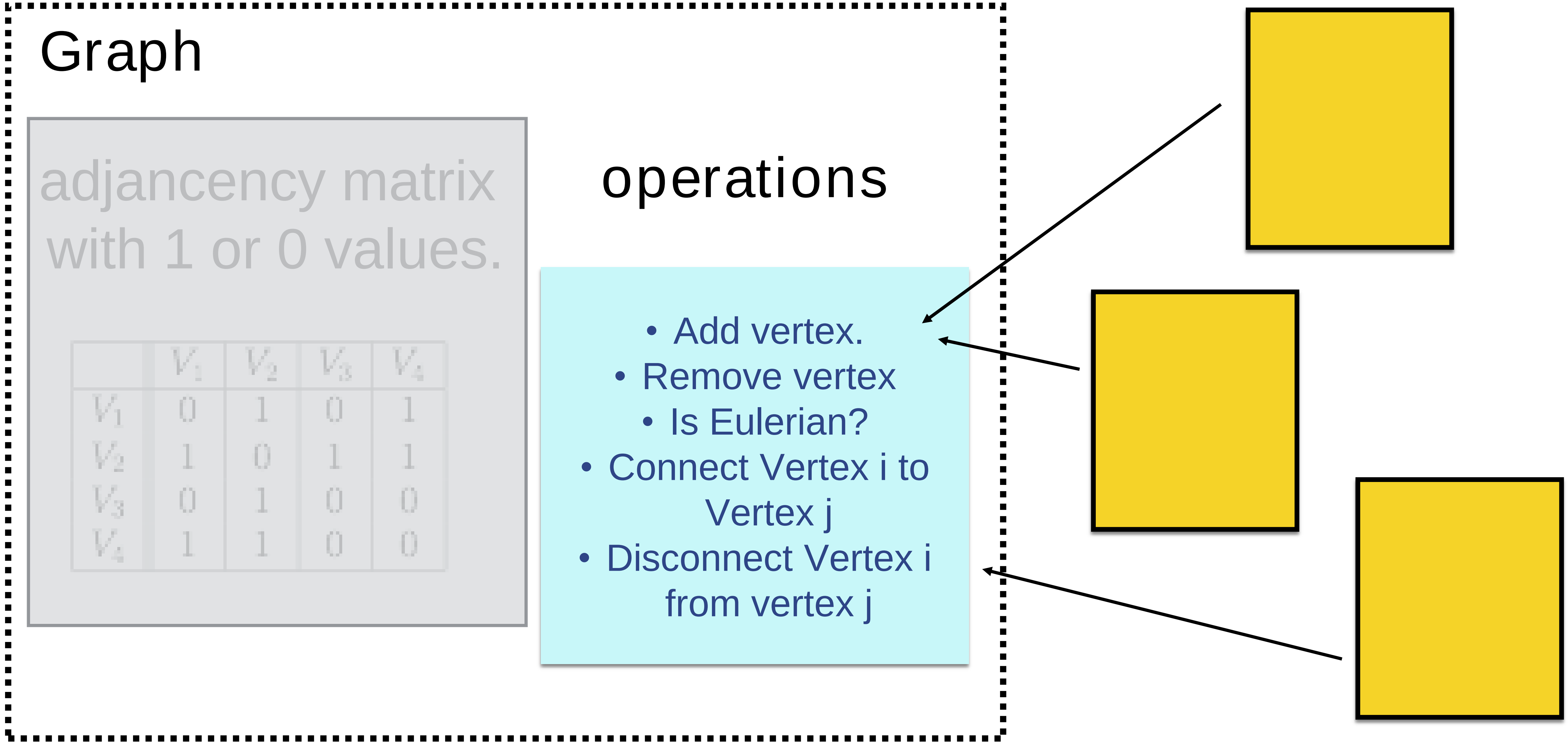


adjancency matrix with 1 or 0 values.

usage requires that I know how my graphs are implemented

as an Abstract Data Type

used in different parts of a program

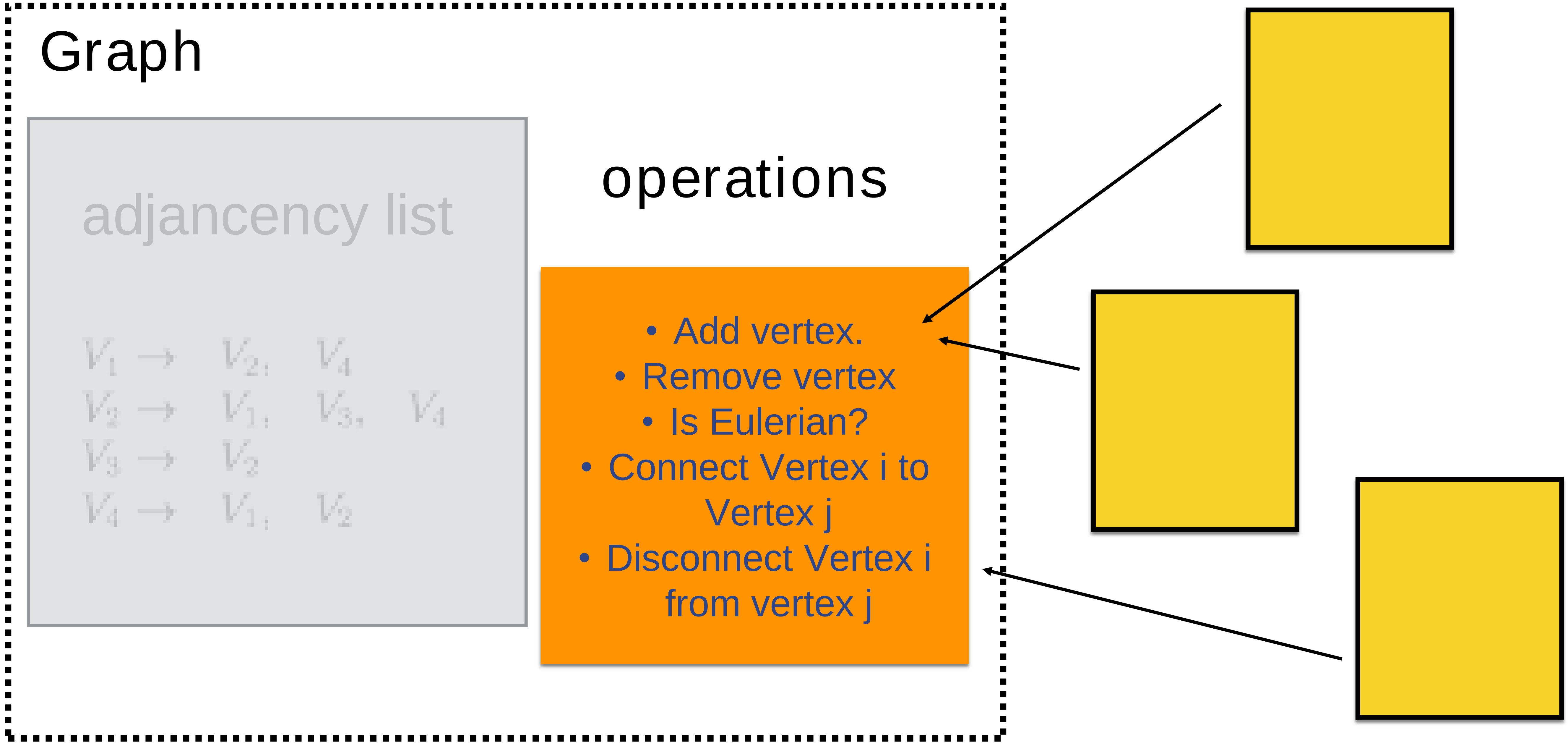

Graph
adjancency matrix with 1 or 0 values.
operations
• Add vertex.
• Remove vertex
• Is Eulerian?
• Connect Vertex i to Vertex j
• Disconnect Vertex i from vertex j

	V_1	V_2	V_3	V_4
V_1	0	1	0	1
V_2	1	0	1	1
V_3	0	1	0	0
V_4	1	1	0	0

usage through operations (a.k.a interface)!

as an Abstract Data Type

used in different parts of a program


Graph
adjacency list
$V_1 \rightarrow V_2, V_4$
$V_2 \rightarrow V_1, V_3, V_4$
$V_3 \rightarrow V_2$
$V_4 \rightarrow V_1, V_2$
operations

- Add vertex.
- Remove vertex
- Is Eulerian?
- Connect Vertex i to Vertex j
- Disconnect Vertex i from vertex j

usage through operations (a.k.a interface)!

No changes outside!

Abstract Data Types (ADTs)

- Often no need to know how types are implemented
  - Data abstraction
- An abstract data type:
  - Provides information regarding:
    - The possible values of the type and their meaning
    - The operations that can done on them
  - BUT not on its implementation, i.e. how:
    - The values are stored
    - The operations are implemented
  - Users interact with the data only through the provided operations

Abstract Data Types (ADTs)

- You have seen (and used) several ADTs in Scribble and Python, such as: Lists, booleans, integers and strings
- Build programs without knowing their implementation: Simplicity.
- The implementation can change without affecting you: Maintenance.
- If several ADTs available, you could easily use any: Flexibility and reusability.
- Different compilers can use different implementations: Portability.

Properties of ADT

- Provides a method for “talking” about a data structure.
- Encapsulate the behaviour of the data structure.
- Hides the implementation details.
- Provides an interface between the user and the data structure.

Benefits

- Programmers can implement a ADT without worrying about affecting the rest of the program.
  - (and change their mind later!)

- Programmers can use the interface without worrying about the implementation.

Data Structures

- At some point we must give ADTs an implementation. Information needs to be organised.

- Some ADTs contain several data fields
  - How do we organise the data? How do we access it?

- That is what a data structure provides:
  - A particular way in which data is physically organised (so that certain operation can be performed efficiently)

- Example: the array data structure
  - Fixed size
  - Data items are stored sequentially
  - Each item occupies the same amount of space

$\Bigg\}$ Physical organisation

This allows constant time access to any element

More terminology…

- We have already talked about
  - Data types
  - Data structures
  - Abstract Data types

- And this is only part of the picture, we also have:
  - Primitive (or built-in) data types versus User-defined
    - Readily available in a given programming language or not
  - Simple (or basic) versus complex ones
    - Single data versus multiple data fields

A way to clarify things a bit

high level

Higher level language (Python, Scribble)	Only ADTs (no details about implementation)	Primitive simple ADTs: integers, booleans,... Primitive complex ADTs: lists, strings, … Non primitive simple/complex ADTs: users can add any ADT they want.
Mid-level (Java)	Primitive data types plus user/library defined ADTs.	Same as below, plus non primitive simple/complex ADTs (implementation is hidden)
Lower level language (C, Fortran)	Primitive data types (both simple and complex). Details of implementation are known.	Primitive simple data types: int, short, float… Primitive complex data types (called data structures): arrays, strings Non primitive simple/complex data types: users can add anything like time, linked lists, array lists…
Assembly language instructions	32-bits registers and a few operations on them	Primitive simple data types: 8-, 16-, 32-bit signed/unsigned integer, float.
Hardware implementation	Bits and logic circuits	No real concept of type: bit, bytes, word …

low level

Sudoku

Each 3x3 box as well as each row and column must containall integers from 1 to 9.
Write a program to find the missing numbers.

					7		4	
			1	8				
	2	7			6	8		
	8						6	
3			4	6	1			5
	5						3	
		3				6	1	
				5	8			
	4		3				2	

Sudoku Puzzle Solver

The components of the program may include:

- Initialiser
- Solver
- Display
- Puzzle

Puzzle Abstract Data Type

- Contains 81 values

DATA STRUCTURE

- Each value is either 1,2,3,4,5,6,7,8, or 9.
- The following interface:
  - Get cell value.
  - Set cell value.
  - Clear cell.
  - Check if all the cells have been filled.
  - Get an empty cell.
  - Check if cell value is legal.

- OPERATIONS

Rational ADT

- Representation of rationals (two integers, p and q, q not zero).

DATA STRUCTURE

- Possible Operations:
  - Arithmetic operations, +, -, /, *
  - Comparison operations, <, >, ==, <=, >=
  - Equality =
  - Read
  - Print

- OPERATIONS

Fundamental Kinds of Abstract Data Types

- Containers
- Dictionaries
- Priority Queues

Container ADTs

- Stores and removes items independent of contents.
- Examples include:
  - List ADT
  - Stack ADT
  - Queue ADT.
- Core operations:
  - add item
  - remove item

List ADT

- Sequence of items
- Possible Operations:
    - Add item
    - Remove item
    - Find item
    - Retrieve item
    - Next item
    - First item
    - Is last item
    - Is empty
    - Print

Stack ADT

- Stack ADT operations
  - Pop
  - Push

Queue ADT

Operations

- Serve
- Append

We will study two implementations: Array-based and Linked

Image @ https://defense1irpu.files.wordpress.com/2014/07/wpid-royston_queue.jpg

Dictionary ADT

- Permits access to data items by content, e.g., a key.

- Operations
  - Search
  - Insert
  - Delete

- Some dictionaries also have the operations
  - Max or Min
  - Next or Previous

Example: A dictionary of student names, whose key is the student ID.

Priority Queue

- Allow items to be processed in a specific order. Each element has a “priority”.

- Operations
  - Insert
  - Min or Max
  - Delete-Min or Delete-Max

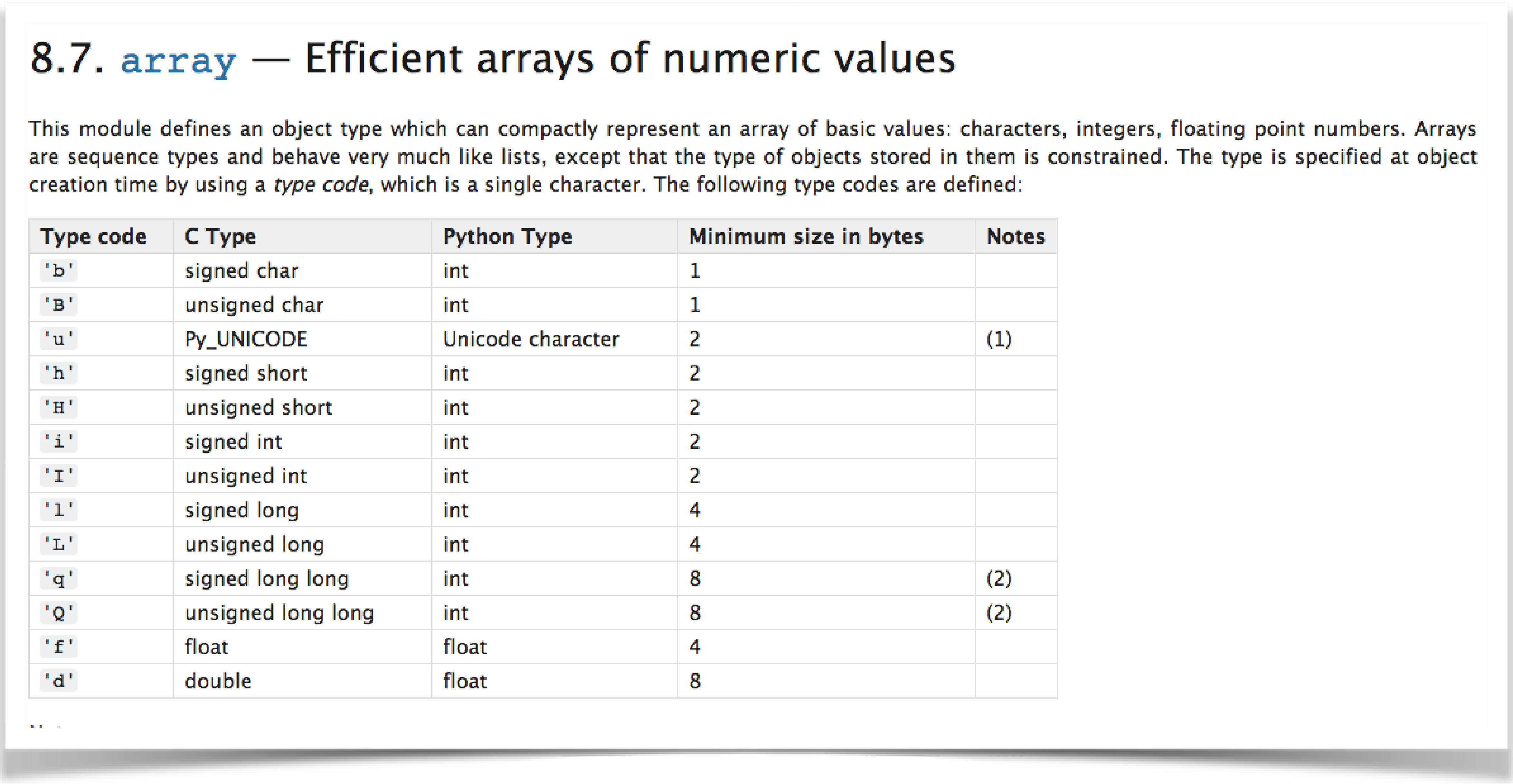
Often implemented using heaps.

Our List ADT

- We will define you own list ADT: Why? They are already in Python!

- Educational value:
  - Learn to implement the operations yourself
  - Reason about the properties of these operations
  - Understand the changes in properties depending on implementation

- What data structure do we use to implement it?We will start with arrays (later, linked nodes)



8.7. `array` — Efficient arrays of numeric values

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a *type code*, which is a single character. The following type codes are defined:

Type code	C Type	Python Type	Minimum size in bytes	Notes
`'b'`	signed char	int	1	
`'B'`	unsigned char	int	1	
`'u'`	Py_UNICODE	Unicode character	2	(1)
`'h'`	signed short	int	2	
`'H'`	unsigned short	int	2	
`'i'`	signed int	int	2	
`'I'`	unsigned int	int	2	
`'l'`	signed long	int	4	
`'L'`	unsigned long	int	4	
`'q'`	signed long long	int	8	(2)
`'Q'`	unsigned long long	int	8	(2)
`'f'`	float	float	4	
`'d'`	double	float	8	

https://docs.python.org/3/library/array.html

List ADT

- Sequence of items

- Possible **Operations**:
    - ➡ Add item
    - ➡ Remove item
    - ➡ Find item
    - ➡ Retrieve item
    - ➡ Next item
    - ➡ First item
    - ➡ Is last item
    - ➡ Is empty
    - ➡ Print

Implementing a List ADT using arrays

- For now we will use Python lists as our arrays
  - After all, they ARE implemented with arrays

- This means our implementation can only use the list operations that are also array operations:
  - Create an array/list
  - Access an element in position P
  - Obtain its length (number of elements already in the array)

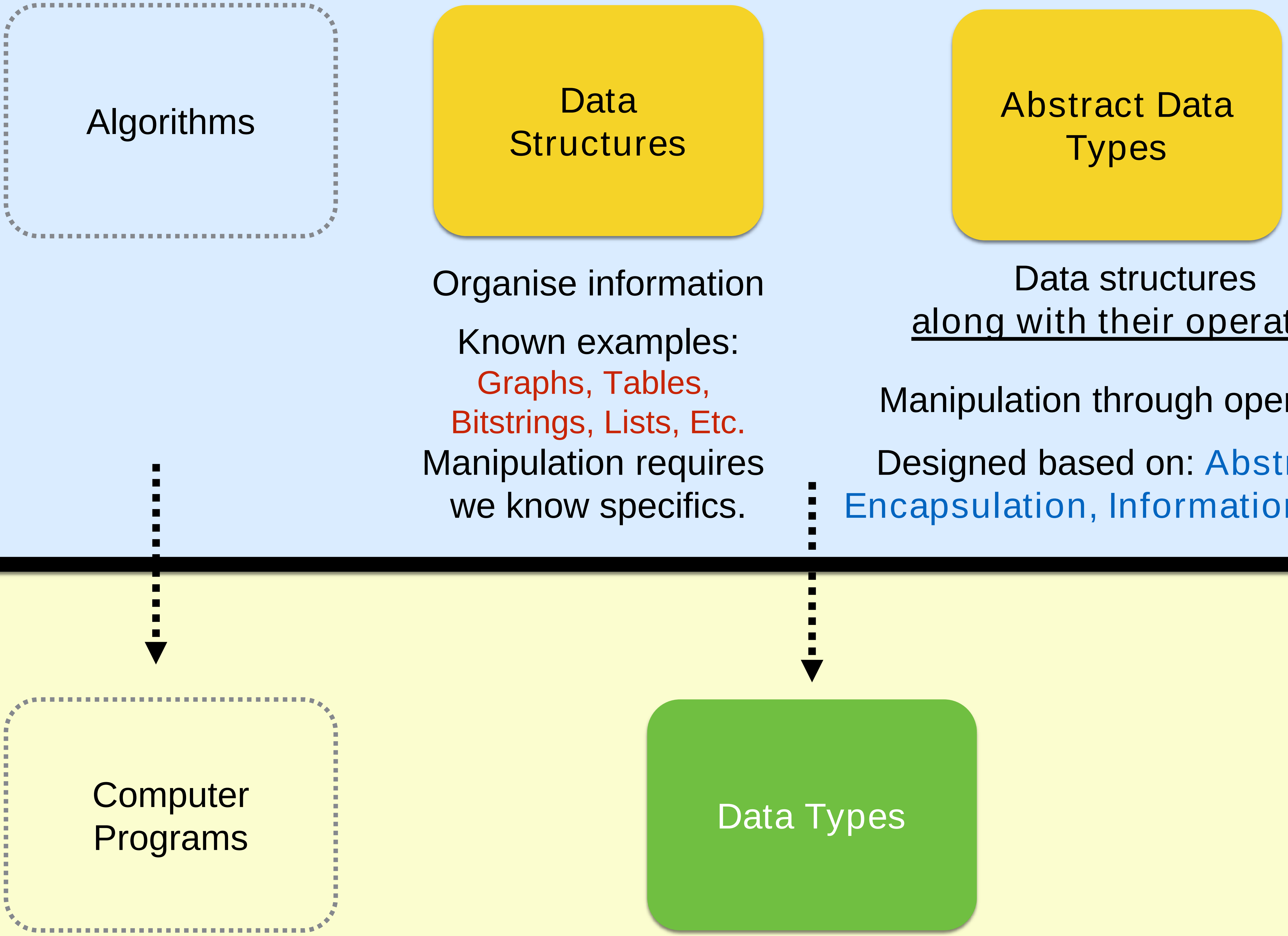
- But this is an ADT. Should we hide the implementation?
  - No!
  - In Python: we trust the programmer to do the right thing
    - For now, don't worry
    - Later, we will see how to avoid users mistakenly accessing things they shouldn't

Implementing your own List ADT

- How do we start?
  - Create a new file (called my_list.py)
  - Add any operation users will need to use.
- What operations?
  - Create a list, access an element, compute the length
  - Determine whether is empty
  - Determine whether it contains a given item
  - Find the position of an item (if in)
  - Add/delete an item
  - Delete/insert the item in position P
- Let’s create also an ADT for sorted lists:
  - Lists whose elements are always sorted (sorted_list.py)
  - Same operations? We will see…

https://cdn.tutsplus.com/net/authors/jeffreyway/holy-grail-of-conference-talks-live-coding.jpg

Concepts & Ideas



Organise information

Known examples:
Graphs, Tables, Bitstrings, Lists, Etc.
Manipulation requires we know specifics.

Data structures
<u>along with their operations</u>

Manipulation through operations.

Designed based on: Abstraction Encapsulation, Information Hiding.

Tangible things.

Summary

- Data Types
- Abstract Data Types
- Data Structures