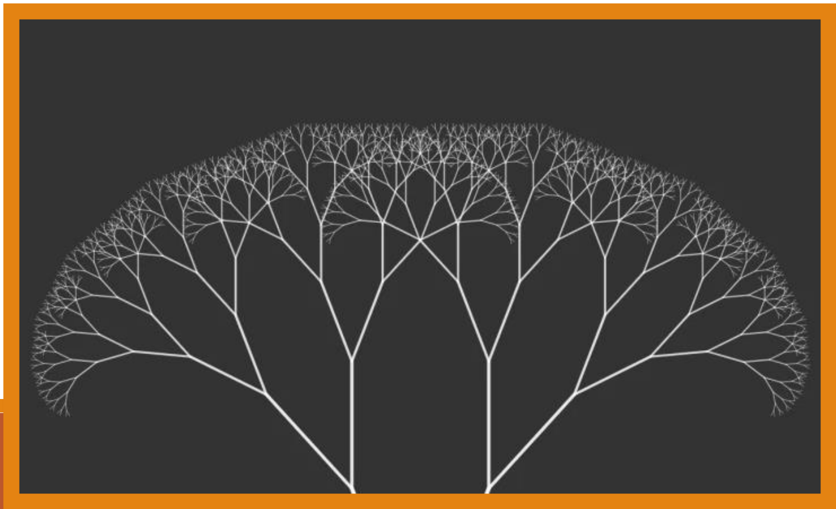
COMPSCI 2C03 WEEK 1

Abstract Data Types

SAM SCOTT, MCMASTER UNIVERSITY, FALL 2023



What is an Algorithm?

What is a Data Structure?

What is an Abstract Data Type?

Whiteboard Slide

What is an Algorithm?

A sequence of well-defined steps for solving a computational problem.

"Algorithms are methods for solving problems that are suited for computer implementation."
- Textbook, page 3

Algorithm design is important!
- Improving your hardware can speed up processing 10 or 100 times.
- Improving your algorithm might speed up processing much more than that.

A procedure to sort a million integers
- ~500 billion operations with **selection sort**
- ~20 million operations with **quicksort** (25 000 times faster)

What is a Data Structure?

A way of storing, organizing, and accessing collections of values in a computer program.

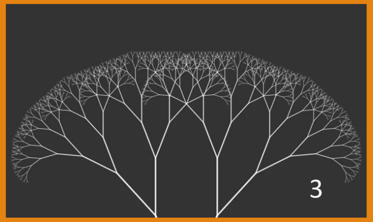
“Data structures are schemes for organizing data that leave them amenable to efficient processing by an algorithm.”

- Textbook, page 3.

Algorithms and data structures are deeply connected.

- For example, search algorithms for arrays, linked lists, trees, and graphs are very different

Data structures and algorithms are used to implement Abstract Data Types (ADTs)



What is an Abstract Data Type (ADT)?

A data type in which operations (behaviors) are defined but implementations are not

"An ADT is a data type whose representation is hidden from the client."
- Textbook, page 64 and 96
- "We use the term ***client*** to refer to a program that calls a method in another library and the term ***implementation*** to describe the Java code that implements the methods in an API."

When implemented, an **ADT** becomes an **API (Application Programmer Interface)** backed by particular data structures and algorithms.

ADTs can be described in many different ways:
- English descriptions
- Mathematical notation
- Pseudo-code
- Etc.

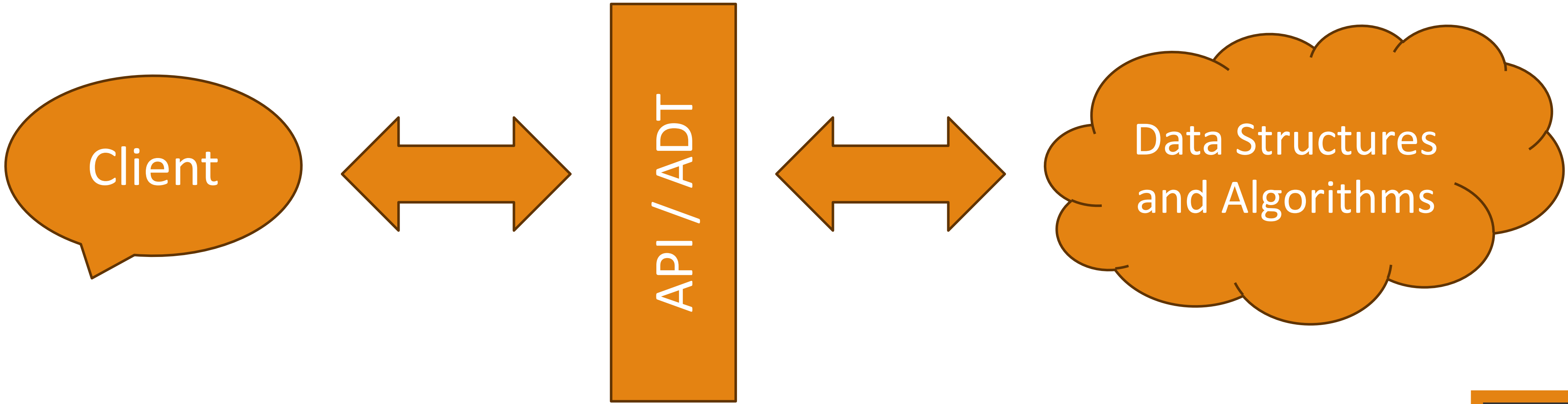
We will tend to use API and ADT interchangeably

Advantages of ADTs/APIs

Precise specification, focus in on we want the data type to do.

Write **client** algorithms without worrying about the **implementation**.

We can change the underlying **data structures** and **algorithms** without disrupting clients.



The Counter ADT

A Counter supports the following operations

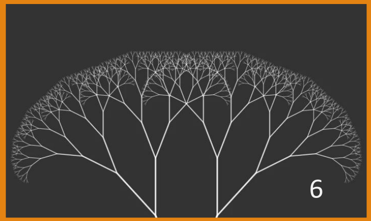
- **Create**: Makes a new counter.
- **Increment**: Adds to the counter.
- **Tally**: Retrieves the number of calls to **increment** since **create**.

Plain English Description

The Counter ADT

create_counter()	returns a new counter
increment(c)	adds to counter c
tally(c)	returns the number of calls to **increment** since **create_counter**

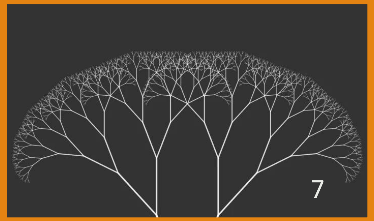
Structured Programming Style Pseudo-code



Structured Client Pseudocode

```
c ← create_counter()
increment(c)
increment(c)
print( tally(c) )
```

Mini Whiteboard



The Counter ADT

Doesn't really mention specific data types.

What could the underlying data model look like? Be creative!

A Python Implementation

```
class Counter:
    count: int = 0


def increment(c):
    c.count += 1

def tally(c):
    return c.count
```

Mini Whiteboard

The Counter ADT (p. 65)

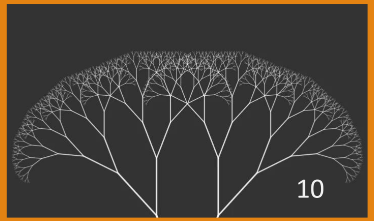
public class Counter

	Counter(String id)	create a counter named id
void	increment()	increment the counter by one
int	tally()	number of increments since creation
String	toString()	string representation

Object-oriented Programming Style

More of an API than an ADT, but who's counting? ☺

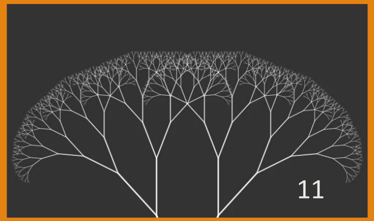
- Specifies data types, includes Java language features.
- See the textbook for an object-oriented Java implementation.
- You can make the Python code object-oriented by simply indenting the functions
  - (And, by convention, the parameter **c** should be renamed to **self**).



Object-Oriented Client Pseudocode

```
c ← Counter()
c.increment()
c.increment()
print( c.tally() )
```

Mini Whiteboard



Activity: Client Code

Write **pseudocode** in any style (structured, object-oriented, etc.) for a function called "**snake_eyes**" that uses a **Counter** to "roll" 2 six-sided dice **n** times (**n** is a **parameter**).

The function should count the number of times the sum of the two dice is equal to 2 and return the result.

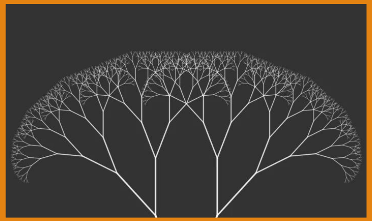
Assume the existence of a function **random(min, max)** that returns a random integer.

Reminder: The **counter** operations are **create**, **increment**, and **tally**.

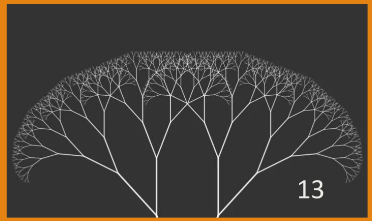
Too easy? Use a data structure to store 11 **Counters** to count all the possible values from 2 through 12. Then return the value that was rolled most often. If there's a tie, return the highest value.

Write **pseudocode** in any style (structured, object-oriented, etc.) for a function called “**snake_eyes**” that uses a **Counter** to “roll” 2 six-sided dice **n** times (**n** is a **parameter**). The function should count the number of times the sum of the two dice is equal to 2 and return the result. Assume the existence of a function **random(min, max)** that returns a random integer.

Reminder: The **counter** operations are **create**, **increment**, and **tally**.



Whiteboard Slide

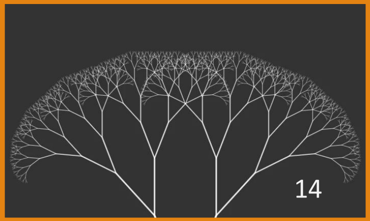


More Examples (pp. 74-83)

The Color ADT/API (partial) as implemented in the JavaFX library.
https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/paint/Color.html

public class Color

static Color	rgb(r, g, b)	returns a new color using given RGB values. Error if not all in [0, 255].
static Color	hsb(h, s, b)	returns a new color using given hue, saturation, brightness. Error if not all in [0, 255].
static Color	web(colorString)	returns a new color using HTML color string. Error if not a legal color string.
double	getRed()	returns the red value of this Color.
double	getHue()	returns the hue value of this Color.
Color	darker()	returns a new color that is a darker version of this Color.
Color	invert()	returns a new color that is an inverted version of this Color.
boolean	equals(obj)	returns **true** if **obj** represents the same color as this Color.



The Color ADT/API

What’s the underlying data model?

What could it be? What are the trade-offs of different models?

This data type is **immutable**. There are no operations that allow a change to the contents of the data type once it’s created.

Most data types we’ll be looking at are **mutable**.

Lots more examples in the textbook, pp. 74-83

That’s All for Now

Important Terminology

- Data Structure
- Algorithm
- ADT
- API
- Client
- Implementation
- Pseudocode
- Structured
- Object-oriented

Next week: Linear Data Structures

- Stack and Queue ADT
- Client algorithms with stacks and queues
- Linked and array implementations

Miscellaneous Notes

- Tutorials and office hours start next week

Credits

- Sedgewick & Wayne, *Algorithms, 4th Edition*.
- Neerja Mhaskar’s *CS 2C03* course slides.