CSCA48 Summer 2017

Week 2 - Abstract Data Types, Stacks & Queues

Brian Harrington

University of Toronto

May 8-12, 2017

UNIVERSITY OF TORONTO SCARBOROUGH

Administrative Details

- Tutorials - Start this week
- Exercise 0 - Due this week
- Practicals/Office Hours/Anti-Lectures - Schedule online

Abstract Data Types

- Data Type: information stored and operations that can be performed
  - We've seen lots of these: str, float, list, dict, etc
- Abstract Data Type: Independent of the implementation

Why ADTs?

- User doesn't care how it works
- Other developers shouldn't **need** to care about implementation details
- Examples:
  - dictionaries
  - lists
  - most things you interact with in the real world

Encapsulation

- ADTs allow us to group together multiple pieces of data, and operations into a single bundle
  - Allows us to think about higher level objects (Events, People, Cars) rather than keeping track of lower level types (strings, ints)
  - Makes our code much easier to read/edit/maintain

Abstraction

- ADTs allow the low-level details to be abstracted away
  - Don't need to care about how the details work
  - Beneficial to both users and programmers
  - Can change one piece without affecting everything
  - Real world: simultaneous delveopment

Parts of an ADT

- Data: What information is held
- Operations: What can we do with that data

Example ADTs

- str (a built in type)
  - Data: A sequence of ASCII characters
  - Operations: `upper, isdigit, replace, ...`
- Person (example from last week)
  - Data: name, age, phone number, address
  - Operations: `set_name, get_address, say_hello`
- Note that we don't care about how data is held/manipulated
  - Does it hold the date of birth and calculate the age? hold the age and update it every year? We don't know and we don't care

Quick Review: Style

- Class Names: `CamelCase`
- Method/Variable Names: `pothole_case`
- Constant Names: `ALL_CAPS_WITH_UNDERSCORES`
- Instance variables/'private' methods: `__surrounding_underscores__`
- PEP-8 Style Guide for Python: `http://www.python.org/dev/peps/pep-0008/`

Quick Review: Classes

- `__init__`
  - Define what happens when a new object gets initialized
- `self`
  - All methods are implicitly passed a pointer to their object called `self`
  - e.g., `my_object.move(5)` $\rightarrow$ `my_object.move(self,5)`

ADTs vs Classes

- For our purposes, an ADT is really just the documentation of the class, before we've written the implementation
- Design -> Document -> Comment -> Code
- ADT is what we have after step 2
- Why is this a good break point?

Break


AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND
BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.
I AM A GOD.

Containers

- We've seen lots of containers already:
  - Lists, Strings, Sets, Dictionaries
  - Defined by what they hold, and how they're accessed
  - Common issue in CS (and real world):
    - We want a container that can hold anything, items added/removed one at a time, order inserted determines order removed

Stacks & Queues

- Stack = Last In First Out (LIFO)
- Queue = First In First Out (FIFO)
- Real life analogues:
  - Plates in a cupboard - Stack
  - Boarding a plane - Stack
  - Undo function - Stack
  - Waiting in line for service - Queue
  - Buffering a video - Queue
  - Controlling requests for CPU time - Queue

Stacks & Queues

- Let's build some ADTs