UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Abstract Data Types

Today's Lecture

- Abstract Data Types
  - More use of functions!
  - Value in documentation and clarity

Reminder: https://go.c88c.org/chat

Computing In the News

The 26 Words That Made The Internet – Section 230 if the Communications Decency Act of 1996

"No provider or user of an interactive computer service shall be treated as the publisher or speaker of any information provided by another information content provider." (47 U.S.C. § 230(c)(1)).

SCOTUS NOT "THE NINE GREATEST EXPERTS ON THE INTERNET" —

SCOTUS "confused" after hearing arguments for weakening Section 230 immunity

SCOTUS sways Google's way, says eroding Section 230 could crash digital economy.

ASHLEY BELANGER - 2/21/2023, 3:23 PM

Attendance: https://go.c88c.org/here

Passcode: sydney

- There are **NO** points involved!
- There are no makeups, no need to let us know if you can't make it… no need to do anything other than submit a form.

Why?

- Coming to class is important!
- Will help us track which times/sessions are most people.
- We're not interested in micromanaging, adding another grading component.

Abstract Data Type

- Uses pure functions to encapsulate some logic as part of a program.
- We rely of built-in types (int, str, list, etc) to build ADTs
- This is a contrast to object-oriented programming
  - Which is coming soon!

Creating Abstractions

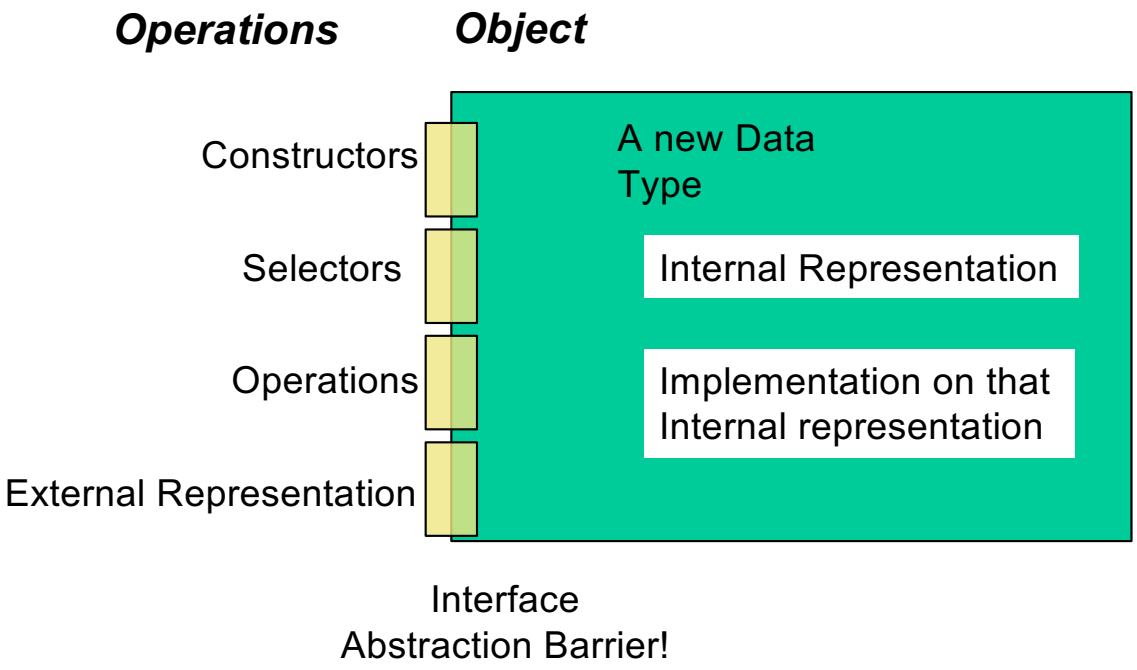
- Compound values combine other values together
  - date: a year, a month, and a day
  - geographic position: latitude and longitude
  - a game board

- Data abstraction lets us manipulate compound values as units
- Isolate two parts of any program that uses data:
  - How data are represented (as parts)
  - How data are manipulated (as units)
- Data abstraction: A methodology by which functions enforce an abstraction barrier between *representation* and *use*

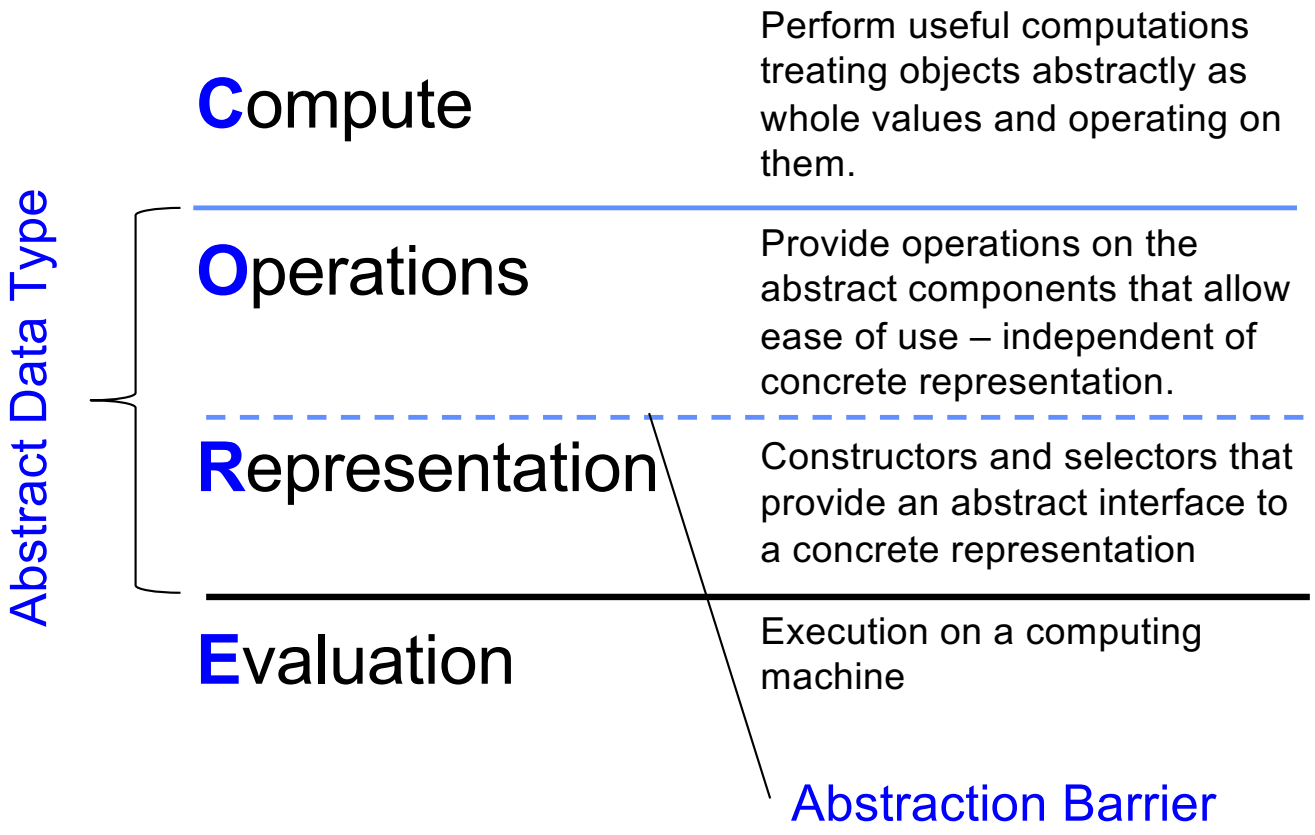
Why Abstract Data Types?

- How do you represent the *idea* of a game board, a "course", a person, a student, anything?
  - Programming languages allow you to do just about anything!
- "Self-Documenting"
  - contact_name(contact)
    - » vs contact[0]
  - "0" may seem clear now, but what about in a week? 3 months?
- Change your implementation
  - Maybe today it's just a Python List
  - Tomorrow: It could be a file on your computer; a database in web

Abstract Data Type



C.O.R.E concepts



Reminder: Lists

- Lists
  - –Constructors:
    - »list(…)
    - »[<exps>,…]
    - »[<exp> for <var> in <list> [if <exp>]]
  - –Selectors: <list> [<index or slice>]
  - –Operations: in, not in, +, *, len, min, max
    - »Mutable ones too (but not yet
- Tuples
  - –A lot like lists, but you cannot edit them. We'll revisit on Monday.

A Small ADT

```
def point(x, y): # constructor
      return [x, y]

x = lambda point: point[0] # selector
y = lambda point: point[1]

def distance(p1, p2): # Operator
      return ((x(p2) - x(p1)**2 + (y(p2) -
y(p1))**2) ** 0.5

origin = point(0, 0)
my_house = point(5, 5)
campus = point(25, 25)
distance_to_campus = distance(my_house, campus)
```

Creating an Abtract Data Type

- Constructors & Selectors
- Operations
  - Express the behavior of objects, invariants, etc
  - Implemented (abstractly) in terms of Constructors and Selectors for the object
- Representation
  - Implement the structure of the object

- An abstraction barrier violation occurs when a part of the program that can use the higher level functions uses lower level ones instead
  - At either layer of abstraction
- Abstraction barriers make programs easier to get right, maintain, and modify
  - Few changes when representation changes

Question: Changing Representations? http://go.c88c.org/10

Question 1.1

Assuming we update our *selectors*, what are valid representations for our `point(x, y)` ADT?

Currently `point(1, 2)` is represented as `[1, 2]`

- A) `[y, x] # [2, 1]`
- B) `“X: ” + str(x) + “ Y: ” + str(y)`
  `# “X: 1 Y: 2”`
- C) `str(x) + ' ' + str(y) # '1 2'`
- D) All of the above
- E) None of the above

A Layered Design Process – Button Up

- **Start with "What do you want to do?"**
- **Build the application based entirely on the ADT interface**
  - **Focus first on Operations, then Constructors and Selectors**
  - *Do not implement them! Your program won't work.*
  - You want to capture the "user's" point of view
- **Build the operations in ADT on Constructors and Selectors**
  - **Not the implementation representation**
  - This is the end of the abstraction barrier.
- **Build the constructors and selectors on some concrete representation**

Example: Tic Tac Toe and Phone Book

- See the companion notebook.
- Download the file "ipynb"
  - Go to datahub.berkeley.edu
  - Log in, then select "Upload"

Question: The Abstraction Barrier

Which of these *violates* a board ADT?

- A) `diag_left = diagonal(board, 0)`
- B) `board[0][2] = 'x'`
- C) `all_rows = rows(board)`
- D) `board = empty_board()`
- E) None of the above