CSC148 fall 2013

abstraction and idiom
week 2

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

September 18, 2013

Computer Science
UNIVERSITY OF TORONTO

Outline

abstract data types (ADTs)

implement an ADT with a class

idiomatic python

common ADTs

In CS we recycle our intuition about the outside world as ADTs. We abstract the data and operations

lists

laundry lists
grocery lists

-
  fname, lnam
  nancy, davo
  erin, bora
  tony, rapha
  ⋮
  names.csv

  sequences of items; can be added, removed, accessed by position

physically impossible

can't get

stacks

-
  specialized list where we only have access to most recently added item

dictionaries

-
  collection of items accessed by their associated keys

stack example

you should click here

visit this visualization of code and step through it

The calls to `first` and `second` are stored on a stack that defies gravity by growing downward

stack class design

Stack

__init__

method.

We'll use this real-world description of a stack for our design:

> *A stack contains items of various sorts. New items are pushed on to the top of the stack, items may only be popped from the top of the stack. It's a mistake to try to remove an item from an empty stack. We can tell how big a stack is, and what the top item is.*

push

pop

top()

Take a few minutes to identify the main noun, verb, and attributes of the main noun, to guide our class design. Remember to be flexible about alternate names and designs for the same class

implementation possibilities

The public interface of our Stack ADT should be constant, but inside we could implement it in various ways

- Use a python list, which already has a pop method and an append method

- Use a python list, but push and pop from position 0

- Use a python dictionary with integer keys 0, 1, . . . , keeping track of the last index used

testing

Use your `docstring` for testing as you develop, but use unit testing to make sure that your particular implementation remains consistent with your ADT's interface. Be sure to:

- import the module `unittest`
- subclass `unittest.Testcase` for your tests, and begin each method that carries out a test with the string `test`
- compose tests before and during implementation

going with the (pep) tide

Python is more flexible than the community you are coding in. Try to figure out what the python way is

- don't re-invent the wheel (except for academic exercises), e.g. `sum`, `set`
- use comprehensions when you mean to produce a new list (tuple, dictionary, set, ...)
- use ternary `iff` when you want an expression that evalutes in different ways, depending on a condition