Happy Thanksgiving, eh!
Abstract Data Types

CSC148, INTRODUCTION TO COMPUTER SCIENCE
DAVID LIU
Interface vs. Implementation
Abstract data types: a common language

Set
Multiset
List
Map
Iterable
Stacks and Queues
Simplicity is powerful

In Python, frames for function calls form a stack.

In Assignment 1, the people on each floor form a queue.
Key phrases for communicating errors

Abstract behaviour
“When this function runs, we expect X to happen, but Y happens instead.”

Concrete prediction
“We can check that Y happens by doing…”

Code-centered explanation
“The problem is at Line n, because…”
Raising exceptions

*How can we *immediately report* to client code that one of our functions was called incorrectly?*
Recall: strategies for handling bad inputs

- Preconditions ("it’s the user’s fault")
- Do nothing ("fail silently")
- Input processing ("fix the problem for them")
Stack.pop
Exceptions

An exception is a special object in Python that represents some kind of error.

Raising an exception is a way to interrupt the normal execution of a program. The exception object is used to report the type of error, and relevant details.
Stack.pop, again
Get to know each other!

If you could instantly become an expert in something, what would it be?
Diversion: balancing parentheses
A string is *balanced* when...

Every ( is followed by a matching ). Nested is allowed.

Balanced

\[ ( ) \quad (1 2 (4 5)) \quad (((4))) \quad ()()()()() \]

Not balanced

\[ (1 )((()) bla())ah) \]
(a(s(d(sdf)safd(s(gdas())))(qwerqwer)sadf)((werqew()df(asd(asdfasdfs)))))()sdfs(d((sfsfsdad)sdfsas))(wer(wer(w)erwe(r))()wqwqqeww())(d(fsdfadssdfs)(())(dsfdfsafdafa)dfsa(dass(fdasqwerwd(d(fas)dfas)dfas)))
Key ideas

Ignore all characters except ( and ).

Keep track of when you see a (, but forget about it when you’ve seen the matching ).
START
END
(1 − (5 × 8) + 6
START
END
\[
\frac{(a \times ((3) + (b))))}{(2 - b)}
\]
START
END
Now design your function!
Evaluating efficiency of implementations

Given multiple implementations of the same interface, what are different ways we can compare them?
Consider another stack implementation
A timing experiment

A common technique used to gain evidence about the efficiency of some code is to run a timing experiment that simply runs the code and see how long it takes to run.

Such experiments often are repeated multiple times for different sizes of data (in our case, stack sizes).
Two fundamental questions

1. Why do Python lists behave this way?

2. How can we talk about running time more precisely, without relying on timing experiments?
Assignment 1 Announcements

Extra office hours starting---check course website

Some updates for python_ta---check A1 handout

FAQ on the course forum!
Efficiency of Python lists

How are Python lists implemented, and what are the implications for the running time of list operations?
A Python list in memory

A Python list stores the ids of its elements in a **contiguous block of memory**.

This is an *array-based* implementation.
A Python list in memory

This makes list indexing take constant time: the running time doesn’t depend on the length of the list.
A Python list in memory

Insertions and deletions must preserve the contiguity of the element ids: elements must be *shifted over*.

Key thing to remember: the front of the list is fixed, while the back can “expand” to take up more or less space.
Worksheet consolidation!

Q1: making sure you understand what just happened.

Q2: putting this into practice. Think of a “step” as a list element being shifted.
Communicating about running time

Goal: communicate how long a function/program takes to run as it is given larger and larger inputs.

In other words, we want to describe running time as a function of input size.
Communicating about running time

We want a way of talking about running time that doesn’t depend on timing experiments or exact step counts.

What we care about is the type of growth:
- logarithmic, linear, quadratic, exponential, etc.
Big-Oh notation

\[ O(n), O(n^2), O(\log n), O(2^n), \ldots \]

**Homework**: read about this!