CSC148: Week 3
http://www.cdf.utoronto.ca/~csc148h/summer/

Sophia Huynh
Summer 2018
Announcements

● Ex2 due tonight @ 11PM
● Ex3 and Lab 3 have been released
● Assignment 1 will be posted on Saturday (due June 17th)
● Ex1 marks have been released
  ○ Remark requests due Saturday @ 11PM
Remark Requests
A bit lengthy and repetitive, but important.
Remark Requests

- **Minor** changes to code are okay
  - E.g. you had a typo somewhere, used the wrong variable name
- No deduction for making changes on exercises
- 5 - 10% deductions for assignments
  - No deduction for legit marking errors
Remark Request: What to write

- Be specific
- Run the tests provided (and client code) to find your mistakes
  - Tell us (specifically) what needs to be changed
- Remark requests that don't give any information will be skipped.
Remark Requests: Steps to Take

1. Download the test file(s)
2. Run the test(s)
3. See what you need to fix in order to pass the tests.
4. Submit a remark request detailing what you had to do in (3).
5. If you think something was marked incorrectly, explain your reasoning.
Good Remark Request

"On line 23 of ex1.py, I used the variable somethingelse instead of something_else. Fixing this would have allowed me to pass the tests. Thanks!"
Skipped Remark Request

"Please re-mark this."
This isn't about anyone in particular: About half the re-mark requests look like this.
Remark Requests

● You have the test cases. Use them!
  ○ The grades shouldn't be a big surprise.
● Remark requests should show that you made some effort to find out where you lost marks/how to fix it/etc.
Outline

● Abstract Data Types
  ○ Stacks
  ○ Queues
Abstract Data Types

● Last week: **Abstract** classes/methods
  ○ "The concept/idea of it is what's important, the implementation can vary."
● **Data types**: strings, ints, lists, etc.
● **Abstract Data Type (ADT)**: Implementation isn't the focus. The concept and 'functionality' is.
Example: Dictionaries

- Maps a key to a value
  - Can get a value given a key
  - Can add a key in and its value

- How this works (how they're stored, etc.) doesn't matter to us when we talk about the type.
  - Matters if we talk about implementation.
Example: Lists

- Keeps items together
  - Can retrieve them
  - Can add items
- How this works doesn't matter when we talk about the type
- The **concept** of a list makes sense
Today's Lecture

● First part: concepts (no code)
  ○ *What* things are supposed to do.
● Second part: implementation
  ○ *How* things actually work.
A university has a student list that should be private. What do you name this attribute?

a) `self.student_list`  
b) `self._student_list`  
d) `self.__student_list`  
c) `self.___student_list___`
Stacks

Like a stack of pancakes, paper, books, etc.
Stack (Data Type)

- Can add to a stack

Our Stack
Stack (Data Type)

- Can add to a stack
Stack (Data Type)

- Can add to a stack
Stack (Data Type)

- Can add to a stack
Stack (Data Type)

- Can add to a stack
Stack (Data Type)

- Can add to a stack

Apple
Chair
Table
Our Stack
Stack (Data Type)

- Can add to a stack

Our Stack

Table
Chair
Apple
Stack (Data Type)

- Can add to a stack

```
Our Stack

Table

Chair

Apple

Pencil
```
Stack (Data Type)

- Can remove from a stack

We remove the top first.
Stack (Data Type)

- Can remove from a stack

![Stack Diagram]

- Pencil
- Apple
- Chair
- Table
- Our Stack
Stack (Data Type)

- Can remove from a stack

```
  Our Stack
     Table
        Chair
           Apple
```
Stack (Data Type)

- Can remove from a stack
Stack (Data Type)

- Can remove from a stack

Table

Our Stack
Stack (Data Type)

- Can remove from a stack
- The first thing we put in our stack is the last thing we take out!
Stacks: Example

- Used for "Undo" buttons
  - E.g. when you type anything and hit undo, it's using a stack.
  - Same goes for art (when you draw something).
Stack

Action Stack
Stack

Action Stack

Draw rectangle

Draw square
Stack

Action Stack

Draw hexagon
thing

Draw rectangle

Draw square
Stack

Action Stack

Draw triangle
Draw hexagon thing
Draw rectangle
Draw square
Stack

Action Stack

- Draw hexagon
- thing
- Draw rectangle
- Draw square
Stack

Action Stack

- Draw rectangle
- Draw square
Stack

Action Stack

Draw square
Stack

Action Stack
Stack

Undo Stack

- Draw triangle
- Draw hexagon
- Draw rectangle
- Draw square

Redo Stack
Stack

Undo Stack

- Draw hexagon
- thing
- Draw rectangle
- Draw square

Redo Stack

- Draw triangle
Stack

Undo Stack

Draw rectangle
Draw square

Redo Stack

Draw hexagon thing
Draw triangle
Stack

Undo Stack

Redo Stack

- Draw square
- Draw rectangle
- Draw hexagon thing
- Draw triangle
Stack

Undo Stack

Redo Stack

- Draw square
- Draw rectangle
- Draw hexagon thing
- Draw triangle
Example: Programming-related

- **Function A**
  - Call B
  - Call C

- **Function B**
  - Call D
  - Call E

- **Function C**

- **Function D**

- **Function E**
Example: Programming-related

Call Stack

Function A
  Call B
  Call C

Function C

Function E

Function B
  Call D

Function D
  Call E
Example: Programming-related

Call Stack

Function A
- Call B
- Call C

Function B
- Call D

Function C

Function D
- Call E

Function E
Example: Programming-related

Call Stack:
- A
- B
- D

Function A:
- Call B
- Call C

Function B:
- Call D

Function C

Function D:
- Call E
Example: Programming-related

Call Stack
- E
- D
- B
- A

Function A
- Call B
- Call C

Function B
- Call D

Function C

Function D
- Call E

Function E
Example: Programming-related

Call Stack

D
B
A

Function A
- Call B
- Call C

Function B
- Call D

Function C

Function D
- Call E

Function E
Example: Programming-related

Call Stack

Function A

Call Stack

Function B

Call D

Function C

Call C

Function D

Call E

Function E
Example: Programming-related

Call Stack

Function A
- Call B
- Call C

Function B
- Call D

Function C

Function D
- Call E

Function E
Example: Programming-related

Call Stack:
- A
- C

Function A:
- Call B
- Call C

Function C

Function B:
- Call D

Function D:
- Call E
Example: Programming-related

Call Stack

Function A

Call B

Call C

Function C

Function D

Call E

Function B

Call D
Example: Programming-related

Call Stack

Function A
  Call B
  Call C

Function B
  Call D

Function C

Function D
  Call E

Function E
Example: Call stack

- Functions that are called are kept in a stack
- We add them to the stack when its called
- Remove it from stack when finished and return to the top of the stack
Stacks

- "Last in First out" (LIFO)
- "First in Last out" (FILO)

(They mean the same thing.)
Queues
Like a ticket queue, bank queue, waiting queue.
Queue

- "First in First out" (FIFO)
- "Last in Last out" (LIFO)
- Usually used to refer to a line of sorts (people enter it, wait, exit in the order they queued up.)
Queue (Data Type)

- Can add to a stack
Queue (Data Type)

- Can add to a stack

Front of our Queue

Apple
Queue (Data Type)

- Can add to a stack

Front of our Queue

Apple  Book
Queue (Data Type)

- Can add to a stack

Front of our Queue

Apple Book Pen
Queue (Data Type)

- Can add to a stack

Front of our Queue

Apple  Book  Pen  Pencil
Queue (Data Type)

- Remove in the order they were added

Front of our Queue

- Apple
- Book
- Pen
- Pencil
Queue (Data Type)

- Remove in the order they were added

Front of our Queue

Apple

Book

Pen

Pencil
Queue (Data Type)

- Remove in the order they were added
Queue (Data Type)

- Remove in the order they were added
Queue (Data Type)

- Remove in the order they were added

Front of our Queue

Pencil
Queue (Data Type)

- Remove in the order they were added
Examples

- Use them when we want to process things in a certain order
Examples

● Use them when we want to process things in a certain order

You are currently 1st in the queue.
Estimated wait: 10 hours
Example: Print Queues

- "A.pdf" is sent to be printed
- "B.pdf" is sent to be printed
- "C.pdf" is sent to be printed

Prints in the order "A.pdf", then "B.pdf", then "C.pdf".
Stacks and Queues

- **Queues**: First thing added is the first thing removed
- **Stacks**: First thing added is the last thing removed.
We have a stack and add "Lemon", then "Marker", then "Ruler", and then "Box". When we remove from the stack, what do we get?

A) Lemon
B) Marker
C) Ruler
D) Box
Answer:
D) Box
We have a *queue* and add "Lemon", then "Marker", then "Ruler", and then "Box". When we remove from the stack, what do we get?

A) Lemon  
B) Marker  
C) Ruler  
D) Box
Answer:
A) Lemon
Break time!
For about 10 minutes.
Afterwards: More complicated example
Stacks vs Queues: Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

- Add everything to a Stack/Queue
- Remove from it
- If we have an int, print it.
- If we have a list, add its content to the Stack/Queue
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top
Stack Example

\[[1, [2, 3, [4, 5], 6], 7, [[[8], 9]]]\]
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top
Stack Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Top
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

[[8], 9] 7 [2, 3, [4, 5], 6] 1

[8] 9
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

9
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 9
Stack Example

$[1, [2, 3, [4, 5], 6], 7, [[8], 9]]$

Printed: 9
Stack Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 9
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top

Printed: 9, 8
Stack Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8
Stack Example

$[1, [2, 3, [4, 5], 6], 7, [[8], 9]]$

Top

Printed: 9, 8, 7
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 9, 8, 7
Stack Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 9, 8, 7
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9, 8, 7
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7
Stack Example

```
[1, [2, 3, [4, 5], 6], 7, [[8], 9]]
```

<table>
<thead>
<tr>
<th>6</th>
<th>[4, 5]</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

Printed: 9, 8, 7
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]\]

Printed: 9, 8, 7
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9, 8, 7, 6
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 9, 8, 7, 6
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9, 8, 7, 6
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9, 8, 7, 6
Stack Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Top

Printed: 9, 8, 7, 6
Stack Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Printed: 9, 8, 7, 6
Stack Example

$[1, [2, 3, [4, 5], 6], 7, [[8], 9]]$

Printed: 9, 8, 7, 6, 5
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7, 6, 5, 4
Stack Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7, 6, 5, 4, 3
Stack Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7, 6, 5, 4, 3, 2
Stack Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 9, 8, 7, 6, 5, 4, 3, 2, 1
Stack Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 9, 8, 7, 6, 5, 4, 3, 2, 1
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

1

Top
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

1  [2, 3, [4, 5], 6]
Queue Example

\[\left[1, \left[2, 3, \left[4, 5\right], 6\right], 7, \left[[8], 9\right]\right]\]
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

<table>
<thead>
<tr>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>[2, 3, [4, 5], 6]</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>[[8], 9]</td>
</tr>
</tbody>
</table>
Queue Example

\[[1, \[2, 3, [4, 5], 6], 7, [[8], 9]]\]

Top

Printed:
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top

Printed: 1
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Printed: 1
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 1
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1
Queue Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 1
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

7  [[8], 9]  2  3  [4, 5]  6

Top

Printed: 1
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 1
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Top

Printed: 1, 7
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Printed: 1, 7
Queue Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7
Queue Example

$[1, [2, 3, [4, 5], 6], 7, [[8], 9]]$

Printed: 1, 7
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 1, 7
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Top

Printed: 1, 7
Queue Example

\[
[1, [2, 3, [4, 5], 6], 7, [[8], 9]]
\]

Top

Printed: 1, 7, 2
Queue Example

\[[1, [2, 3, [4, 5], 6], 7, [[8], 9]]\]

Printed: 1, 7, 2, 3
Queue Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7, 2, 3
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 1, 7, 2, 3
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7, 2, 3
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7, 2, 3
Queue Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

[8] 9 4 5

Top

Printed: 1, 7, 2, 3, 6
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 1, 7, 2, 3, 6
Queue Example

\[\{1, [2, 3, [4, 5], 6], 7, [[8], 9]\}\]

Printed: 1, 7, 2, 3, 6
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top

Printed: 1, 7, 2, 3, 6
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top

Printed: 1, 7, 2, 3, 6
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Printed: 1, 7, 2, 3, 6, 9
Queue Example

\[ [1, [2, 3, [4, 5], 6], 7, [[8], 9]] \]

Top

Printed: 1, 7, 2, 3, 6, 9, 4
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7, 2, 3, 6, 9, 4, 5
Queue Example

[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Top

Printed: 1, 7, 2, 3, 6, 9, 4, 5, 8
Queue Example

\[1, [2, 3, [4, 5], 6], 7, [[8], 9]]

Printed: 1, 7, 2, 3, 6, 9, 4, 5, 8
Queue Example

[1, ____________, 7, ____________ ]
[2, 3, ____________, 6] [ ____________, 9]
[4, 5] [8]

Printed: 1, 7, 2, 3, 6, 9, 4, 5, 8
Queue Example

Stack: 9, 8, 7, 6, 5, 4, 3, 2, 1

Queue: 1, 7, 2, 3, 6, 9, 4, 5, 8
Stack vs Queue

Stack [Top]

Queue [Front]
Stack vs Queue

Stack [Top]  Queue [Front]

Pineapple
Stack vs Queue

Stack  [Top]
Pen
Pineapple

Queue  [Front]
Stack vs Queue

Stack [Top]

Pineapple

Pen

Queue [Front]
Stack vs Queue

Stack [Top]

Pen

Pineapple

Queue [Front]
Stack vs Queue

Stack
[Top]

Queue
[Front]

Apple

Pen
Pineapple
Stack vs Queue

Stack [Top]
- Pen
- Pineapple

Queue [Front]
- Apple
- Pen
Stack vs Queue

Stack [Top]

Pen
Pineapple

Queue [Front]

Pen
Apple
Stack vs Queue

Stack [Top]:
- Pen
- Pineapple

Queue [Front]:
- Apple
- Pen
Stacks vs Queues

- **Stacks**: Last in First Out (LIFO)
  - Or First in Last Out (FILO)

- **Queues**: First in First Out (FIFO)
  - Or Last in Last Out (LILIO)
Break time!
For about 10 minutes.
Afterwards: How to implement Stacks/Queues
Implementing Stacks

● Write a Class for Stack
● Methods:
  ○ Add to the Stack
  ○ Remove from the Stack
  ○ Find out if the Stack is empty

Note: Avoid the name "stack.py" for a file!
Contents of a Stack

- Make it a private variable
  - Lots of choices for this!
- Might want to make it more efficient later -- not possible if it's a public variable!
```python
init

def __init__(self):
    self._content =
```
def __init__(self):
    self._content = []
def add(self, item):
def add(self, item):
    self._content.append(item)
remove

def remove(self):

def remove(self):
    return self._content.pop()
Alternate wording: "we pop from the stack". I'll use 'remove' to keep it consistent with queues.
def is_empty(self):
def is_empty(self):
    return self._content == []
Alternative implementations

- Add the start of the list \( \text{insert}(0) \) and remove from the start \( \text{pop}(0) \)
- Use a dictionary with numbers as keys, track our current index and adjust.
Adding to the Start vs End

- Difference in efficiency
- More on efficiency towards the end of the course
Lists in Python

<table>
<thead>
<tr>
<th></th>
<th>id1</th>
<th>id2</th>
<th>id3</th>
<th>id4</th>
</tr>
</thead>
</table>
### Lists in Python

**my_list**: starts at id1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>
Lists

If we want to get my_list[2], we add 2 to where my_list starts: id1 + 2 ids over = id3

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>

my_list: starts at id1
Lists in Python: Add to End

my_list: starts at id1
Lists in Python: Add to End

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>'A'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>id4</td>
</tr>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>

**my_list:** starts at id1
Lists in Python: Add to Start

```
my_list: starts at id1
```

1 2 3

id1 id2 id3 id4
Lists in Python: Add to Start

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>

my_list: starts at id1
Lists in Python: Add to Start

my_list: starts at id1
**Lists in Python: Remove from End**

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>'A'</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>

**my_list**: starts at id1
```
Lists in Python: Remove from End

my_list: starts at id1
Lists in Python: Add to Start

<table>
<thead>
<tr>
<th>'A'</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
</tr>
</tbody>
</table>

**my_list:** starts at id1
Lists in Python: Add to Start

my_list: starts at id1
Lists in Python: Add to Start

my_list: starts at id1
Efficiency

- Adding and removing from the end of a stack is more 'efficient'
  - Takes fewer steps
- We can implement stacks without lists too!
  - One way will be shown next week.
Implementing Queues

- Similar to Stacks in implementation
- Methods:
  - Add to the Queue
  - Remove from the Queue
  - Find out if the Queue is empty

Note: Avoid the name "queue.py" for a file! Your PythonTA will break!
__init__

def __init__ (self):
    self._content = []

Same as a Stack
Adding and Removing from a Queue

- Add to the **end** of the list; remove from the **start**
  - Faster for adding, slower for removing
- Add to the **start** of the list; remove from the **end**
  - Faster for removing, slower for adding
def add(self, item):
    self._content.append(item)
def remove(self):
def remove(self):
    return self._content.pop(0)
def is_empty(self):
    return self._content == []
Stacks and Queues

● Same method names and concept
  ○ Different implementations

● Could write a parent class: `Container`
  ○ Most things would raise a `NotImplementedError`
Course Logistics

- Lectures usually end before 9
  - I still cover all the material, though
- **Reason:** I'm skipping over the live coding demos
Slides vs Live Coding

● Pros of Slides:
  ○ You have slides to follow and review
  ○ Less fussing with switching views
  ○ Finishing early = You can ask me questions and not stay until 10PM

● Cons of Slides:
  ○ Pacing can get fast/difficult to follow
  ○ Don't get to see how the code is written and common mistakes
Give me Feedback!

- Will send out a survey later
- Do you prefer slides? Live programming?
- How's the pacing? Do you like/dislike class ending early?
- Feedback on exercises, labs, etc.
Give me Feedback!

● What material aren't you comfortable with yet?
  ○ Next week I'll talk about the midterm and have some review. This will give me an idea of what to focus on!
● Let me know what to adjust! :)

Homework

- **Reminder: Ex2 is due at 11PM.**
  - Remember to submit your files on MarkUs!
  - Run PythonTA and the client code before submitting!
- Ex3 is out and due next week at 11PM
- Assignment 1 out on Saturday