CS 61A SU26

# Lecture 15: Mutable Trees

July 16, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

# Potpourri

10 min

- Announcements
- Computing in the News
- WWPD: Attribute Lookup with Inheritance

# Announcements (1 of 2)

- **Midterm grades released**
  - Due to the difficulty of the exam, we gave everyone +6 pts for free (capped at 60 points total)
  - Check what you got right/wrong and **why** (blank/solution PDFs posted and walkthrough videos are coming)
  - Regrade requests have opened and close Sun 7/19 at 11:59 pm
    - Note that Gradescope will be down from 8 AM - 2 PM on Sat 7/18 for scheduled maintenance
  - If you'd like to talk to a head TA for advice, you can sign up for Student Support Meetings. We are also planning to add Exam Support Meetings, stay tuned!
  - My instructor tea hours and OH are also open (but not private)!

# Announcements (2 of 2)

- Due to the **Soda → Gateway building move**, all floors of Soda except the 2nd floor will be closed and you may notice movers working **starting Mon 7/20**
  - Labs will still happen in their normal locations
  - **Some discussion sections will be moving** to Grimes Engineering Center or McLaughlin Hall, make sure you check this Ed post
  - The Gateway building won't be open to the public until sometime in August :(

# Computing in the News

- The U.S. spent $30 billion to ditch textbooks for laptops and tablets: The result is the first generation less cognitively capable than their parents (Fortune)
- Kaiser nurses say technology is making their jobs — and patient care — worse (CalMatters)
    - More on workplace surveillance (requires NYT subscription)
- Data Centers to Add Billions in Power Costs in 13 States (NY Times, gift article)

# WWPD: Attribute Lookup with Inheritance (1 of 3)

```
class A:
    foo = 0
    def __init__(self, foo, bar):
        self.foo = foo + A.foo
        A.foo += 1
        self.bar = bar

class B(A):
    foo = 5
    def __init__(self, bar):
        super().__init__(B.foo, bar)
```

```
>>> first = A(2, 3)
>>> first.foo
2
>>> first.bar
3
>>> A.foo
1
```

# WWPD: Attribute Lookup with Inheritance (2 of 3)

```
class A:
    foo = 0
    def __init__(self, foo, bar):
        self.foo = foo + A.foo
        A.foo += 1
        self.bar = bar

class B(A):
    foo = 5
    def __init__(self, bar):
        super().__init__(B.foo, bar)
```

```
>>> second = A(2, 3)
>>> second.foo
3
>>> second.bar
3
>>> A.foo
2
```

# WWPD: Attribute Lookup with Inheritance (3 of 3)

```
class A:
    foo = 0
    def __init__(self, foo, bar):
        self.foo = foo + A.foo
        A.foo += 1
        self.bar = bar

class B(A):
    foo = 5
    def __init__(self, bar):
        super().__init__(B.foo, bar)
```

```
>>> third = B(2, 3)
Error
>>> third = B(3)
>>> third.foo
7
>>> third.bar
3
>>> B.foo
5
>>> A.foo
3
>>> third.foo = 100
>>> third.foo
100
>>> B.foo
5
```

# Inheritance (Cont'd)

15 min

- Implementing Iterators
- Exception Classes
- Special Methods

# Implementing Iterators (1 of 3)

To implement an iterator for a particular class, you must implement these special methods:

- `__iter__` - called when `iter` is called on an instance
- `__next__` - called when `next` is called on an instance

# Implementing Iterators (2 of 3)

```
class Evens:
    """Iterator that returns positive even numbers <= n"""
    def __init__(self, n: int):
        self.curr = 0
        self.n = n
    def __iter__(self):
        return self
    def __next__(self):
        if self.curr <= self.n:
            temp = self.curr
            self.curr += 2
            return temp
        else:
            raise StopIteration
```

# Implementing Iterators (3 of 3)

```
>>> evens = Evens(5)
>>> evens_iter = iter(evens)
>>> next(evens_iter)
0
>>> next(evens_iter)
2
>>> next(evens_iter)
4
>>> next(evens_iter)
StopIteration
```

# Exception Classes (1 of 4)

What if you want to have a custom exception class? All you have to do is inherit from `Exception`!

# Exception Classes (2 of 4)

```
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            raise InsufficientFundsException(amount, self.balance)
        self.balance = self.balance - amount
        return self.balance
```

# Exception Classes (3 of 4)

```
class InsufficientFundsException(Exception):
    def __init__(self, amount, balance):
        message = f'Requested ${amount} is larger than current balance ${balance}'
        super().__init__(message)

        self.amount = amount
        self.balance = balance
```

# Exception Classes (4 of 4)

```
>>> acct = Account('Jane')
>>> acct.deposit(15)
>>> try:
...     acct.withdraw(20)
... except InsufficientFundsException as ex:
...     print(f'You need ${ex.amount - ex.balance} more')
...
You need $5 more
```

# Special Methods

In summary, the special methods you should know so far are:

- `__init__` - the constructor of a class
- `__eq__` - implements the `==` operator
- `__str__` - returns the object's `str` representation
- `__repr__` - returns the object's `repr` representation
- `__iter__` - returns an iterator for the object
- `__next__` - returns what would be given when calling `next` on an iterator of the object

There's a lot more! You don't need to know all of them for this class.

# Mutable Trees

15 min

- Recall: Tree ADT
- Tree Class
- Fib Tree Revisited

# Recall: Tree ADT (1 of 2)

```
def tree(root_label, branches=[]):
    for branch in branches:
        assert is_tree(branch), 'branches must be trees'
    return [root_label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
```

# Recall: Tree ADT (2 of 2)

```
def is_tree(tree):
    if type(tree) != list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True

def is_leaf(tree):
    return not branches(tree)
```

# Tree Class (1 of 2)

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches
```

# Tree Class (2 of 2)

```
>>> t = Tree(3, [Tree(2, [Tree(5)]), Tree(4)])
>>> t.label
3
>>> t.branches[0].label
2
>>> t.branches[1].is_leaf()
True
```

# Fib Tree Revisited

```
def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n - 2)
        right = fib_tree(n - 1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

```
def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n - 2)
        right = fib_tree(n - 1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])
```

# Practice

**(reminder to self to use high contrast theme)**

30 min

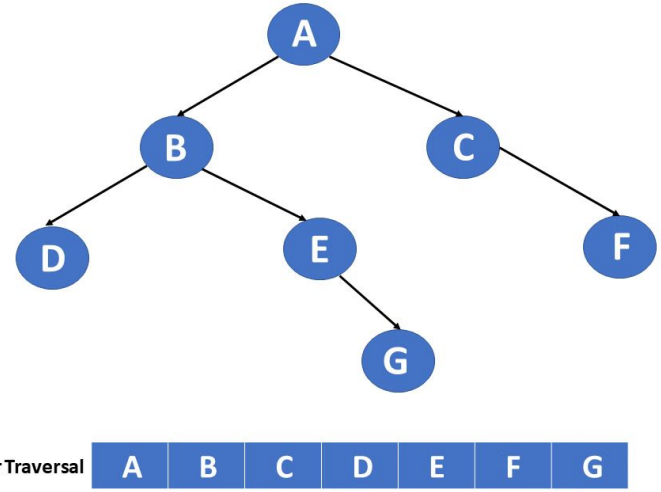
- Height
- Tree Iterator
- Bracket Breaker II

# Height

- Implement `height`, which returns the height of a `Tree`
  - **Hint**: Draw out the trees from the doctests
- Download starter code `15.py` from the course website under Lecture 15
- Run doctests with `python3 -m doctest 15.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

# Tree Iterator

- Implement `TreeIterator`
  - **Hint**: Draw out the trees from the doctests (they're the same as the trees from the `height` problem)
  - **Hint**: See the diagram for an example of a level-order traversal
- Download starter code `15.py` from the course website under Lecture 15
- Run doctests with `python3 -m doctest 15.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?



# Bracket Breaker II

- Implement `reveal_winner` (and `prune_lowest_leaves`), except with the `Tree` class instead of the `tree` ADT
  - See the full problem description from the midterm exam
  - The doctests are identical to diagrams 1-4 on the exam
  - See the comments for guiding questions/hints
- Download starter code `15.py` from the course website under Lecture 15
- Run doctests with `python3 -m doctest 15.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?