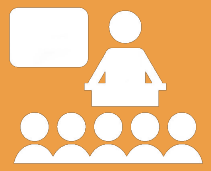
Katya Stukalova
Jongmin Jerome Baek

SUMMER 2016
FINAL REVIEW

Topics we will cover



OOP: 10min

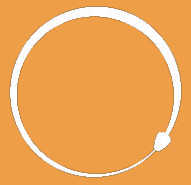
Nonlocal: 5min

Mutation: 5min

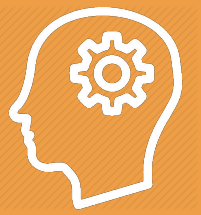
Interfaces: 5min

Recursive Objects: 20min

Scheme: 20min



Tail Recursion: 5min



Logic: 5min

0.
Object Oriented Programming

Meaningful chunks of data

OOP Reminders

- Class attributes
  - Belongs to class
  - All instances of the class share one class attribute
- Instance attributes
  - Belongs to instance
  - Not shared, each instance has its own
- Local variables
  - Exists only inside a frame

What Would Python Display?

```
class Animal(object):
    def __init__(self, health):
        self.health = health
class Dog(Animal):
    health = 9
>>> lassie = Dog(3)
>>> lassie.health
3
>>> Dog.health
9
>>> Animal.health
Error
```

(Credit: Andrew Huang)

- `lassie.health` is 3 because
  - `__init__` is not defined for `Dog`, so `Dog` uses `Animal`'s `__init__`.
  - If an instance attribute and a class attribute have the same name, the instance attribute takes precedence here, because `lassie` is an instance of `Dog`.
- `Dog.health` is 9 because it explicitly asks for the class attribute.
- `Animal.health` is not defined; inheritance goes from parent to child, not from child to parent.

Spot the Errors

```
class Cat(Pet):
  def __init__(self, name, yob, lives=9):
    Pet.__init__(self, name, yob)
    self.lives = 9
  def talk():
    print('meow')
```

(Credit: Andrew Huang)

Spot the Errors

```
class Cat(Pet):
  def __init__(self, name, yob, lives=9):
    Pet.__init__(self, name, yob)
    self.lives = 9 #need self.lives = lives
  def talk(): #need the parameter "self"
    print('meow')
```

(Credit: Andrew Huang)

Barking Up the Wrong Tree

Brian defined the following class:

```
class Dog(object):
    def bark(self):
        print("woof!")
```

One day Marvin wants his dog to bark differently.

```
>>> fido = Dog()
>>> fido.bark = "bow wow!"
```

Brian points out that this won't work, since bark is a method, not a string. Marvin tries to restore his mistake.

```
>>> fido.bark = Dog.bark
```

Barking Up the Wrong Tree

```
class Dog(object):
    def bark(self):
        print("woof!")
>>> fido = Dog()
>>> fido.bark = "bow wow!"
>>> fido.bark = Dog.bark
```

Concerning the last line of code, which of the following statements are True?

(1) Executing this assignment statement will cause an error.

(2) After this assignment, invoking fido.bark() will cause an error.

(3) This assignment statement will have no effect at all.

(4) None of the above criticisms are valid. Everything will be fine.

Barking Up the Wrong Tree

```
class Dog(object):
    def bark(self):
        print("woof!")
>>> fido = Dog()
>>> fido.bark = "bow wow!"
>>> fido.bark = Dog.bark
```

Concerning the last line of code, which of the following statements are True?

(1) Executing this assignment statement will cause an error.

(2) After this assignment, invoking fido.bark() will cause an error.

(3) This assignment statement will have no effect at all.

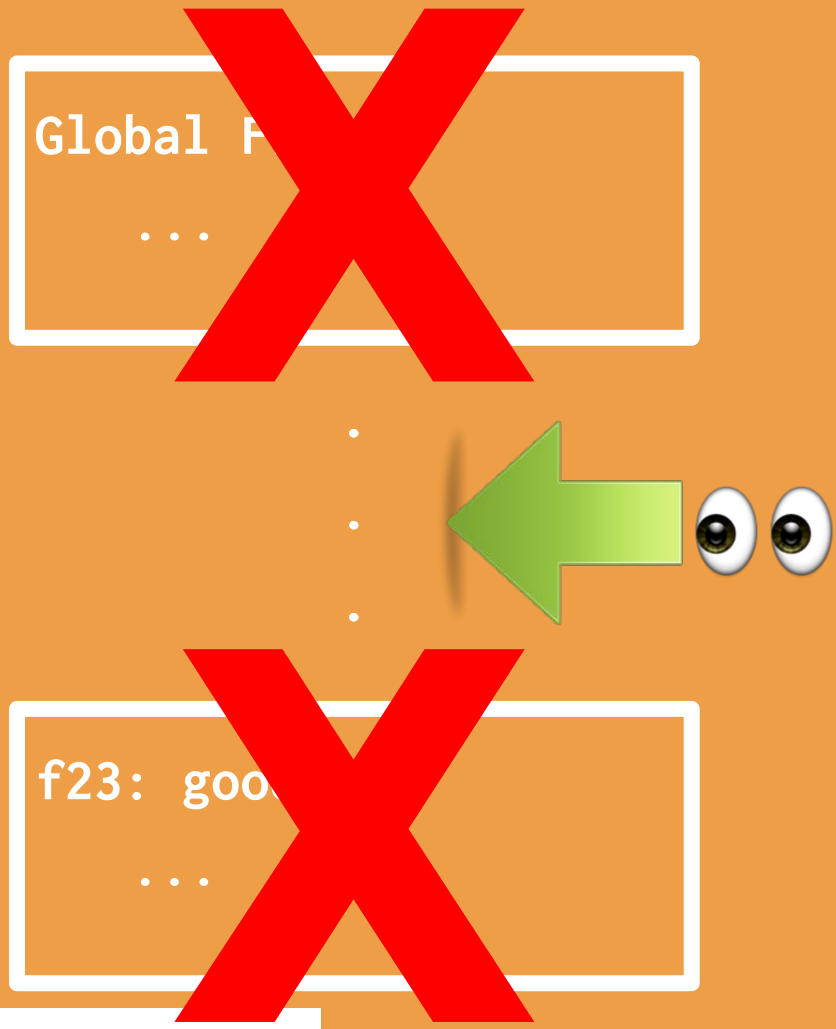
(4) None of the above criticisms are valid. Everything will be fine.

1.
Nonlocal

Change binding in first frame
where name is already bound

Nonlocal Facts

- Reassign nonlocal variables in the parent frame
- If a variable is declared as nonlocal, never look in the *global* or *current* frame!

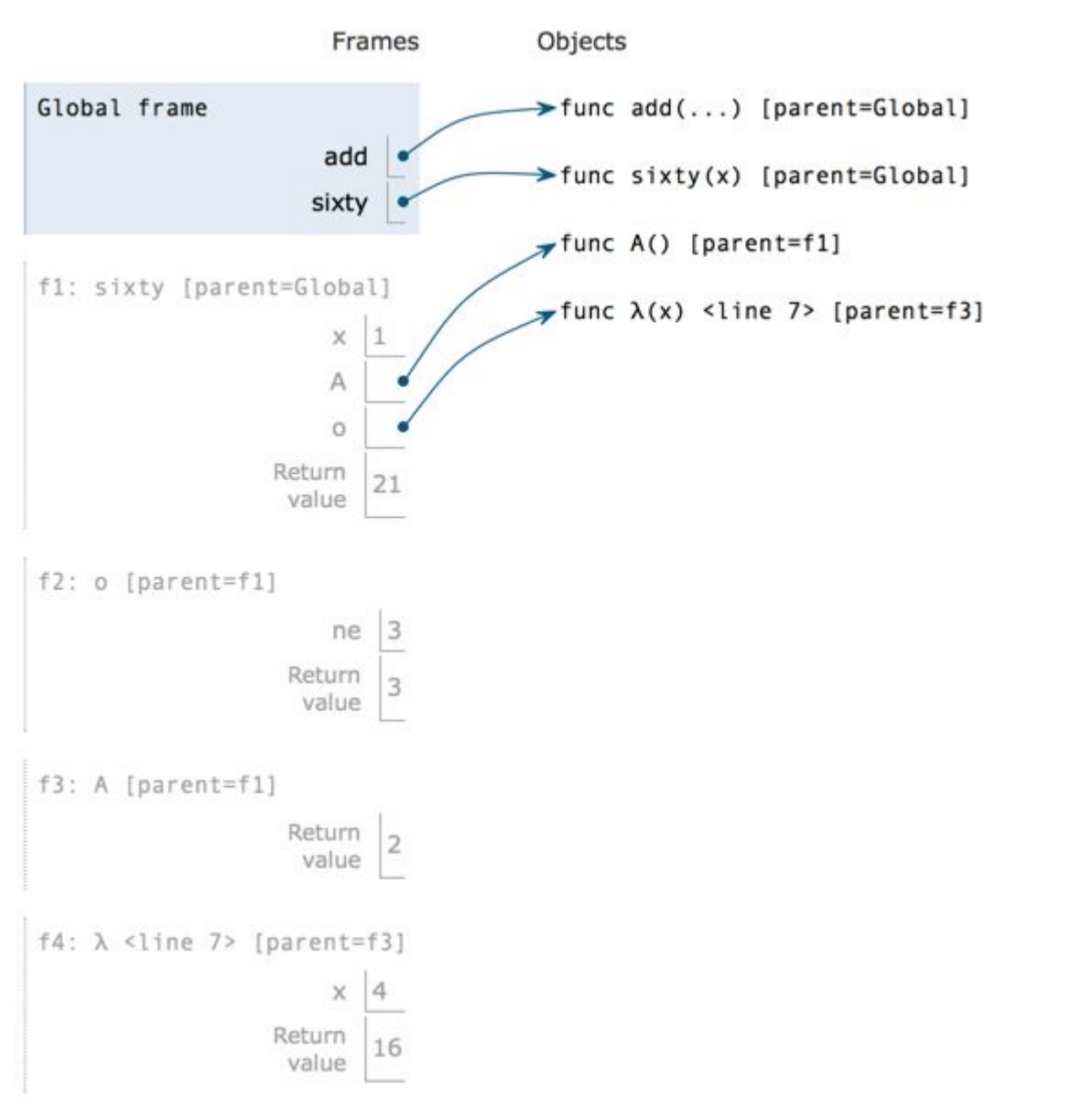



```
def good(luck):
    nonlocal on #the
    return final
```

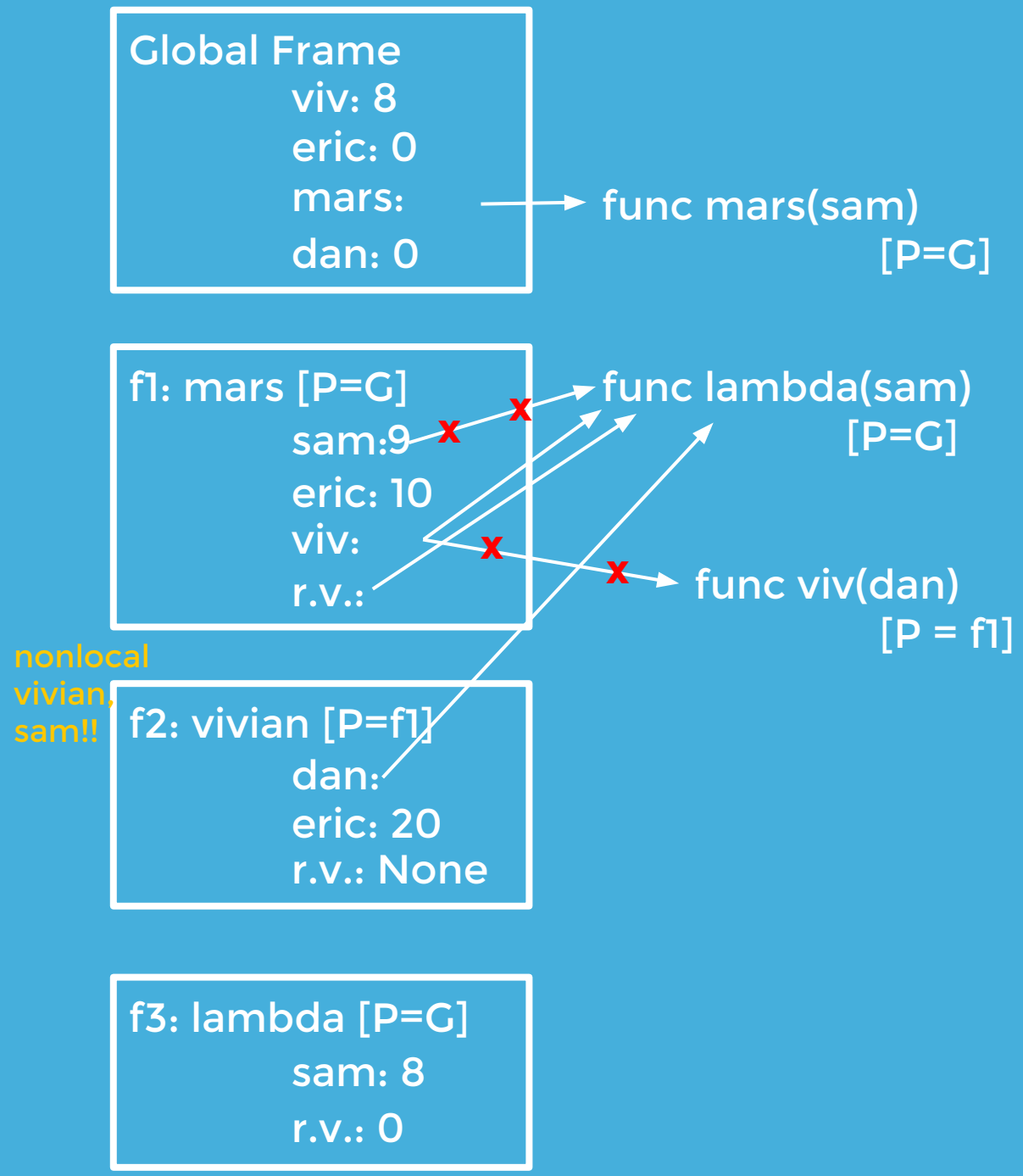
Draw an environment diagram for the following code.

```
from operator import add
def sixty(x):
    def o(ne):
        return x * ne
    def A():
        nonlocal o
        o = lambda x: x * x
        return 2
    return add(o(3), add(A(), o(4)))
sixty(1)
```

Solution:


Frames
Objects
Global frame
add
sixty
func add(...) [parent=Global]
func sixty(x) [parent=Global]
f1: sixty [parent=Global]
x
1
A
o
Return value
21
func A() [parent=f1]
func λ(x) <line 7> [parent=f3]
f2: o [parent=f1]
ne
3
Return value
3
f3: A [parent=f1]
Return value
2
f4: λ <line 7> [parent=f3]
x
4
Return value
16

```
viv = 8
eric = 0
def mars(sam):
    eric = 10
    def viv(dan):
        nonlocal viv
        nonlocal sam
        sam = 9
        eric = 20
        viv = dan
    viv(sam)
    return viv
dan = mars(lambda sam:
           eric*sam)(viv)
```



2.
Mutation

Modify what is already there

ABC\FGH

Draw environment diagrams for the following piece of code.

NOTE: We made a mistake during the review session. Contrary to our claim, where c is a list, c = c + [1] is NOT the same as c += [1]. c += [1] basically does what append does. c = c + [1] makes a new list and makes c point to it.

For h(c), we meant to write c = c + [1], as shown to the right. During the review session, we wrote c += [1]. Please forgive us for this confusion.

```
a, b, c = 0, [], []
def f(a):
    a += 1
def g(b):
    b.append(1)
def h(c):
    c = c + [1]
f(a)
g(b)
h(c)
```

ABC\FGH

```
a, b, c = 0, [], []
def f(a):
    a += 1
def g(b):
    b.append(1)
def h(c):
    c += [1]
f(a)
g(b)
h(c)
```

Map & Mutate

Implement a function map_mut that takes a list L as an argument and maps a function f onto each element of the list. You should mutate the original list. Do NOT return anything.

(Credit: Albert Wu)

```
def map_mut(f, L):
    """
    >>> L = [1, 2, 3, 4]
    >>> map_mut(lambda x: x**2, L)
    >>> L
    [1, 4, 9, 16]
    """
```

Map & Mutate

Implement a function map_mut that takes a list L as an argument and maps a function f onto each element of the list. You should mutate the original list. Do NOT return anything.

(Credit: Albert Wu)

```
def map_mut(f, L):
    """
    >>> L = [1, 2, 3, 4]
    >>> map_mut(lambda x: x**2, L)
    >>> L
    [1, 4, 9, 16]
    """
    for i in range(len(L)):
        L[i] = f(L[i])
```

3.
Interfaces

A common tongue across classes

Magic Methods

Magic methods are special methods that are called in special ways.

ex)

`lst[0]` calls

`lst.__getitem__(0)`.

```
__str__
__repr__
__getitem__
__len__
__init__
__iter__
__next__
```

The Iterator/Iterable Interface

- Iterable
  - Like a book
  - Just sits there while the iterator runs all over it
  - Must implement `__iter__`
  - `__iter__` gives bookmark of this book!
- Iterator
  - Like a bookmark
  - Must implement `__iter__` and `__next__`
  - `__next__` is like flipping to the next page
  - If no more pages, raise an exception

Write an iterator that takes two strings as input and outputs the letters interleaved when iterated over. Assume the strings are of equal length.

```
class StringWeaver:
    """
    >>> s = StringWeaver("ah", "HA")
    >>> for char in s:
    >>>     print(char)
    a
    H
    h
    A
    """
    def __init__(self, str1, str2):
        ***YOUR CODE HERE***
    def __iter__(self):
        ***YOUR CODE HERE***
    def __next__(self):
        ***YOUR CODE HERE***
```

Write an iterator that takes two strings as input and outputs the letters interleaved when iterated over. Assume the strings are of equal length.

```
class StringWeaver:
    def __init__(self, str1, str2):
        self.str1 = str1
        self.str2 = str2
        self.i = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.i == len(self.str1) + len(self.str2):
            raise StopIteration
        letter_to_output = ''
        if self.i % 2 == 0:
            letter_to_output = self.str1[self.i//2]
        else:
            letter_to_output = self.str2[self.i//2]
        self.i += 1
        return letter_to_output
```

4.
Recursive Objects

Heard you like objects...

Talk Binary to Me

```
class BinaryTree:
    empty = ()
    def __init__(self, entry, left=empty, right=empty):
        assert left is BinaryTree.empty or
                                    isinstance(left,BinaryTree)
        assert right is BinaryTree.empty or
                                    isinstance(right, BinaryTree)
        self.entry = entry
        self.left, self.right = left, right
```

The Doubly Linked Binary Tree

Create a new class that is identical to BinaryTree, but where each node has a parent as well as children.

```
class DLBT(BinaryTree):
    A BinaryTree with a parent
    def __init__(self,entry, left=BinaryTree.empty,
                             right=BinaryTree.empty):
        BinaryTree.__init__(self, entry, left, right)
```

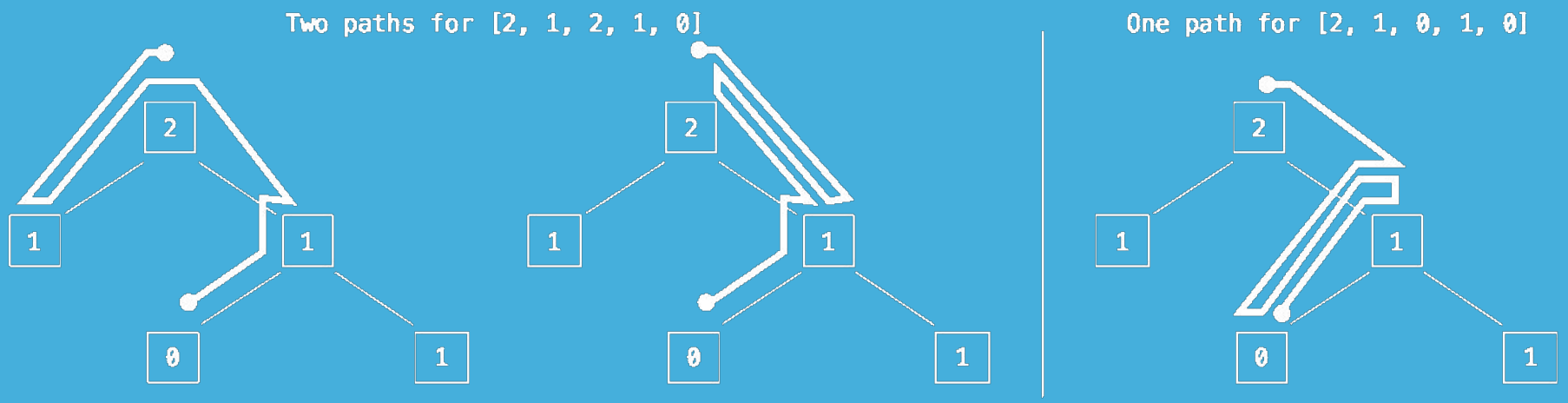
The Doubly Linked Binary Tree

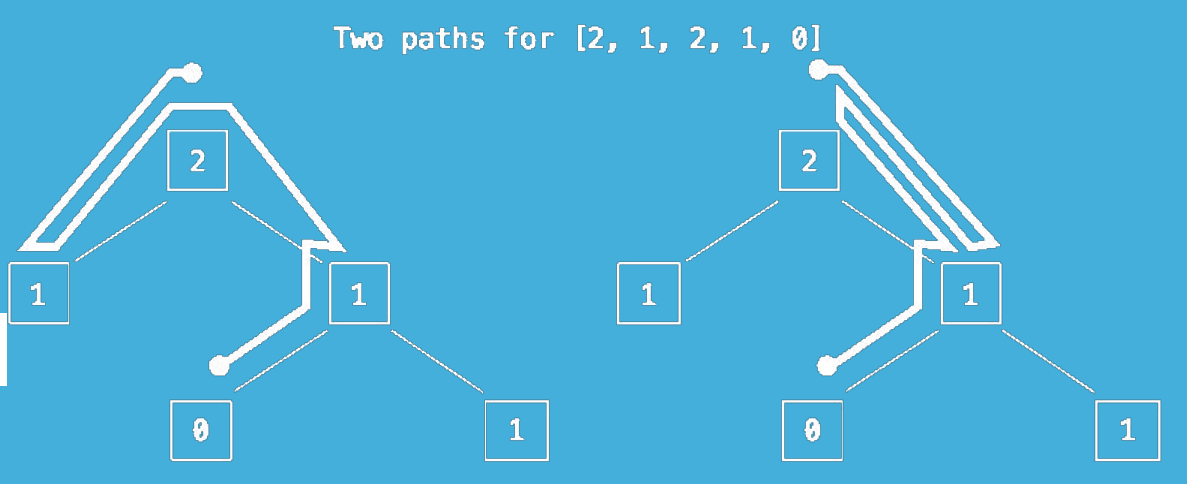
Create a new class that is identical to BinaryTree, but where each node has a parent as well as children.

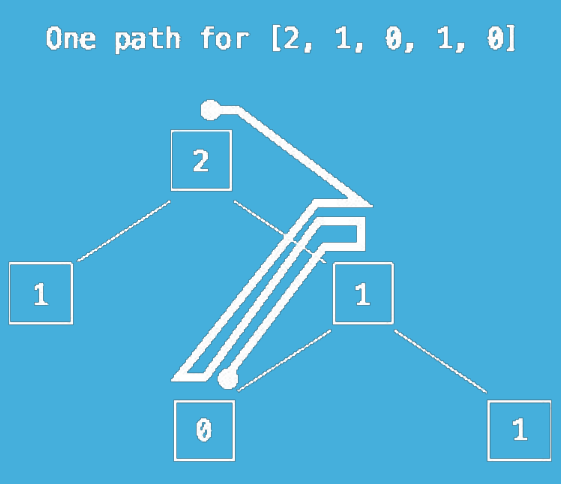
```
class DLBT(BinaryTree):
    A BinaryTree with a parent
    def __init__(self,entry, left=BinaryTree.empty,
                             right=BinaryTree.empty):
        BinaryTree.__init__(self, entry, left, right)
        self.parent = BinaryTree.empty
        for b in [left, right]:
            if b is not BinaryTree.empty
                b.parent = self
```

Walking on Some Tree

Write a function that takes in a DLBT `g` and a list `s`. It returns the number of paths through `g` whose entries are elements of `s`.





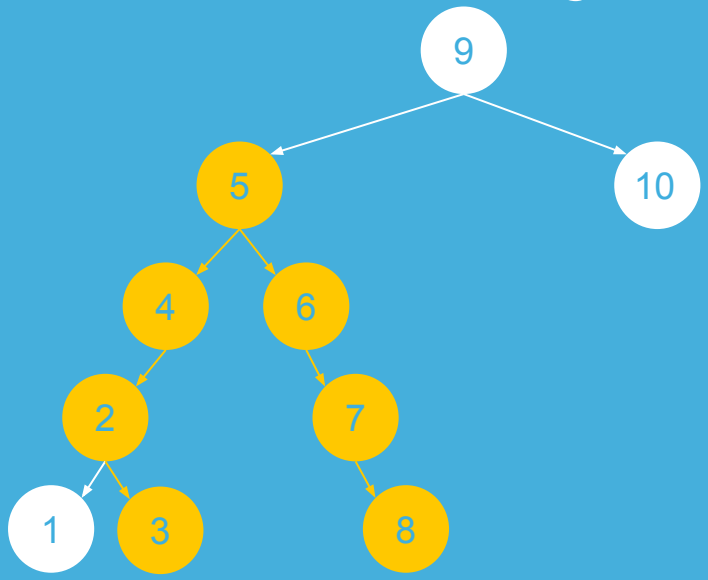


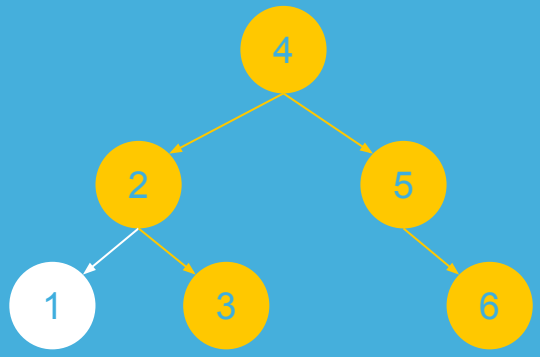
Write a function that takes in a DLBT `g` and a list `s`. It returns the number of paths through `g` whose entries are elements of `s`.

```
def paths(g, s):
    if g is BinaryTree.empty or s == [] or g.entry != s[0]:
        return 0
    elif len(s) == 1:
        return 1
    else:
        next_steps = [g.left, g.right, g.parent]
        return sum([paths(n, s[1:]) for n in next_steps])
```

Diameter Alley

Write a function that takes as input a BinaryTree, `g`, and returns its diameter. A diameter of a tree is the longest path between any two leaves. You can use `height` to determine the height of a tree.





Diameter Alley

Write a function that takes as input a BinaryTree, `g`, and returns its diameter. A diameter of a tree is the longest path between any two leaves. You can use `height` to determine the height of a tree.

```
def diameter(g):
```

Diameter Alley

Write a function that takes as input a BinaryTree, `g`, and returns its diameter. A diameter of a tree is the longest path between any two leaves. You can use `height` to determine the height of a tree.

```
def diameter(g):
    left_height = height(g.left)
    right_height = height(g.right)
    left_diameter = diameter(g.left)
    right_diameter = diameter(g.right)
    return max(left_height + right_height + 1,
               left_diamater, right_diameter)
```

The Link Before Time

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        if not (rest is Link.empty or isinstance(rest, Link)):
            raise ValueError('rest must be Link or empty')
        self.first = first
        self.rest = rest
    def __repr__(self):
        ...
    def __len__(self):
        ...
```

Linked List Revolution

Change the `Link` class so that each node now points to the element directly after it AND directly before it.

```
class DoubleLink(Link):
    def __init__(self, first, rest=Link.empty, prev=Link.empty):
        Link.__init__(self, entry, first, rest)
```

_____

_____

_____

_____

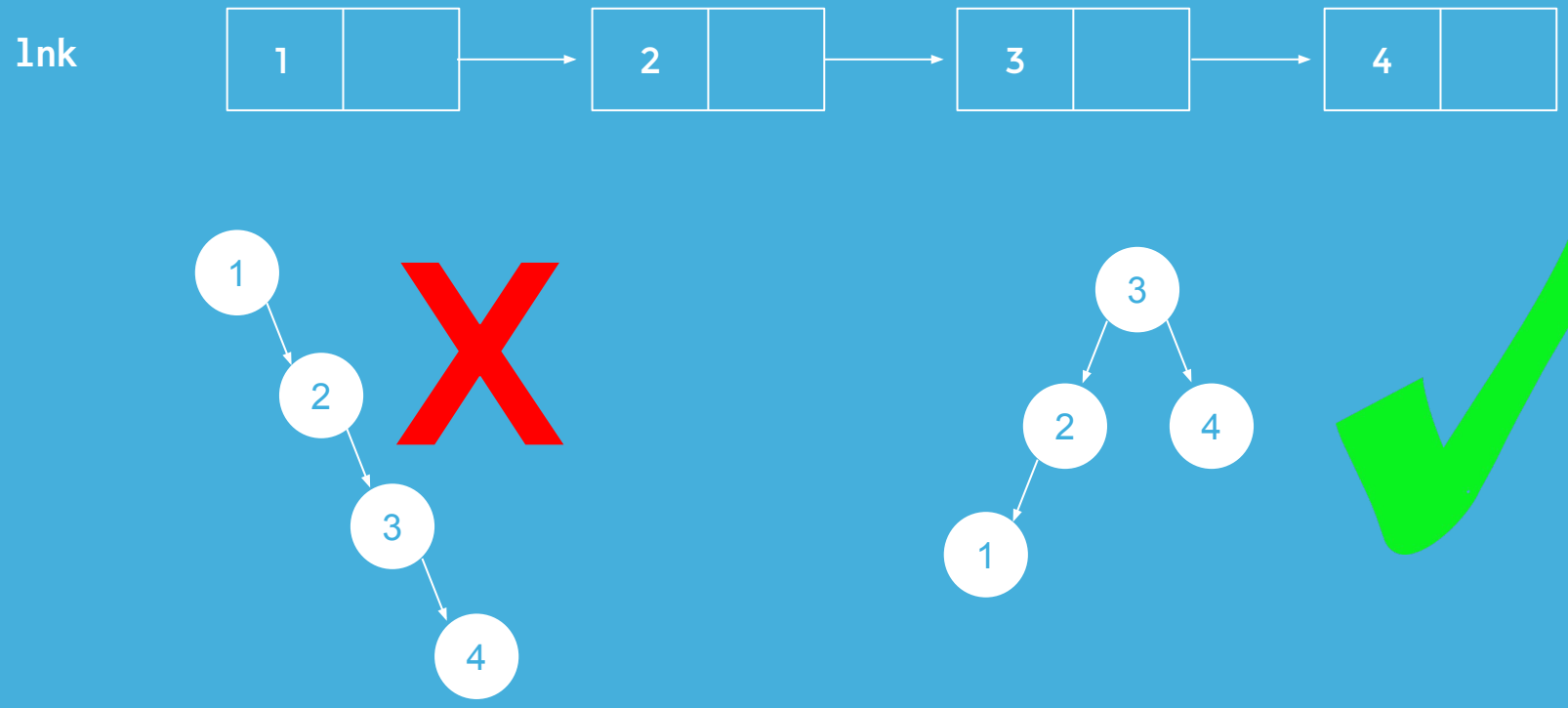
Linked List Revolution

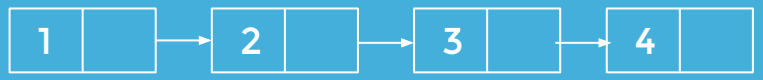
Change the `Link` class so that each node now points to the element directly after it AND directly before it.

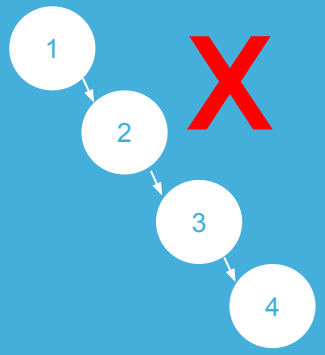
```
class DoubleLink(Link):
    def __init__(self, first, rest=Link.empty, prev=Link.empty):
        Link.__init__(self, entry, first, rest)
        self.prev = Link.empty
        if self.rest is not Link.empty:
            self.rest.prev = self
```

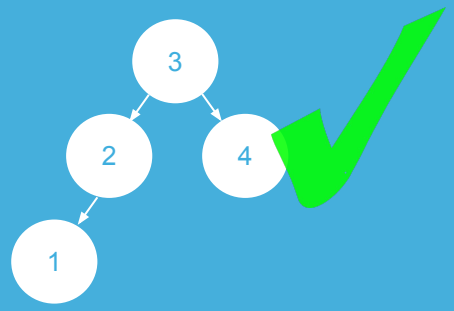
The Giving Link

Given a sorted DoubleLink lnk, construct the corresponding BST (NOT DLBT!) that is *balanced*





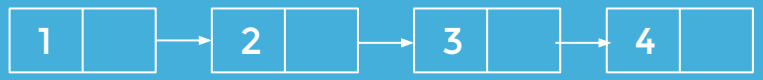


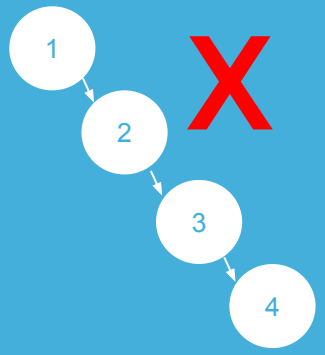


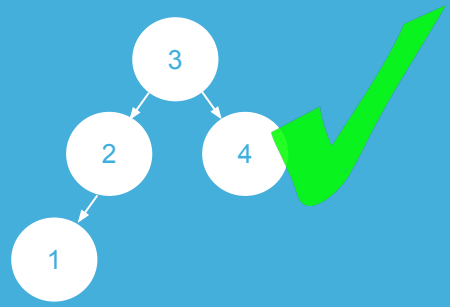
The Giving Link

Given a sorted DoubleLink lnk, construct the corresponding BST (NOT DLBT!) that is balanced

```
def convert(lnk):
    length = len(lnk)
    if length == 0:
        _____
    if length == 1:
        _____
    if length == 2:
        _____
    l, r = lnk, lnk
    for i in range(length/2):
        _____
        _____
    return BST(_____)
```







The Giving Link

Given a sorted DoubleLink lnk, construct the corresponding BST (NOT DLBT!) that is balanced

```
def convert(lnk):
    length = len(lnk)
    if length == 0:
        return BST.empty
    if length == 1:
        return BST(lnk.first)
    if length == 2:
        return BST(lnk.rest.first, BST(lnk.first))
    l, r = lnk, lnk
    for i in range(length/2):
        r = r.rest
    r, r.prev.rest, r.rest.prev = r.rest, BST.empty, BST.empty
    return BST(lnk.first, convert(l), convert(r))
```

5.
Scheme

"The only computer language that is beautiful"

- Neal Stephenson

Scheme Synopsis

- No iteration, just recursion
- When to define a helper function?
  - When the number of variables you need to keep track of is bigger than the number of arguments to the function
- Call a function by surrounding it with parenthesis

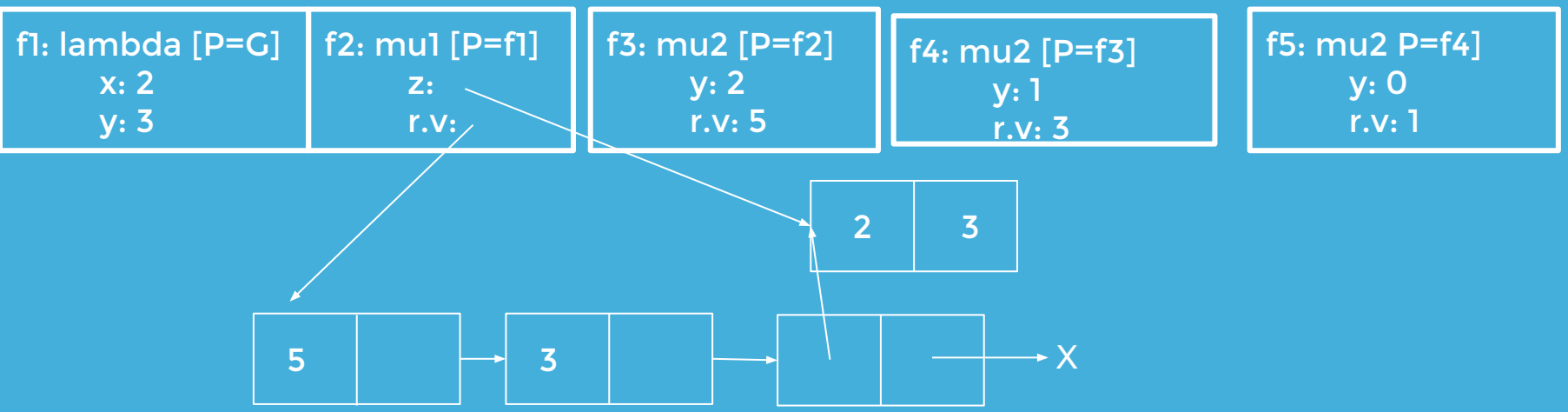
WWSD?

```
( define f ( lambda ( x y ) ( g ( cons x y )) ))
( define g ( mu ( z ) ( list ( h x ) y z )))
( define h ( mu ( y ) ( if ( > y 0) (+ x ( h ( - y 1))) 1)))


(f 2 3)
```

WWSD?

```
(define x 0)
(define y 1)
(define f (lambda (x y) (g (cons x y))))
(define g (mu (z) (list (h x) y z)))
(define h (mu (y) (if (> y 0) (+ x (h (- y 1))) 1)))
(f 2 3)
```



Enter Interpretation

In your project 4 implementation, how many total calls to scheme eval and scheme apply would result from evaluating the following two expressions? Assume that you are not using the tail call optimized scheme eval optimized function for evaluation.

```
( define ( square x ) (* x x ))
```

```
(+ ( square 3) ( - 3 2))
```

14 eval calls!

4 apply calls!

I Scheme, You Scheme, We all Scheme for Scheme Streams

Let's try to compress repetitive data! For example, in the (finite) sequence

1, 1, 1, 1, 1, 6, 6, 6, 6, 2, 5, 5, 5

there are four runs: one each of 1, 6, 2, and 5. We can represent the same sequence as a sequence of two-element lists:

(1 5), (6 4), (2 1), (5 3)

We will extend this idea to (possibly infinite) streams. Write a function called `rle` that takes in a stream of data, and returns a corresponding stream of two-element lists, which represents the run-length encoded version of the stream. You do not have to consider compressing infinite runs.

I Scheme, You Scheme, We all Scheme for Scheme Streams

```
(define (rle s)
  (define (track-run elem st len)
    (cond ((null? st) (cons-stream (list elem len) nil))
          ((= elem (car st)) (track-run elem (cdr-stream st) (+ len 1)))
          (else (cons-stream (list elem len) (rle st))))
  )
  (if (null? s)
      nil
      (track-run (car s) (cdr-stream s) 1))
)
```

6.
Tail Recursion

Recursive calls in a tail context

Chase Your Tail

Determine which of the following definitions are tail recursive.

```
(define foo
  (lambda (x)
    (if (even? x)
      1
      (foo (- x 1)))))
```

✓

```
(define (even1? x)
     (if (= 0 x)
     #t
     (not (even1? (- x 1)
))))
```

✗

```
(define (even2? x)
     (cond
     (= 0 x) #t
     (= 1 x) #f
     (else (even2? ( - x 2)))))
```

✓

```
(define (even2? x)
     (if (= 0 x)
     #t
     (odd2? (- x 1)))))
```

```
(define (odd2? x)
     (if (= 0 x)
     #f
     (even2? (- x 1)))))
```

✓

```
(define (even3? x)
     (cond
     (= 0 x) #t
     (= 1 x) #f
     (else (begin
          (define x (- x 1))
          (even3? x)))
```

✓

```
(define (even4? x)
     (or
     (= 0 x)
     (not (even4? (- x 1)))
     (even4? (- x 2)))
```

✗

```
(define (even5? x)
     (or
     (= 0 x)
     (= 1 x)
     (even5? (- x 2))))
```

✓

Tail Reverse

Write a function that takes in a list, `lst`, and returns a new list that contains all the elements of `l` in reverse order.

```
(define (reverse lst)
  (define (reverse-tail sofar rest)
    (if _________

        _____
        (reverse-tail ________________________  ________ )))
  (reverse-tail ___  __))
```

Tail Reverse

Write a function that takes in a list, `l`, and returns a new list that contains all the elements of `l` in reverse order.

```
(define (reverse lst)
  (define (reverse-tail sofar rest)
    (if (null? rest)
        sofar
        (reverse-tail (cons (car rest) sofar) (cdr rest))))
  (reverse-tail nil lst))
```

Tail Insert

Write a function that takes in a list, `l`, element, `elem`, and index, `i`, and returns a new list that is the same as `l` but with `elem` inserted at index `i`.

```
(define (insert l elem i)
    (define (helper l i so-far)
          (if (or _______ ______)
          (append ______ ___________)
          (helper ______ _____ ____________________________)))
    (helper ___ __))
```

Tail Insert

Write a function that takes in a list, `l`, element, `elem`, and index, `i`, and returns a new list that is the same as `l` but with `elem` inserted at index `i`.

```
(define (insert l elem i)
    (define (helper l i so-far)
            (if (or (null? l) (= i 0))
            (append so-far (cons elem l))
            (helper (cdr l) (- i 1) (append so-far (list (car l))))))
    (helper l i nil))
```

Axioms
and worlds that satisfy those axioms

Different Paradigms

Imperative programming

- Python & Scheme
- Programmer writes very specific instructions

Declarative programming

- Logic
- Programmer writes what the solution should look like, computer does rest of the work to get to the solution

Basic Syntax of Logic

```
logic> (fact (eats cat fish))
logic> (query (eats cat ?what))
Success!
what: fish
```

Compound Facts

Conclusion is true if ALL of the hypotheses are true

```
(fact
(<conclusion>)
(<hypothesis_1>)
…
(<hypothesis_n>))
```

Recursive Facts

A compound fact that uses the same relation in its conclusion and its hypotheses

```
(fact (parent dan neil))
(fact (parent marv dan))

(fact (ancestor ?p1 ?p2)
      (parent ?p1 ?p2))
(fact (ancestor ?p1 ?p2)
      (parent ?p1 ?p3)
      (ancestor ?p3 ?p2))
```

Define a set of facts for `dank`, which takes in a list. A list is `dank` if it has the symbol `memes` inside of it.

Define a set of facts for `danker`, which takes in a list. A list is `danker` if two consecutive entries are each the symbol `memes`.

Define a set of facts for `dankest`, which takes in a list. A list is `dankest` if every one of its entries is the symbol `memes`.

Define a set of facts for `dank`, which takes in a list.
A list is `dank` if it has the symbol `memes` inside of it.

```
(fact (dank (memes . ?cdr)))
(fact (dank (?car . ?cdr))
      (dank ?cdr))
```

Define a set of facts for `danker`, which takes in a list.
A list is `danker` if two consecutive entries are each the symbol `memes`.

```
(fact (danker (memes memes . ?cddr)))
(fact (danker (?car . ?cdr))
      (danker ?cdr))
```

Define a set of facts for `dankest`, which takes in a list.
A list is `dankest` if every one of its entries is the symbol `memes`.

```
(fact (dankest ()))
(fact (dankest (memes . ?cdr))
      (dankest ?cdr))
```

THANKS!

Good luck on the final!