61A Lecture 22

Monday, March 16

Announcements

- Midterm 2 is on Thursday 3/19 7pm-9pm
  - Topics and locations: http://cs61a.org/exams/midterm2.html
  - Bring 1 hand-written, 2-sided sheet of notes; Two study guides will be provided
  - Emphasis: mutable data, object-oriented programming, recursion, and recursive data
  - Review session on Tuesday 5:00pm-6:30pm in 2050 VLSB
  - Includes content through Friday 3/13 (today is review & examples)
- No lecture next Wednesday 3/18
- No discussion sections next Thursday 3/19 or Friday 3/20
- Lecture next Friday 3/20 is a video (but a great one)

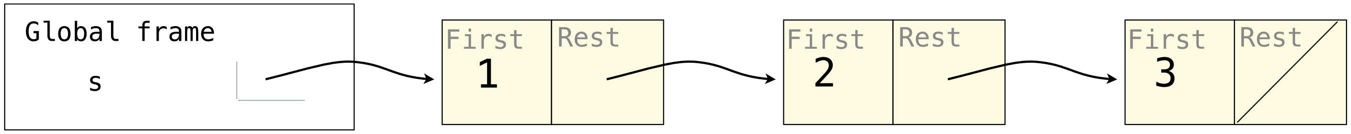
Linked Lists

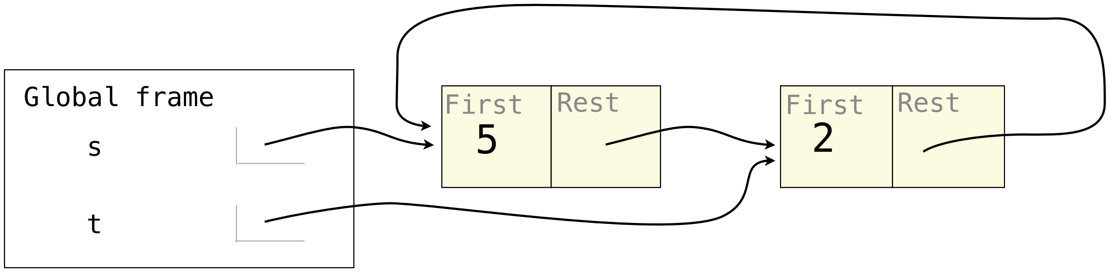
Recursive Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
2
```



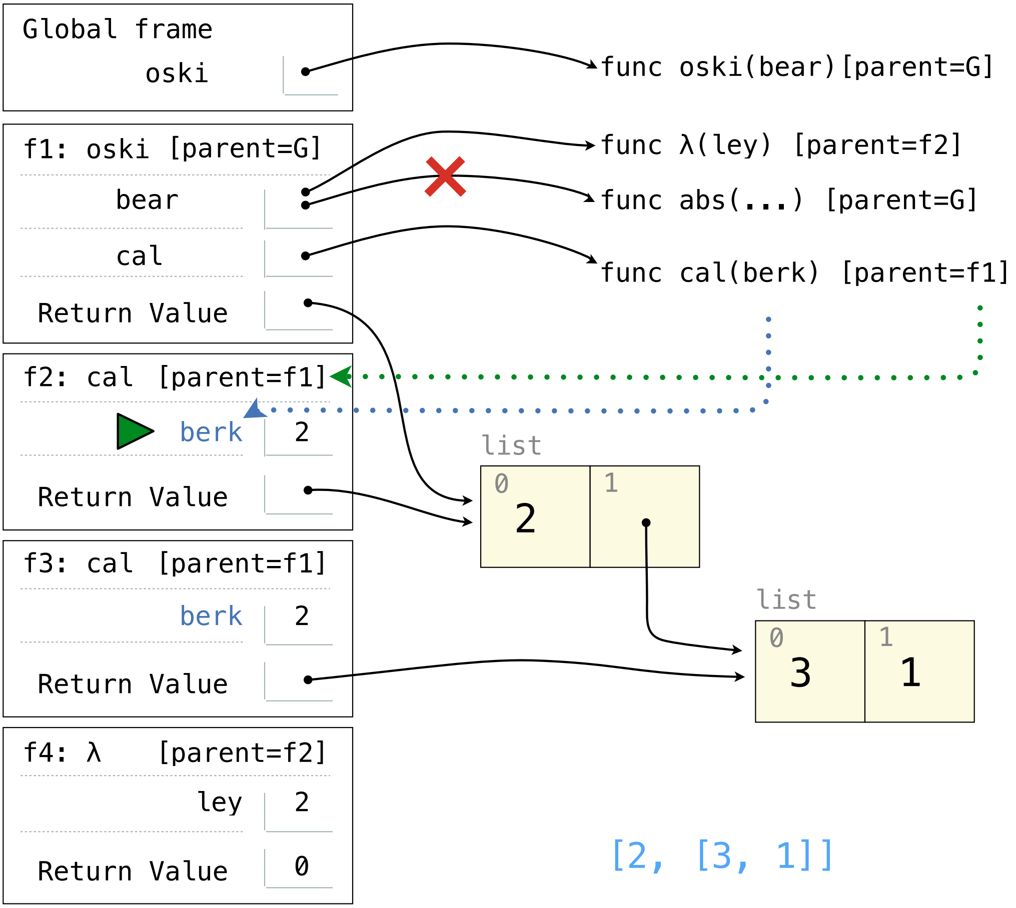


Note: The actual environment diagram is much more complicated.

Environment Diagrams

Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



Global frame
oski
func oski(bear)[parent=G]

f1: oski [parent=G]
bear
cal
Return Value
func λ(ley) [parent=f2]
func abs(...) [parent=G]
func cal(berk) [parent=f1]

f2: cal [parent=f1]
berk 2
Return Value

f3: cal [parent=f1]
berk 2
Return Value

f4: λ [parent=f2]
ley 2
Return Value 0

list
0 2
1

list
0 3
1 1

[2, [3, 1]]

Objects

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

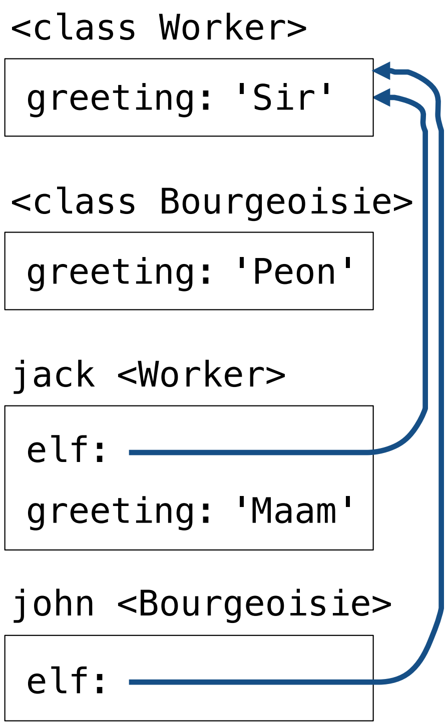
```
>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
Peon, I work
'I gather wealth'

>>> john.elf.work(john)
'Peon, I work'
```



Binary Trees

Morse Code

Morse code is a signaling protocol that transmits messages by sequences of signals

Problem: Implement **morse** so that **decode** works correctly

A: ● ▬
B: ▬ ● ● ●
C: ▬ ● ▬ ●
D: ▬ ● ●
E: ●
...

```
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter using a morse code tree.

    >>> t = morse(abcde)
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', '.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """
    for signal in signals:
        if signal == '.':
            tree = tree.left
        elif signal == '-':
            tree = tree.right
    return tree.entry
```

```
def morse(code):
    ....
```

(Demo)