# HKN CS 61A Final Review

## Spring 2015

Austin Le
Sherdil Niyaz
Allen Li
Mike Ambrose

# Hello!

Hosted by HKN (hkn.eecs.berkeley.edu)

Office hours from 11AM to 5PM in 290 Cory, 345 Soda

Check our website for exam archive, course guide, course surveys, tutoring schedule (hkn.eecs.berkeley.edu/tutor)

**DISCLAIMER:** This is an unofficial review session and HKN is not affiliated with this course. All of the topics we are reviewing will reflect the material you have covered, our experiences in CS 61A, and past exams. We make no promise that what we cover will necessarily reflect the content of the final. Some members of the course staff may be presenting, but this review is *still not* official.

# Agenda

- Environment diagrams
- Linked lists
- Trees
- Orders of growth
- Object-oriented programming
- Streams
- Iterators/Generators
- Scheme
- SQL

Follow along! **http://tinyurl.com/ld2wej9**

Unfortunately, we cannot cover everything that is within scope for the final.

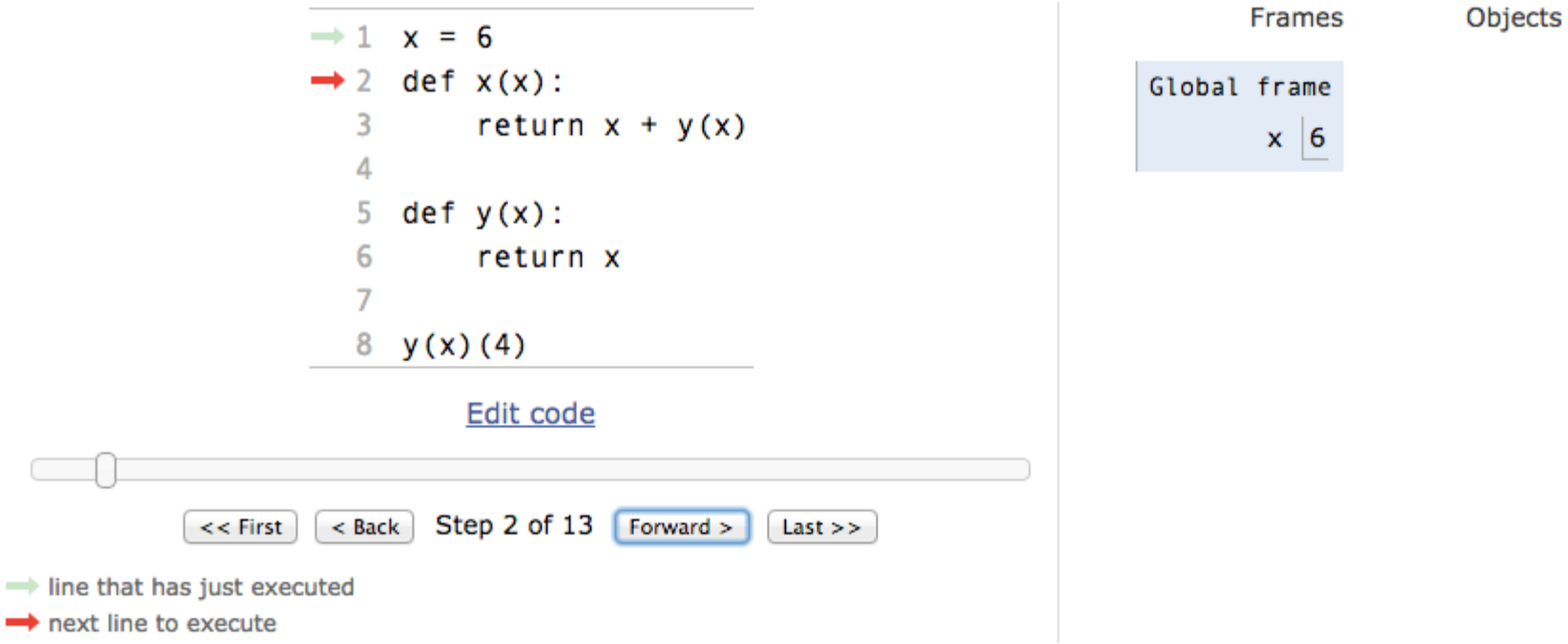
**This is not necessarily an exhaustive list of things to study! Check out the official details on Piazza and on cs61a.org.**

# Environment Diagrams

# Environment diagrams

- Evaluate the right side first
- New frame when you call a function
- When you're assigning a primitive expressions to a variable, write the value inside the box
- Anything else, draw an arrow
- Don't forget parent frames

# Environment Diagrams



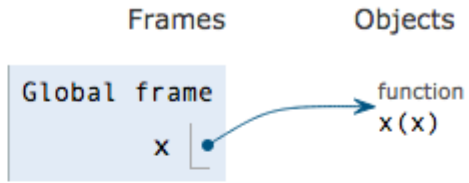
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 3 of 13 | Forward > | Last >>

line that has just executed
next line to execute



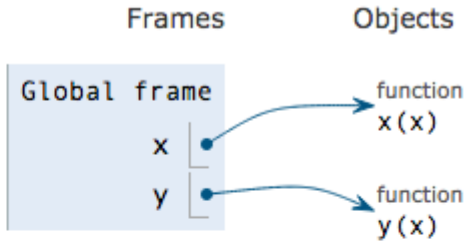
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 4 of 13 | Forward > | Last >>

line that has just executed
next line to execute



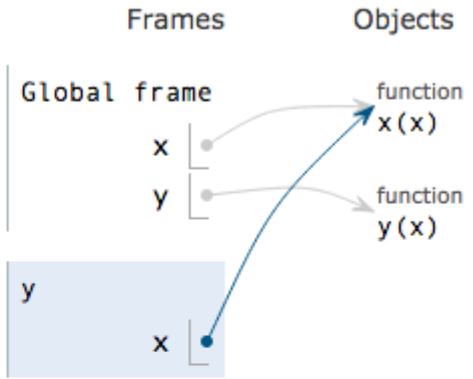
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 5 of 13 | Forward > | Last >>

line that has just executed
next line to execute



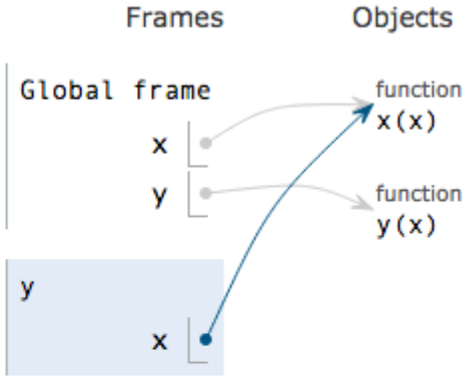
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 6 of 13 | Forward > | Last >>

line that has just executed
next line to execute



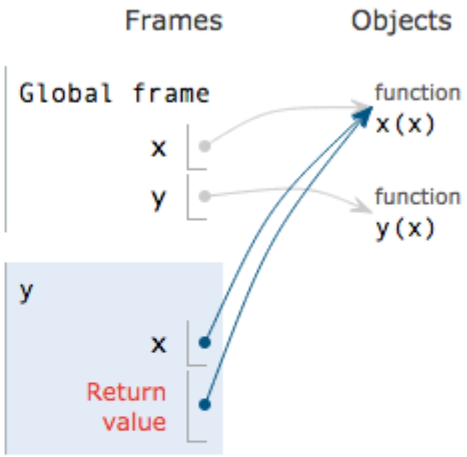
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 7 of 13 | Forward > | Last >>

line that has just executed
next line to execute



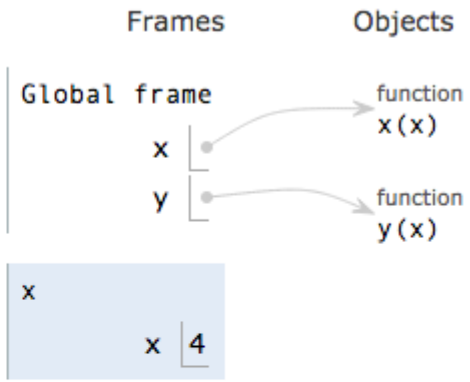
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 8 of 13 | Forward > | Last >>

line that has just executed
next line to execute



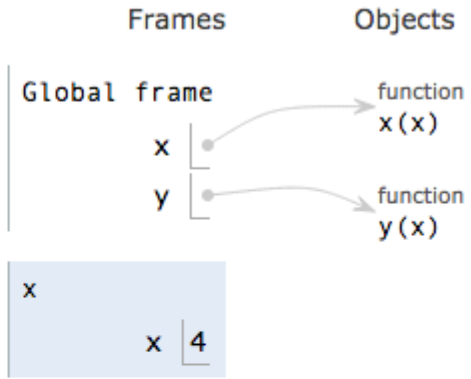
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 9 of 13 | Forward > | Last >>

line that has just executed
next line to execute



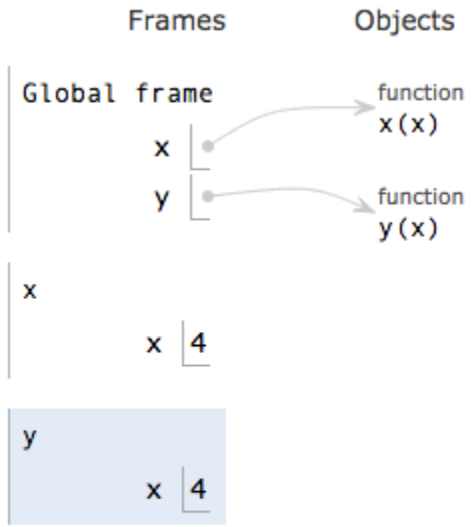
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 10 of 13 | Forward > | Last >>

line that has just executed
next line to execute



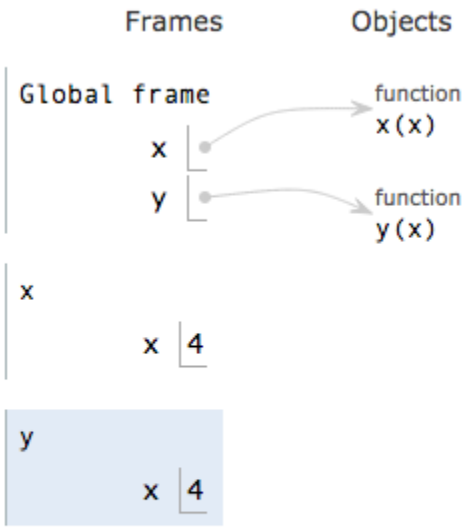
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 11 of 13 | Forward > | Last >>

line that has just executed
next line to execute



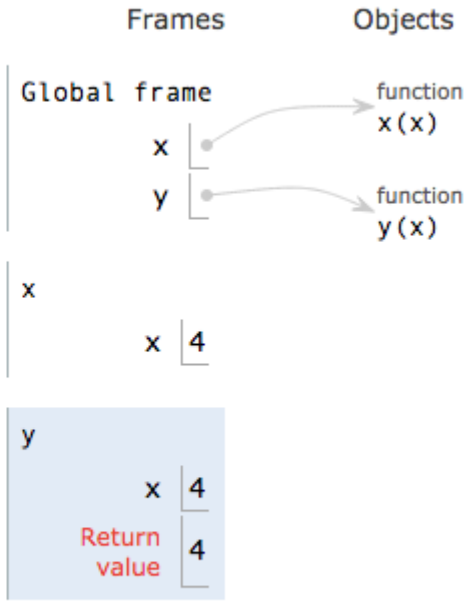
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 12 of 13 | Forward > | Last >>

line that has just executed
next line to execute



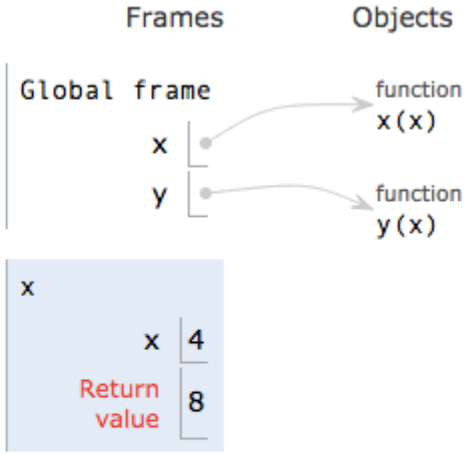
# Environment Diagrams

```
1  x = 6
2  def x(x):
3      return x + y(x)
4
5  def y(x):
6      return x
7
8  y(x)(4)
```

Edit code

<< First | < Back | Step 13 of 13 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

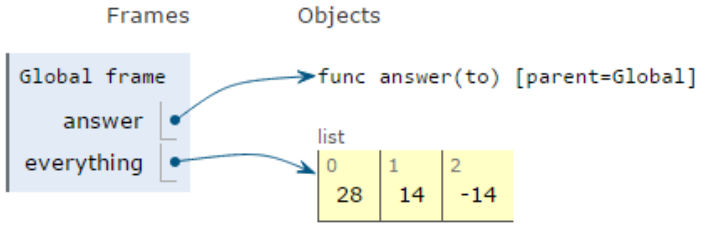
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 3 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

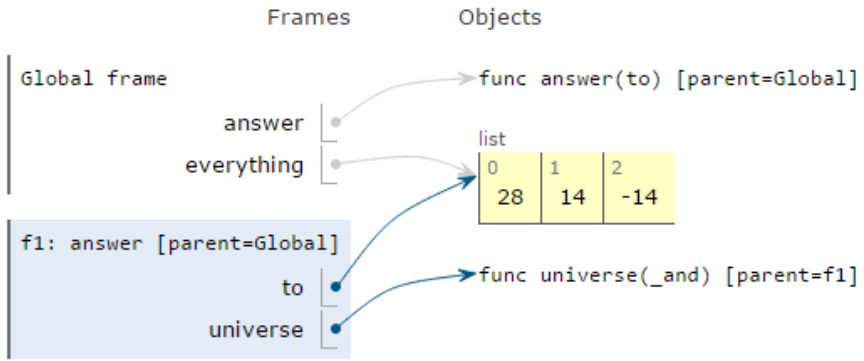
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 6 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

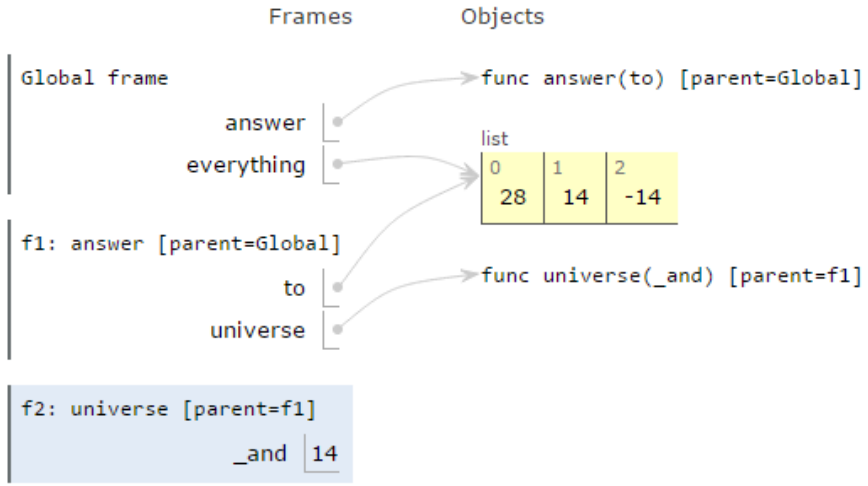
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 8 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

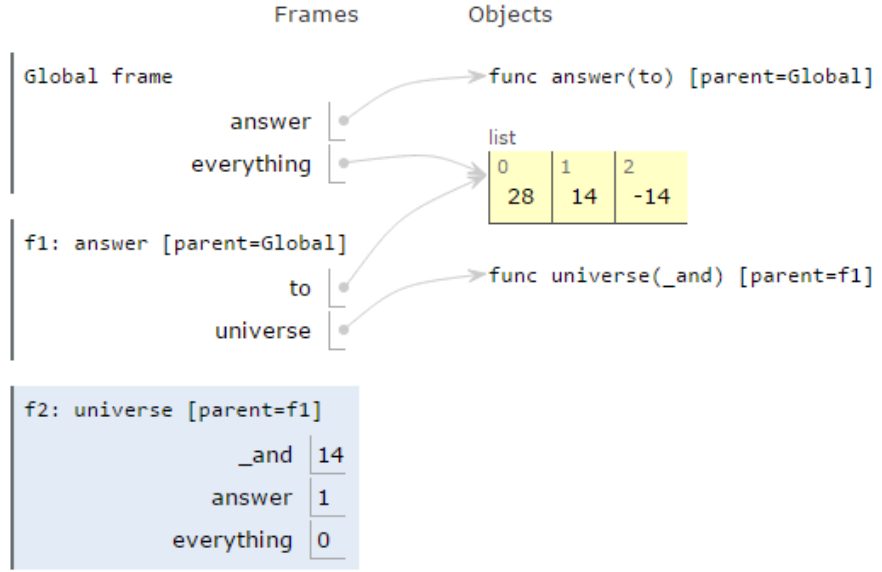
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 10 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

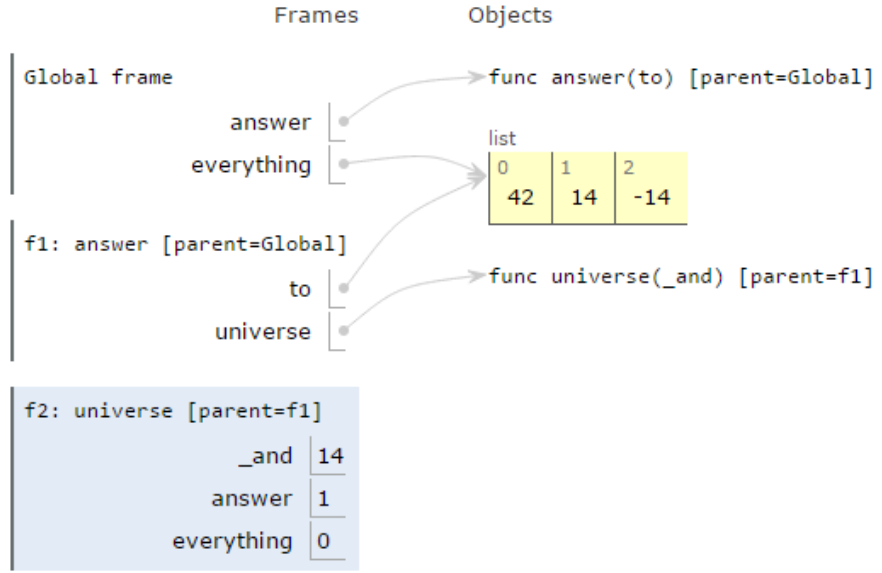
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 11 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

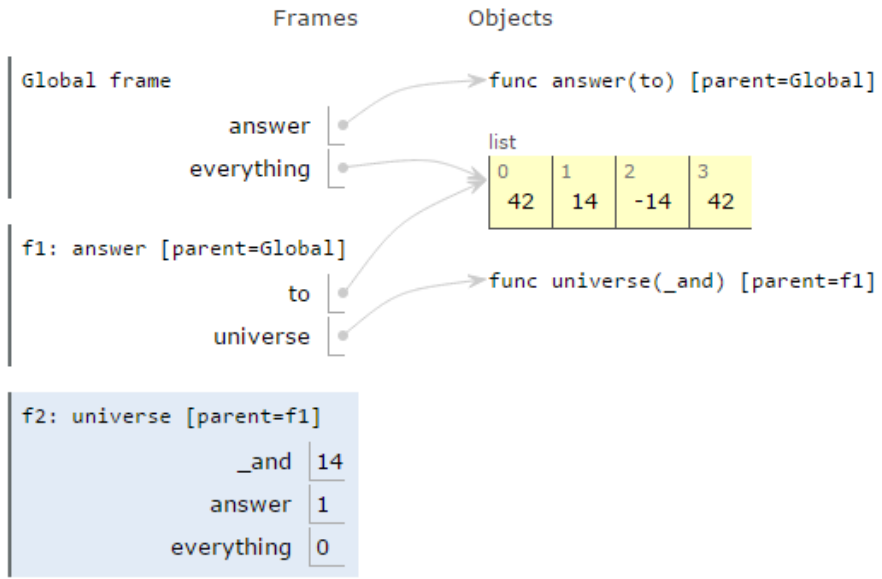
Python 3.3

```
 1  def answer(to):
 2      def universe(_and):
 3          nonlocal to
 4          answer = 1
 5          for everything in range(len(to)):
 6              to[everything] += _and
 7              to += [_and//answer * 3]
 8              _and *= 2
 9              answer *= 2
10      return universe(14)
11
12  everything = [28, 14, -14]
13  answer(everything)
```

Edit code

<< First | < Back | Step 12 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

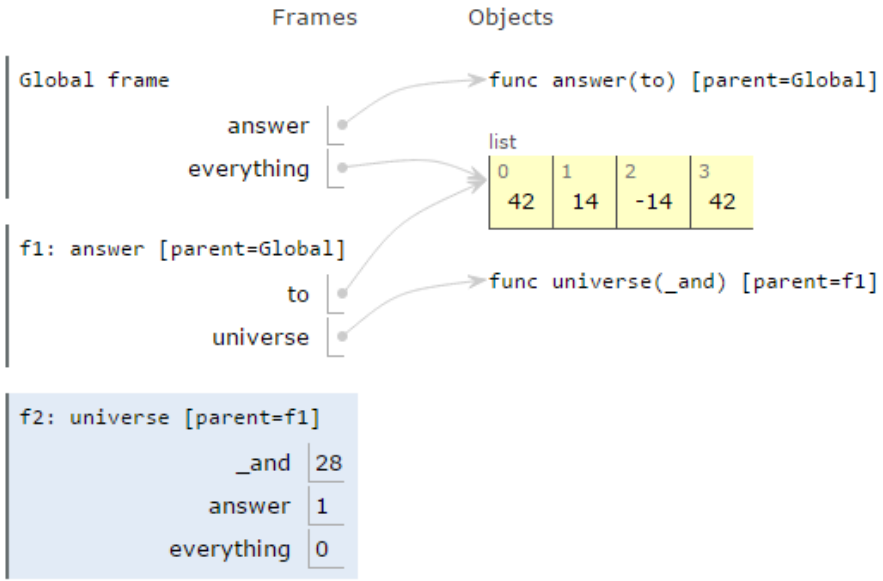
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 13 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

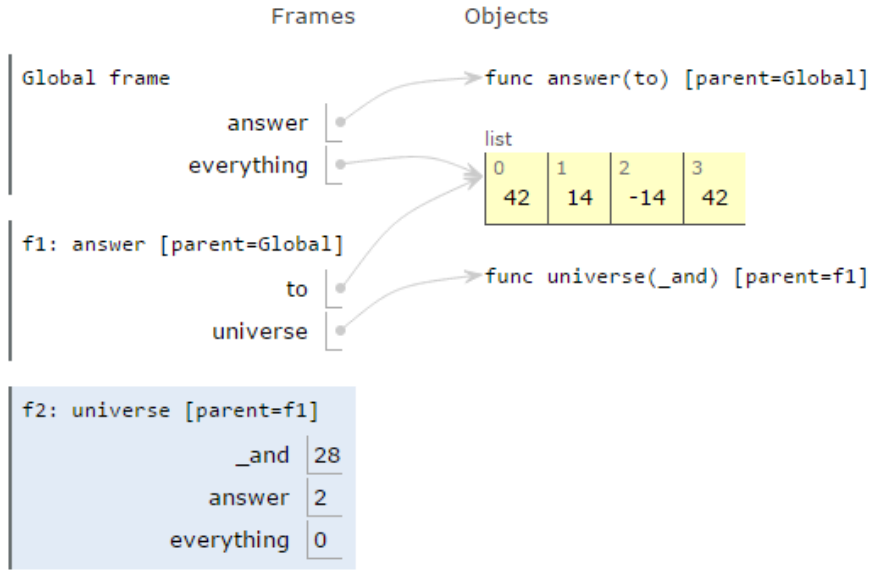
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 14 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

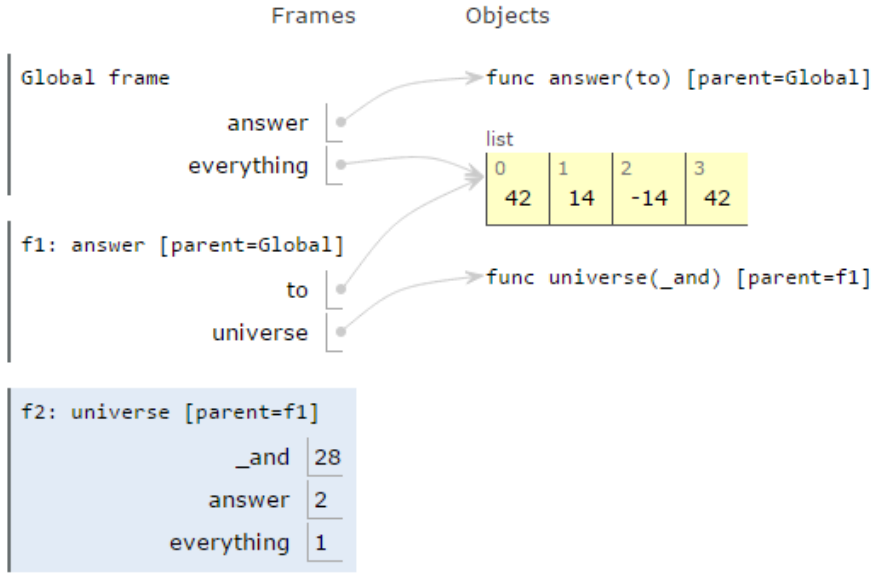
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 15 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

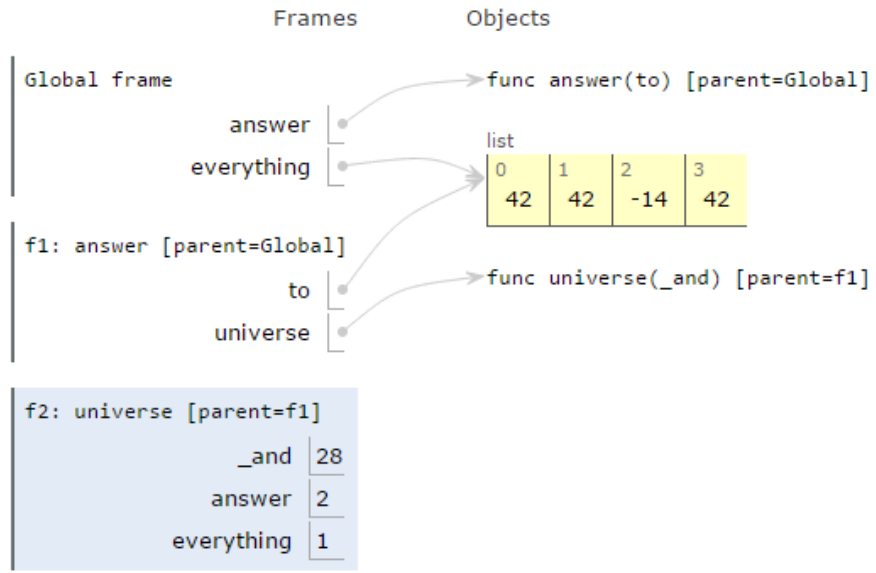
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 16 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

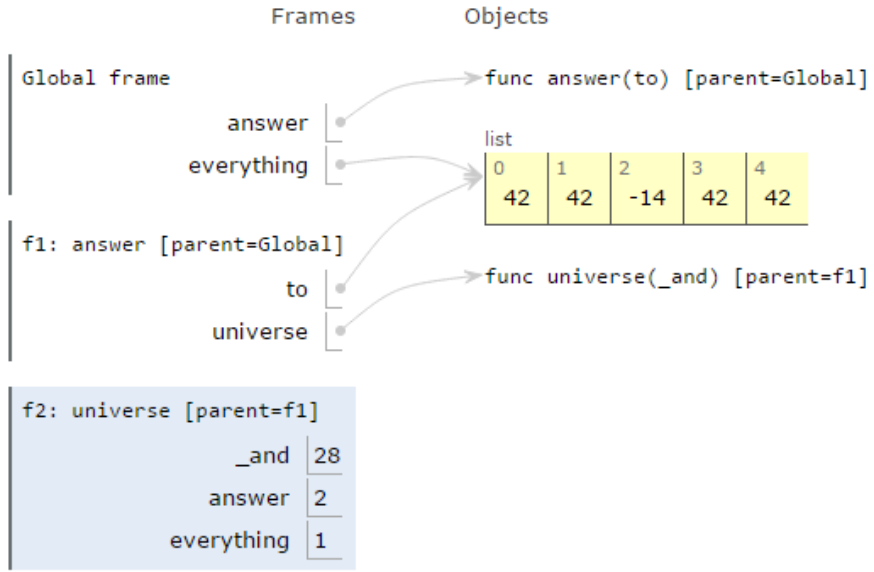
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 17 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

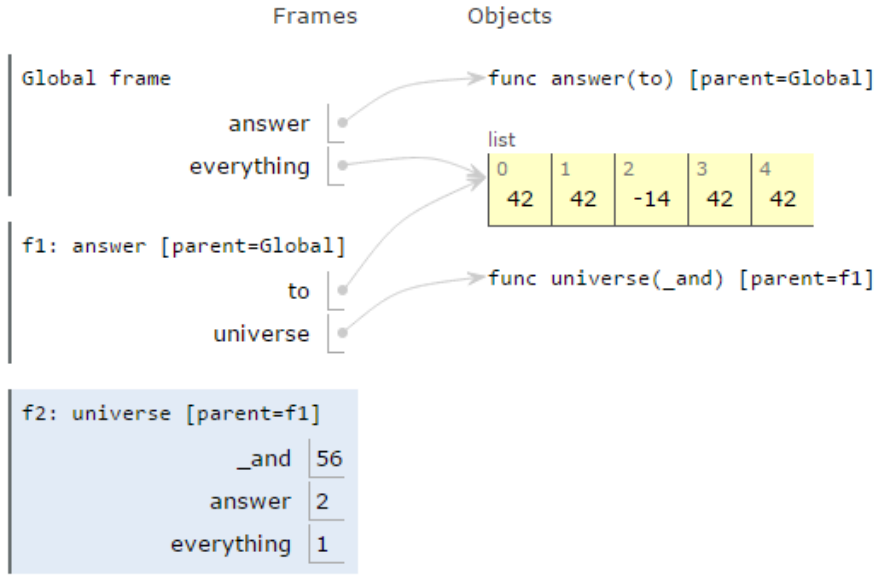
Python 3.3

```
 1  def answer(to):
 2      def universe(_and):
 3          nonlocal to
 4          answer = 1
 5          for everything in range(len(to)):
 6              to[everything] += _and
 7              to += [_and//answer * 3]
 8              _and *= 2
 9              answer *= 2
10      return universe(14)
11
12  everything = [28, 14, -14]
13  answer(everything)
```

Edit code

<< First | < Back | Step 18 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

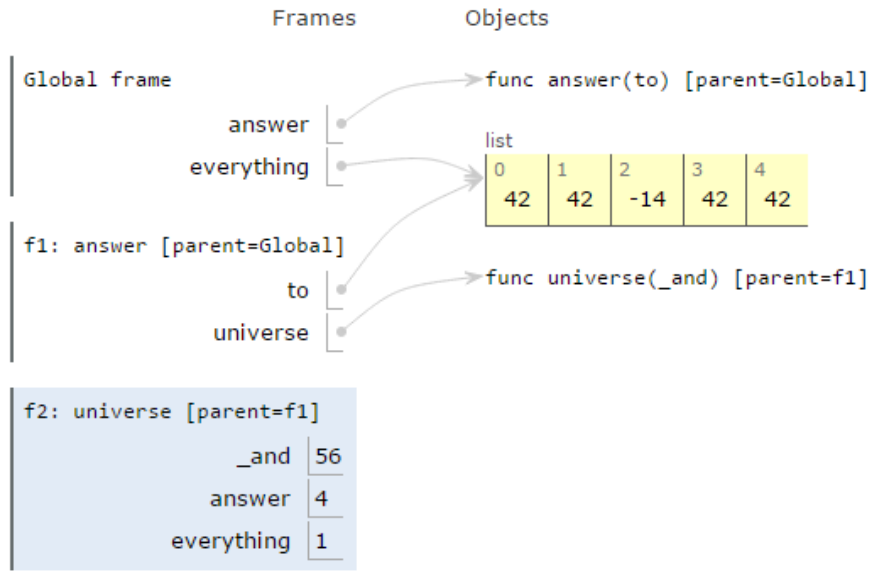
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 19 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

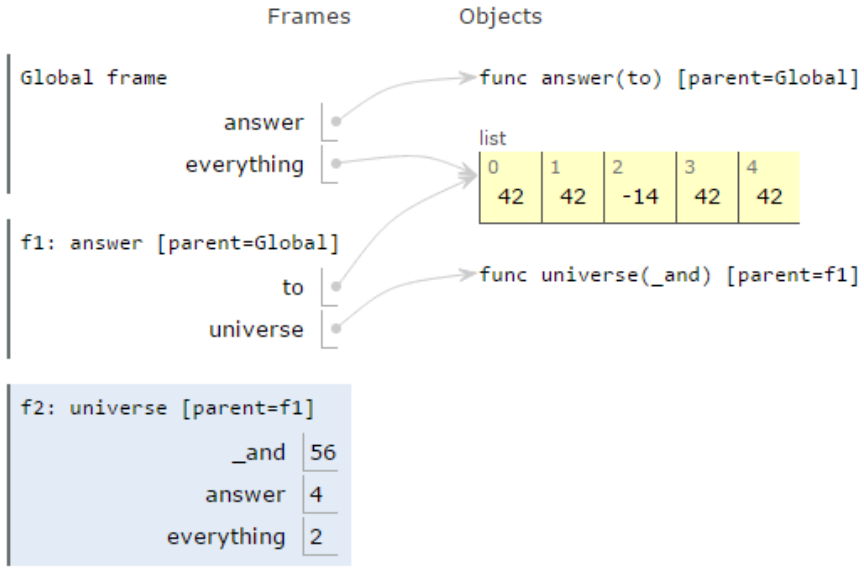
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 20 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

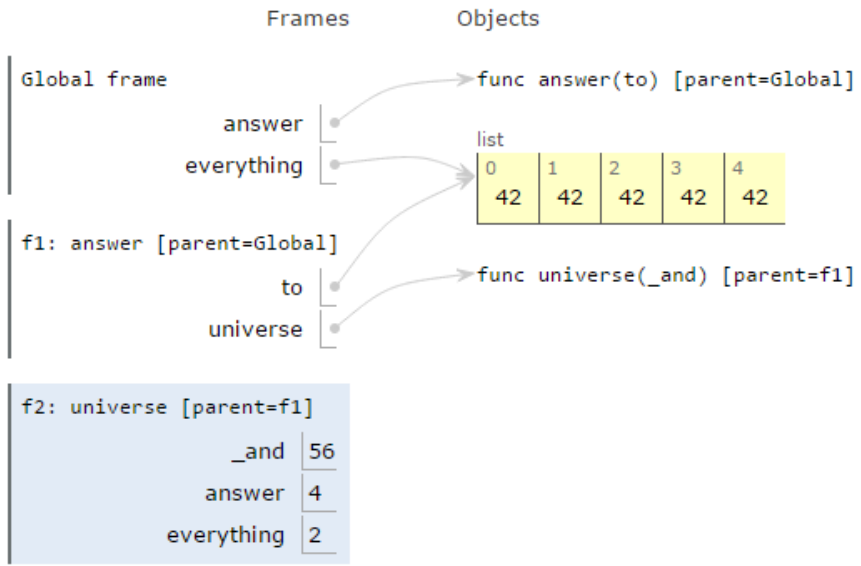
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 21 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

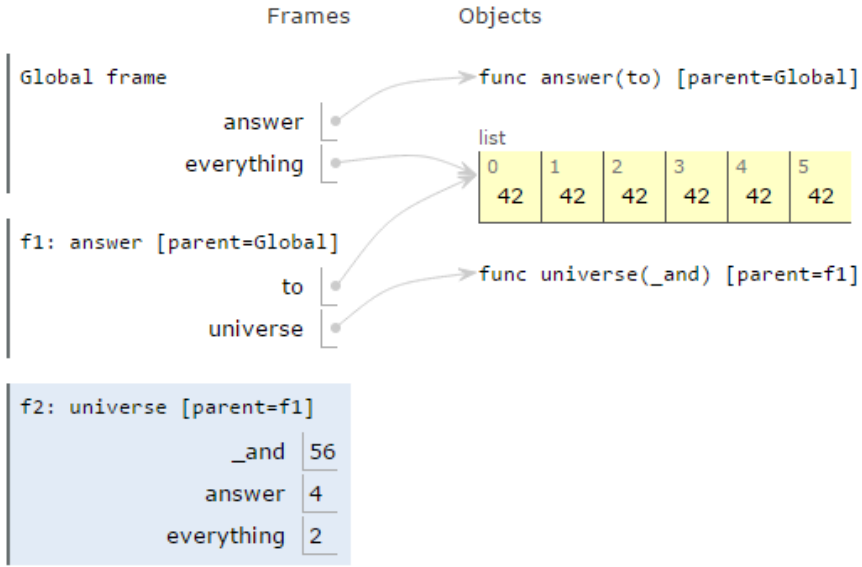
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 22 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

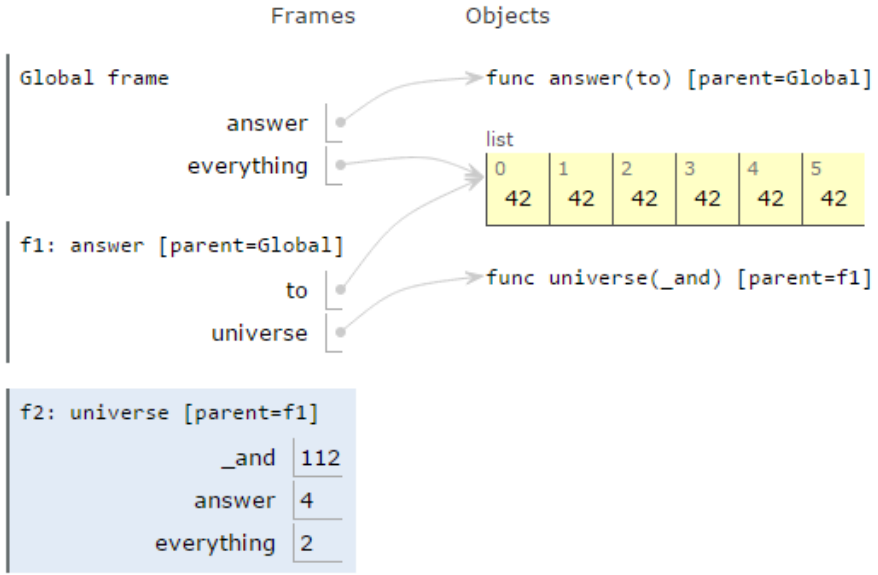
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 23 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

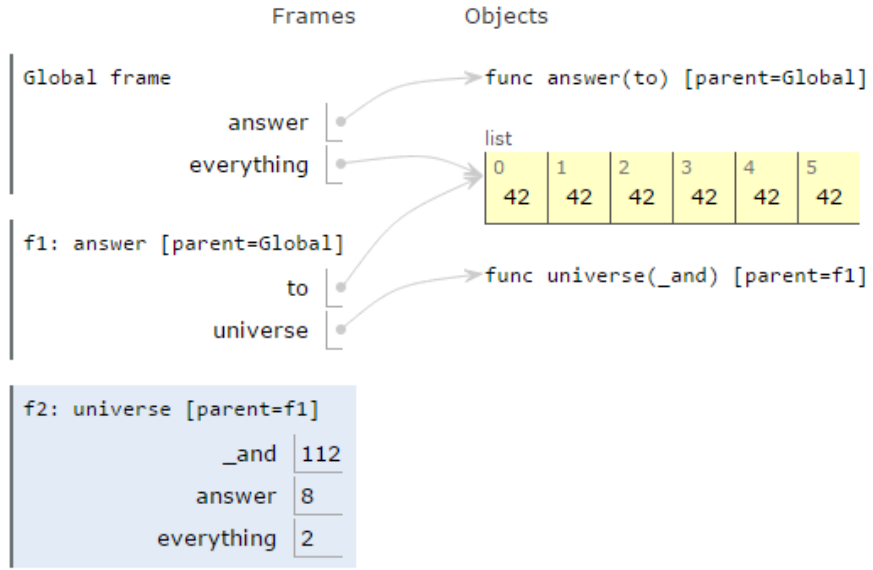
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 24 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

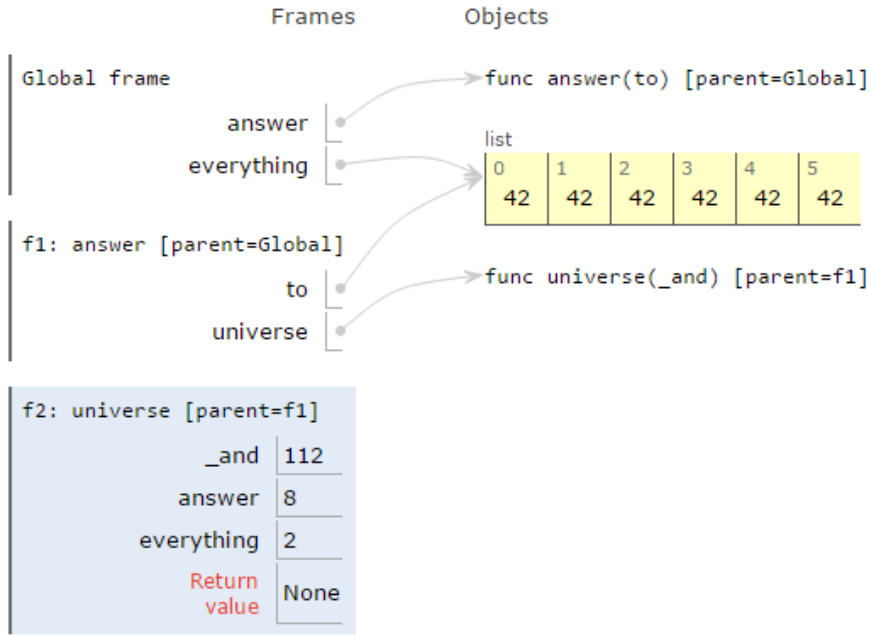
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 25 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Environment Diagrams

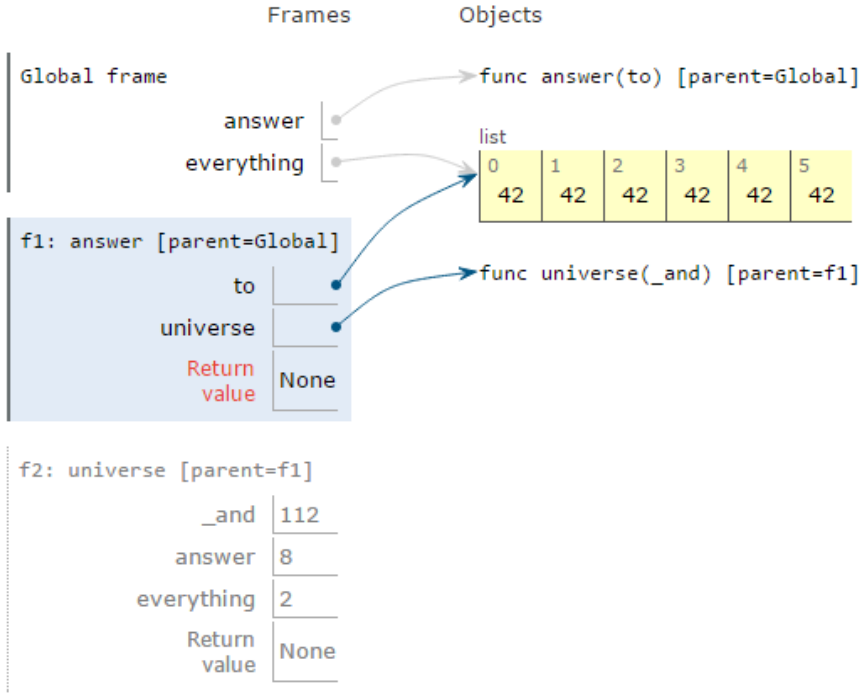
Python 3.3

```
1  def answer(to):
2      def universe(_and):
3          nonlocal to
4          answer = 1
5          for everything in range(len(to)):
6              to[everything] += _and
7              to += [_and//answer * 3]
8              _and *= 2
9              answer *= 2
10     return universe(14)
11
12 everything = [28, 14, -14]
13 answer(everything)
```

Edit code

<< First | < Back | Step 26 of 26 | Forward > | Last >>

line that has just executed
next line to execute



# Linked Lists

# Linked Lists

```
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    def __len__(self):
        return 1 + len(self.rest)


    def __repr__(self):
        return "Link({}, {})".format(self.first, self.rest)
```

# Linked Lists: Swap Pairs

Write the function `swap_pairs` which will take in a linked list and swap every pair of entries. (Assume there is an even number of entries.)

```
def swap_pairs(lst):
    """
    >>> a = Link(2, Link(1, Link(4, Link(3, Link(6, Link(5))))))
    >>> swap_pairs(a)
    >>> a
    Link(1, Link(2, Link(3, Link(4, Link(5, Link(6, ()))))))
    """
```

# Linked Lists: Swap Pairs Solution

```
def swap_pairs(lst):
    if lst != Link.empty:
        lst.first, lst.rest.first = lst.rest.first, lst.first
        swap_pairs(lst.rest.rest)
```

# Linked Lists: Swap Pairs Solution

```
def swap_pairs(lst):
    if lst != Link.empty:
        lst.first, lst.rest.first = lst.rest.first, lst.first
        swap_pairs(lst.rest.rest)
```

Iterative Solution:

```
def swap_pairs(lst):
    while lst != Link.empty:
        lst.first, lst.rest.first = lst.rest.first, lst.first
        lst = lst.rest.rest
```

# Swap Pairs without Mutation

What if don't want to the change the original list?

```
def swap_pairs(lst):
    if lst == Link.empty:
        return lst
    return Link(lst.rest.first, Link(lst.first,
                                     swap_pairs(lst.rest.rest)))
```

# Linked Lists: Double Double

```
def double_double(lst):
    """
    >>> a = Link(1, Link(2, Link(3)))
    >>> double_double(a)
    >>> a
    Link(2, Link(2, Link(4, Link(4, Link(6, Link(6, ()))))))
    """
```

# Linked Lists: Double Double

Fill in the blank

```
def double_double(lst):
    if lst != Link.empty:
        lst.first = _________
        double_double(_________)
        lst.rest = Link(_________, lst.rest)
```

# Linked Lists: Double Double

Fill in the blank

```
def double_double(lst):
    if lst != Link.empty:
        lst.first = 2*lst.first
        double_double(________)
        lst.rest = Link(_________, lst.rest)
```

# Linked Lists: Double Double

Fill in the blank

```
def double_double(lst):
    if lst != Link.empty:
        lst.first = 2*lst.first
        double_double(lst.rest)
        lst.rest = Link(_________, lst.rest)
```

# Linked Lists: Double Double

Fill in the blank

```
def double_double(lst):
    if lst != Link.empty:
        lst.first = 2*lst.first
        double_double(lst.rest)
        lst.rest = Link(lst.first, lst.rest)
```

# Trees

# Trees

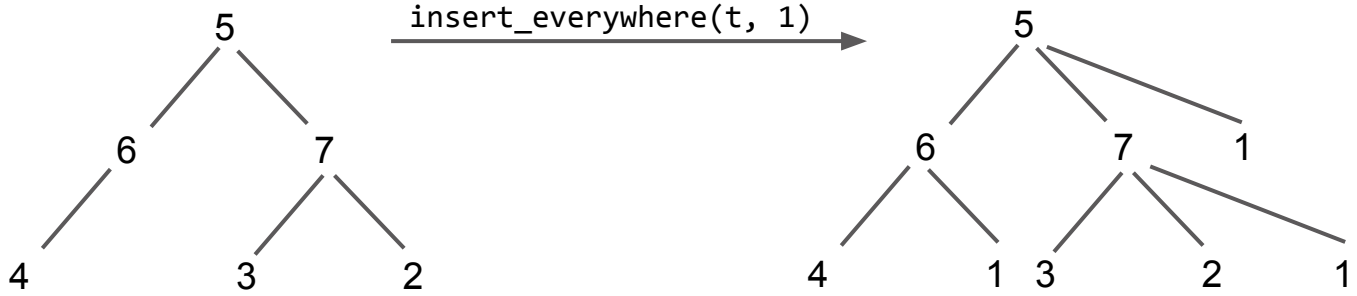
```
class Tree:
    def __init__(self, entry, branches=()):
        self.entry = entry
        self.branches = branches

    def __repr__(self):
        if self.branches:
            return 'Tree({0}, {1})'.format(repr(self.entry), repr(self.branches))
        else:
            return 'Tree({0})'.format(repr(self.entry))
```

# Trees: Insert Everywhere

Define `insert_everywhere`, a function that will add a node with the given value as a child of every internal (non-leaf) node of a tree.

```
def insert_everywhere(t, val):
```



# Trees: Insert Everywhere

Complete the implementation below.

```
def insert_everywhere(t, val):
    if not t.branches:
        return
    for child in t.branches:
        insert_everywhere(child, val)
    _____________________________________________________________
```

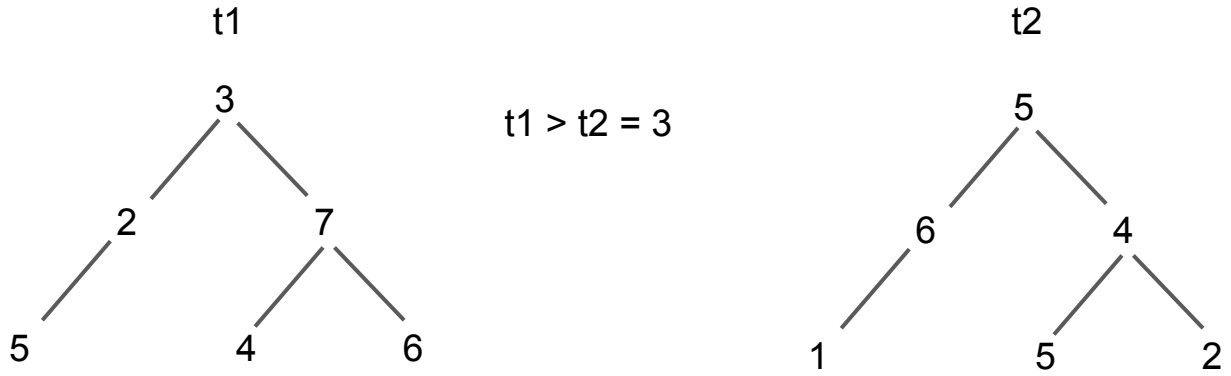
# Trees: Insert Everywhere

```
def insert_everywhere(t, val):
    if not t.branches:
        return
    for child in t.branches:
        insert_everywhere(child, val)
    t.branches.append(Tree(val))
```

# Trees: Greater Than

Write a function that compares two trees of identical structure, returning the number of nodes from t1 that have larger entries than the corresponding nodes in t2.

```
def tree_greater_than(t1, t2):
```



# Trees: Greater Than Solution

```
def tree_greater_than(t1, t2):
    if t1.entry > t2.entry:
        count = 1
    else:
        count = 0
    for i in range(len(t1.branches)):
        count += tree_greater_than(t1.branches[i],
                                   t2.branches[i])
    return count
```

# Orders of Growth

# Orders of Growth

The limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions

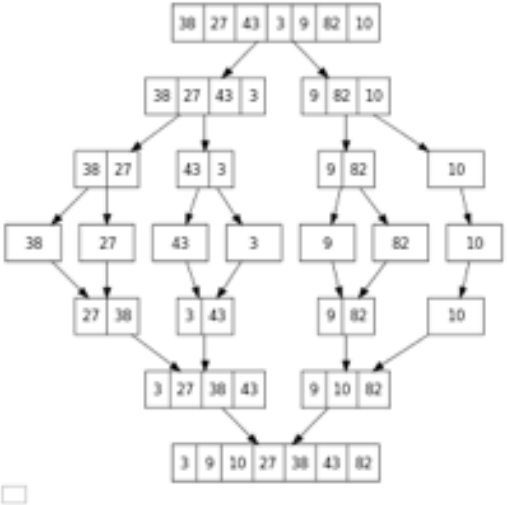
Big Θ notation is used to classify algorithms by how they respond (e.g., in their processing time or working space requirements) to changes in input size.

# Orders of Growth - Merge Sort

```
def merge_sort(m):
    if len(m) <= 1:
        return m
    middle = len(m) // 2
    left = merge_sort(m[:middle])
    right = merge_sort(m[middle:])
    result = merge(left, right)
    return result
```

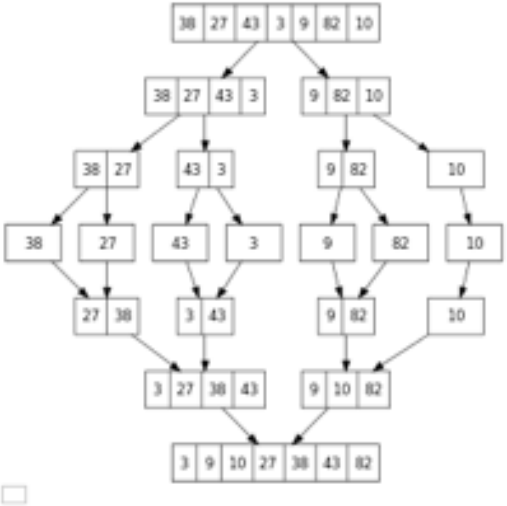
# Orders of Growth - Merge Sort

```
def merge_sort(m):
    if len(m) <= 1:
        return m
    middle = len(m) // 2
    left = merge_sort(m[:middle])
    right = merge_sort(m[middle:])
    result = merge(left, right)
    return result
```



# Orders of Growth - Merge Sort

```
def merge_sort(m):
    if len(m) <= 1:
        return m
    middle = len(m) // 2
    left = merge_sort(m[:middle])
    right = merge_sort(m[middle:])
    result = merge(left, right)
    return result
```




Θ(nlogn)

# Orders of Growth

```
def G(n):
    if n == 1:
        return
    s = 0
    for i in range(n):
        s += G(n-1)
    return s
```

# Orders of Growth

```
def G(n):
    if n == 1:
        return
    s = 0
    for i in range(n):
        s += G(n-1)
    return s
```

We make n calls to G(n-1), each of which makes n-1 calls to G(n-2), each of which makes n-2 calls to G(n-3), and so on until we reach n == 1.
So, the number of calls to G(n) = n * (n-1) * (n-2) …. = n! = Θ(n!)

# Object-Oriented Programming

# Object Oriented Programming

- Objects: an abstract data type (ADT)
- Lets us structure our data

# OOP: Variables

- Class Variables
  - Associated with the class itself
- Instance Variables
  - Associated with an instance of the class
- Local Variables
  - Variables that are local to a method

# OOP: Variables

```
class Foo(object):
    class_var = 1
    def __init__(self):
        self.inst_var = 2
    def bar(self):
        local_var = 3
```

```
>>> Foo.class_var
1
>>> Foo.inst_var
Error
>>> f = Foo()
>>> f.class_var
1
>>> f.instance_var
2
>>> f.local_var
Error
```

# OOP: What would Python print?

```
class Plant(object):
    color = 'green'
    def __init__(self, color):
        self.color = color
        self.seeds = 0

    def fruit(self):
        self.seeds += 1

class BlueBerry(Plant):
    def __init__(self):
        Plant.__init__(self, 'blue')

    def fruit(self):
        self.seeds += 5
```

```
>>> Plant.color
???
>>> Plant.seeds
???
>>> BlueBerry.seeds
???
>>> b = BlueBerry()
>>> b.color
???
>>> BlueBerry.color
???
>>> b.seeds
???
>>> b.fruit()
>>> b.seeds
???
```

# OOP: What would Python print?

```
class Plant(object):
    color = 'green'
    def __init__(self, color):
        self.color = color
        self.seeds = 0

    def fruit(self):
        self.seeds += 1

class BlueBerry(Plant):
    def __init__(self):
        Plant.__init__(self, 'blue')

    def fruit(self):
        self.seeds += 5
```

```
>>> Plant.color
'green'
>>> Plant.seeds
Error
>>> BlueBerry.seeds
Error
>>> b = BlueBerry()
>>> b.color
'blue'
>>> BlueBerry.color
'green'
>>> b.seeds
0
>>> b.fruit()
>>> b.seeds
5
```

# Streams

# Streams

- Streams are a way to represent infinite (or very long) sequences

# Streams

Write a procedure `combine_streams` that takes in two (infinite) streams `s1`, `s2`, and a two-argument function combiner returns a new stream that is the result of adding elements from `s1` by elements from `s2`. For instance, if `s1` was (1, 2, 3, ...), `s2` was (2, 4, 6, ...), and combiner was `lambda x, y: x * y` then the output would be the stream (2, 8, 18, ...).

```
def combine_streams(s1, s2, combiner):
```

# Streams

Write a procedure `combine_streams` that takes in two (infinite) streams `s1`, `s2`, and a two-argument function combiner returns a new stream that is the result of adding elements from `s1` by elements from `s2`. For instance, if `s1` was (1, 2, 3, ...), `s2` was (2, 4, 6, ...), and combiner was `lambda x, y: x * y` then the output would be the stream (2, 8, 18, ...).

```
def combine_streams(s1, s2, combiner):
    def compute_rest():
        return combine_streams(s1.rest, s2.rest, combiner)
    return Stream(combiner(s1.first, s2.first), compute_rest)
```

# Streams

Write a procedure `loopify` that takes as input a **finite** stream and returns an infinite stream with that stream infinitely repeated. For example, if `stream` were a stream `(1, 2, 3)`, loopify would return a stream `(1, 2, 3, 1, 2, 3, 1, 2, 3, …)`

```
def loopify(stream):
```

# Streams

Write a procedure `loopify` that takes as input a **finite** stream and returns an infinite stream with that stream infinitely repeated. For example, if `stream` were a stream `(1, 2, 3)`, loopify would return a stream `(1, 2, 3, 1, 2, 3, 1, 2, 3, …)`

```
def loopify(stream):
    first_stream = Stream(stream.first, lambda: next_stream(stream.rest))
    def next_stream(rest):
        if rest == Stream.empty:
            return first_stream
        return Stream(rest.first, lambda: next_stream(rest.rest))
    return first_stream
```

# Iterators & Generators

# Iterators/Generators

- An **iterable**
  - is an object that has an `__iter__` method which returns an **iterator**.
- An **iterator**
  - is an object that can be iterated over using its `__next__` method.
  - must implement both `__next__` and `__iter__`

Useful analogy: a book is an **iterable**; a bookmark is an **iterator**.

- A **generator** is
  - an **iterator** returned by a **generator function**
  - a call to `__next__` on a generator executes the function's body until it reaches the **yield** and then pauses there until the next call.
- A **generator function** is
  - a function that contains a **yield** statement to return a value

# Iterators/Generators

```
class StrangeIterator:
      def __init__(self):
            """ YOUR CODE HERE """
      def __next__(self):
            """ YOUR CODE HERE """
      def __iter__(self):
            """ YOUR CODE HERE """
>>> strange_obj = StrangeIterable()
>>> elems = []
>>> for i in strange_obj:
...     elems.append(i)
>>> elems
[1, 3, 6, 10, 15, 21, 28, 36, 45]
```

```
class StrangeIterable:
      def __init__(self):
            pass
      def __iter__(self):
            """ YOUR CODE HERE """
```

Any **iterable** object must have a __iter__ that returns an **iterator** which must have a __next__.

# Iterators/Generators

```
class StrangeIterator:
    def __init__(self):
        self.start = 0
        self.step = 1
    def __next__(self):
        if self.step >= 10:
            raise StopIteration
        self.start += self.step
        self.step += 1
        return self.start
    def __iter__(self):
        return self
```

```
class StrangeIterable:
    def __init__(self):
        pass
    def __iter__(self):
        return StrangeIterator()
```

# Iterators/Generators

```
def mystery_gen():
    """
    >>> mg = mystery_gen()
    >>> next(mg)
    [1]
    >>> next(mg)
    [2, 2]
    >>> next(mg)
    [4, 4, 4, 4]
    >>> next(mg)
    [8, 8, 8, 8, 8, 8, 8, 8]
    >>> next(mg)
    Traceback (most recent call last):
      ...
    StopIteration
    """
```

# Iterators/Generators

```
def mystery_gen():
    n_of_n = [1]
    while n_of_n[0] < 9:
        yield n_of_n
        next_n = n_of_n[0] * 2
        n_of_n = [next_n] * next_n
```

# Scheme

# What Would Scheme Print?

```
scm> (cons `(list 1 2 3) (cons 4 (cons 5 nil)))
_____________________________________
scm> (or `false #f 0)
_____________________________________
```

# What Would Scheme Print?

```
scm> (cons `(list 1 2 3) (cons 4 (cons 5 nil)))
((list 1 2 3) 4 5)
scm> (or `false #f 0)
_____________________________
```

# What Would Scheme Print?

```
scm> (cons `(list 1 2 3) (cons 4 (cons 5 nil)))
((list 1 2 3) 4 5)
scm> (or `false #f 0)
false
```

# What Would Scheme Print?

```
scm> (define magic ((lambda (x) (lambda (y) (* x y))) 3))
_____________________________________
scm> (magic 4)
_____________________________________
```

# What Would Scheme Print?

```
scm> (define magic ((lambda (x) (lambda (y) (* x y))) 3))
magic
scm> (magic 4)
_____________________________________
```

# What Would Scheme Print?

```
scm> (define magic ((lambda (x) (lambda (y) (* x y))) 3))
magic
scm> (magic 4)
12
```

# What Would Scheme Print?

```
scm> (define f (mu (x) (* x y)))
f
scm> (define g (mu (x y z) (list (f z) w (f x))))
g
scm> (define h (lambda (w x y) (* (car (g w w x)) (f x))))
h
scm> (h 2 3 4)
```

_____

# What Would Scheme Print?

```
scm> (define f (mu (x) (* x y)))
f
scm> (define g (mu (x y z) (list (f z) w (f x))))
g
scm> (define h (lambda (w x y) (* (car (g w w x)) (f x))))
h
scm> (h 2 3 4)
(* (car (g 2 2 3)) (f 3))
→ (* (car (list (f 3) 2 (f 2))) (f 3))
→ (* (car (list 6 2 4)) (f 3))
→ (* 6 12)
→ 72
```

# Scheme

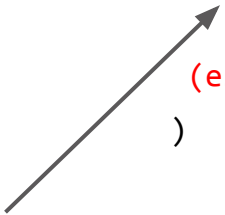
Implement `deep-remove-all`, which removes all instances of `val` from the given `lst`, which may contain nested lists. Assume all of the elements are integers.

```
(define (deep-remove-all val lst)
    `YOUR-CODE-HERE)
```

```
scm> (deep-remove-all 3 `(8 (1 3 3 3 2) 3 (4 3 (3 2 (3 1)))))
(8 (1 2) (4 (2 (1))))
```

# Scheme

```
(define (deep-remove-all val lst)
     (cond ((null? lst) lst)
           ((list? (car lst))
                 (cons (deep-remove-all val (car lst)) (deep-remove-all val (cdr lst))))
           ((= val (car lst))
                 (deep-remove-all val (cdr lst)))
           (else (cons (car lst) (deep-remove-all val (cdr lst)))))
          )
```



It's okay to use = here since we were guaranteed the elements were integers.

eq? and equal? would work too.

# SQL

# SQL

```
create table costs as
      select "Warbot" as name,     1 as cost union
      select "Puddlestomper",      2         union
      select "Blingtron 3000",     5         union
      select "Annoy-o-tron",       1         union
      select "Jeeves",             3         union
      select "Madder Bomber",      5         union
      select "Piloted Shredder",   4;
```

```
create table attacks as
      select "Warbot" as name,     1 as attack union
      select "Puddlestomper",      3           union
      select "Blingtron 3000",     3           union
      select "Annoy-o-tron",       1           union
      select "Jeeves",             1           union
      select "Madder Bomber",      5            union
      select "Piloted Shredder",   4;
```

```
create table armors as
      select "Warbot" as name,     3 as armor  union
      select "Puddlestomper",      2           union
      select "Blingtron 3000",     4           union
      select "Annoy-o-tron",       2           union
      select "Jeeves",             4           union
      select "Madder Bomber",      4            union
      select "Piloted Shredder",   3;
```

#1: Write a SQL statement to create a new table called `cards` that combines all 3 tables.

#2: Write a SQL query to get all of the cards whose **attack** is less than 4 and whose **armor** is greater than 2, in ascending order of **cost**.

# SQL

#1: Write a SQL statement to create a new table called `cards` that combines all 3 tables.

# SQL

#1: Write a SQL statement to create a new table called `cards` that combines all 3 tables.

```
sqlite> create table cards as
          select costs.name, cost, attack, armor from costs, attacks, armors
          where costs.name = attacks.name and attacks.name = armors.name;
```

**The table looks something like this:**

```
create table cards as
     select "Warbot" as name,    1 as cost, 1 as attack, 3 as armor union
     select "Puddlestomper",     2,         3,           2          union
     select "Blingtron 3000",    5,         3,           4          union
     select "Annoy-o-tron",      1,         1,           2          union
     select "Jeeves",            3,         1,           4          union
     select "Madder Bomber",     5,         5,           4          union
     select "Piloted Shredder",  4,         4,           3;
```

# SQL

#2: Write a SQL query to output the **name** and **cost** of all cards whose **attack** is less than 4 and whose **armor** is greater than 2, in ascending order of **cost**.

# SQL

#2: Write a SQL query to output the **name** and **cost** of all cards whose **attack** is less than 4 and whose **armor** is greater than 2, in ascending order of **cost**.
Hint: Use the table you wrote in problem #1.

# SQL

#2: Write a SQL query to output the **name** and **cost** of all cards whose **attack** is less than 4 and whose **armor** is greater than 2, in ascending order of **cost**.

Hint: Use the table you wrote in problem #1.

```
sqlite> select name, cost from cards
            where attack < 4 and armor > 2
            order by cost;
Warbot|1
Jeeves|3
Blingtron-3000|5
```

# SQL

#2: Write a SQL query to output the **name** and **cost** of all cards whose **attack** is less than 4 and whose **armor** is greater than 2, in ascending order of **cost**.

Hint: Use the table you wrote in problem #1.

```
sqlite> select name, cost from cards
            where attack < 4 and armor > 2
            order by cost;
Warbot|1
Jeeves|3
Blingtron-3000|5
```

Alternate solution without the table from problem #1 written already:

```
sqlite> with
          cards(name, cost, attack, armor) as (
             select costs.name, cost, attack, armor from costs, attacks, armors
             where costs.name = attacks.name and attacks.name = armors.name
        )
        select name, cost from cards
           where attack < 4 and armor > 2
           order by cost;
```

# SQL

```
create table costs as
      select "Warbot" as name,     1 as cost union
      select "Puddlestomper",      2         union
      select "Blingtron 3000",     5         union
      select "Annoy-o-tron",       1         union
      select "Jeeves",             3         union
      select "Madder Bomber",      5         union
      select "Piloted Shredder",   4;
create table armors as
      select "Warbot" as name,     3 as armor  union
      select "Puddlestomper",      2           union
      select "Blingtron 3000",     4           union
      select "Annoy-o-tron",       2           union
      select "Jeeves",             4           union
      select "Madder Bomber",      4            union
      select "Piloted Shredder",   3;
```

```
create table attacks as
      select "Warbot" as name,     1 as attack union
      select "Puddlestomper",      3           union
      select "Blingtron 3000",     3           union
      select "Annoy-o-tron",       1           union
      select "Jeeves",             1           union
      select "Madder Bomber",      5            union
      select "Piloted Shredder",   4;
```

#3: Write a SQL query that outputs the names of a pair of cards **a** and **b** where **a.attack >= b.armor** and **b.attack >= a.armor.**
**Do NOT use the table from #1.**

**Expected output:**

Blingtron 3000 trades with Piloted Shredder
Madder Bomber trades with Madder Bomber
Madder Bomber trades with Piloted Shredder
Piloted Shredder trades with Blingtron 3000
Piloted Shredder trades with Madder Bomber
Piloted Shredder trades with Piloted Shredder
Piloted Shredder trades with Puddlestomper
Puddlestomper trades with Puddlestomper
Puddlestomper trades with Piloted Shredder

(Fun fact: In Hearthstone, this is called a trade, since both cards die as a result of one card attacking the other.)

# SQL

#3: Write a SQL query that outputs the names of a pair of cards **a** and **b** where **a.attack >= b.armor** and **b.attack >= a.armor.**
**Do NOT use the table from #1.**

```
sqlite> select a_att.name || " trades with " || b_att.name
           from attacks as a_att, attacks as b_att, armors as a_arm, armors as b_arm
           where a_att.attack >= b_arm.armor and b_att.attack >= a_arm.armor
              and a_att.name = a_arm.name and b_att.name = b_arm.name;

Blingtron 3000 trades with Piloted Shredder
Madder Bomber trades with Madder Bomber
Madder Bomber trades with Piloted Shredder
Piloted Shredder trades with Blingtron 3000
Piloted Shredder trades with Madder Bomber
Piloted Shredder trades with Piloted Shredder
Piloted Shredder trades with Puddlestomper
Puddlestomper trades with Puddlestomper
Puddlestomper trades with Piloted Shredder
```

# SQL

```
create table costs as
      select "Warbot" as name,    1 as cost union
      select "Puddlestomper",     2         union
      select "Blingtron 3000",    5         union
      select "Annoy-o-tron",      1         union
      select "Jeeves",            3         union
      select "Madder Bomber",     5         union
      select "Piloted Shredder",  4;
```

#4: Write a SQL query that outputs all subsets and their total costs of cards whose total costs are at least 7.

(Hint #1: Use recursion!)
(Hint #2: it might help to put the cards of each subset in a particular order!)

# SQL

#4: Write a SQL query that outputs all subsets and their total costs of cards whose total costs are at least 7.
(Hint #1: Use recursion!)
(Hint #2: it might help to put the cards of each subset in a particular order!)

```
sqlite> with sums(names, total, last_cost) as (
          select name, cost, cost from costs union
          select names || ", " || name, total + cost, cost
              from sums, costs
              where cost > last_cost
        )
        select names, total from sums where total >= 7 order by total;
```

# Conclusion

This was HKN's second ever CS 61A Final Review Session. Please fill out the feedback forms to help us improve future reviews.

***Thanks for coming, and best of luck on the final!***