61A Lecture 5

Wednesday, September 10
Announcements
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - [http://cs61a.org/hw/released/quiz1.html](http://cs61a.org/hw/released/quiz1.html)
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
  ▪ Open computer & course materials, but no external resources such as classmates
  ▪ Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
  ▪ Open computer & course materials, but no external resources such as classmates
  ▪ Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html

• "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
  ▪ Open computer & course materials, but no external resources such as classmates
  ▪ Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html

• "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  ▪ http://42.cs61a.org, run by Sumukh Sridhara (TA)
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ http://cs61a.org/hw/released/quiz1.html
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
  ▪ Open computer & course materials, but no external resources such as classmates
  ▪ Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html

• "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  ▪ http://42.cs61a.org, run by Sumukh Sridhara (TA)

• Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  ▪ 3 points; graded for correctness
  ▪ Submit in the same way that you submit homework assignments
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped
  ▪ Open computer & course materials, but no external resources such as classmates
  ▪ Practice quiz from Fall 2013: [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html)
• "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  ▪ [http://42.cs61a.org](http://42.cs61a.org), run by Sumukh Sridhara (TA)
• Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
• Homework 2 (which is small) due Monday 9/15 at 11:59pm.
Announcements

• Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
• "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)
• Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
• Homework 2 (which is small) due Monday 9/15 at 11:59pm.
• Project 1 (which is BIG) due Wednesday 9/17 at 11:59pm.
Office Hours: You Should Go!
Office Hours: You Should Go!

You are not alone!
Office Hours: You Should Go!

You are not alone!
Office Hours: You Should Go!

You are not alone!

http://cs61a.org/staff.html
Environments for Higher-Order Functions
Environments Enable Higher-Order Functions
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**

- Express general methods of computation
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**

- Express general methods of computation
- Remove repetition from programs
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**
- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

*Environment diagrams describe how higher-order functions work!*
Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

*Environment diagrams describe how higher-order functions work!*

(Demo)
Names can be Bound to Functional Arguments

```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```
Names can be Bound to Functional Arguments

```python
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

Interactive Diagram
Names can be Bound to Functional Arguments

```python
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

Applying a user-defined function:
- Create a new frame
- Bind formal parameters \( f \) and \( x \) to arguments
- Execute the body:
  
  ```python
  return f(f(x))
  ```
Names can be Bound to Functional Arguments

```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```

Applying a user-defined function:
- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body:
  return f(f(x))

Interactive Diagram
Names can be Bound to Functional Arguments

def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
Names can be Bound to Functional Arguments

```python
1 def apply_twice(f, x):
2     return f(f(x))

4 def square(x):
5     return x * x

7 result = apply_twice(square, 2)
```
Discussion Question

What is the value of the final expression below? (Demo)
Discussion Question

What is the value of the final expression below? (Demo)

```python
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

Interactive Diagram
Discussion Question

What is the value of the final expression below? (Demo)

def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
Discussion Question

What is the value of the final expression below? (Demo)

```python
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

If you think there's an error
Environments for Nested Definitions

(Demo)
Environment Diagrams for Nested Def Statements

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```

Interactive Diagram
Environment Diagrams for Nested Def Statements

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```
Environment Diagrams for Nested Def Statements

```python
# Nested def

def make_adder(n):
    def adder(k):
        return k + n
    return adder

6 add_three = make_adder(3)
7 add_three(4)
```

Interactive Diagram
Environment Diagrams for Nested Def Statements

```python
1  def make_adder(n):
2      def adder(k):
3          return k + n
4      return adder
5
6  add_three = make_adder(3)
7  add_three(4)
```
Environment Diagrams for Nested Def Statements

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```

Interactive Diagram
Environment Diagrams for Nested Def Statements

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```

Interactive Diagram
Environment Diagrams for Nested Def Statements

1 def make_adder(n):
2     def adder(k):
3         return k + n
4     return adder
5
6 add_three = make_adder(3)
7 add_three(4)
Environment Diagrams for Nested Def Statements

```python
# Nested def

def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```

Interactive Diagram
Every user-defined function has a parent frame (often global)
Environment Diagrams for Nested Def Statements

```
1  def make_adder(n):
2      def adder(k):
3          return k + n
4      return adder
5
6  add_three = make_adder(3)
7  add_three(4)
```

- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined

Interactive Diagram
Every user-defined function has a parent frame (often global).
The parent of a function is the frame in which it was defined.
Every local frame has a parent frame (often global).
Environment Diagrams for Nested Def Statements

- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame (often global)
- The parent of a frame is the parent of the function called

```
1 def make_adder(n):
2     def adder(k):
3         return k + n
4     return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```
How to Draw an Environment Diagram
How to Draw an Environment Diagram

When a function is defined:
How to Draw an Environment Diagram

When a function is defined:

Create a function value:  func <name>(<formal parameters>) [parent=<label>]
How to Draw an Environment Diagram

When a function is defined:

Create a function value: func <name>(<formal parameters>) [parent=<label>]

Its parent is the current frame.
How to Draw an Environment Diagram

When a function is defined:

Create a function value:   func <name>(<formal parameters>) [parent=<label>]

Its parent is the current frame.

\[\begin{array}{c}
\text{f1: make_adder} \\
\text{func adder(k) [parent=f1]}
\end{array}\]
How to Draw an Environment Diagram

When a function is defined:

Create a function value:   `func <name>(<formal parameters>) [parent=<label>]`

Its parent is the current frame.

```
f1: make_adder       func adder(k) [parent=f1]
```

Bind <name> to the function value in the current frame
How to Draw an Environment Diagram

When a function is defined:
Create a function value:    func <name>(<formal parameters>) [parent=<label>]
Its parent is the current frame.

Bind <name> to the function value in the current frame

When a function is called:
How to Draw an Environment Diagram

When a function is defined:
Create a function value:   \texttt{func <name>(<formal parameters>) [parent=<label>]} \\
Its parent is the current frame.

\begin{itemize}
  \item \texttt{f1: make_adder}
  \item \texttt{func adder(k) [parent=f1]}
\end{itemize}

Bind \texttt{<name>} to the function value in the current frame

When a function is called:
1. Add a local frame, titled with the \texttt{<name>} of the function being called.
How to Draw an Environment Diagram

When a function is defined:

Create a function value:  `func <name>(<formal parameters>) [parent=<label>]`

Its parent is the current frame.

```
f1: make_adder       func adder(k) [parent=f1]
```

Bind `<name>` to the function value in the current frame

When a function is called:

1. Add a local frame, titled with the `<name>` of the function being called.

2. Copy the parent of the function to the local frame: `[parent=<label>]`
How to Draw an Environment Diagram

When a function is defined:

Create a function value:  \texttt{func <name>(<formal parameters>) [parent=<label>]}

Its parent is the current frame.

\texttt{f1: make_adder\hspace{1cm} func adder(k) [parent=f1]}

Bind <name> to the function value in the current frame

When a function is called:

1. Add a local frame, titled with the <name> of the function being called.
2. Copy the parent of the function to the local frame: [parent=<label>]
3. Bind the <formal parameters> to the arguments in the local frame.
How to Draw an Environment Diagram

When a function is defined:

Create a function value: `func <name>(<formal parameters>) [parent=<label>]`

Its parent is the current frame.

Bind `<name>` to the function value in the current frame

When a function is called:

1. Add a local frame, titled with the `<name>` of the function being called.
2. Copy the parent of the function to the local frame: `[parent=<label>]`
3. Bind the `<formal parameters>` to the arguments in the local frame.
4. Execute the body of the function in the environment that starts with the local frame.
Local Names

(Demo)
Local Names are not Visible to Other (Non-Nested) Functions

```
1 def f(x, y):
2     return g(x)
3
4 def g(a):
 5     return a + y
6
7 result = f(1, 2)
```

Interactive Diagram
Local Names are not Visible to Other (Non-Nested) Functions

```python
1  def f(x, y):
2      return g(x)
3
4  def g(a):
5      return a + y
6
7  result = f(1, 2)
```

**Interactive Diagram**
Local Names are not Visible to Other (Non-Nested) Functions

```python
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

Interactive Diagram
Local Names are not Visible to Other (Non-Nested) Functions

```
1 def f(x, y):
2     return g(x)
3  
4 def g(a):
5     return a + y
6  
7 result = f(1, 2)
```

"y" is not found
Local Names are not Visible to Other (Non-Nested) Functions

```python
1 def f(x, y):
2     return g(x)
3
4 def g(a):
5     return a + y
6
7 result = f(1, 2)
```

Interactive Diagram
Local Names are not Visible to Other (Non-Nested) Functions

```python
1 def f(x, y):
2     return g(x)
3
4 def g(a):
5     return a + y
6
7 result = f(1, 2)
```

"y" is not found

"y" is not found, again

Interactive Diagram
Local Names are not Visible to Other (Non-Nested) Functions

```
def f(x, y):
    return g(x)
def g(a):
    return a + y
result = f(1, 2)
```

• An environment is a sequence of frames.
Local Names are not Visible to Other (Non-Nested) Functions

```
1 def f(x, y):
2     return g(x)
3
4 def g(a):
5     return a + y
6
7 result = f(1, 2)
```

- An environment is a sequence of frames.
- The environment created by calling a top-level function (no def within def) consists of one local frame, followed by the global frame.

Interactive Diagram
Function Composition

(Demo)
The Environment Diagram for Function Composition

```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Interactive Diagram
The Environment Diagram for Function Composition

```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose(f, g):
    def h(x):
        return f(g(x))
    return h

compose(square, make_adder(2))(3)
```

Interactive Diagram
The Environment Diagram for Function Composition

```python
1 def square(x):
2     return x * x
3
4 def make_adder(n):
5     def adder(k):
6         return k + n
7     return adder
8
9 def compose1(f, g):
10    def h(x):
11        return f(g(x))
12    return h
13
14 compose1(square, make_adder(2))(3)
```
The Environment Diagram for Function Composition

```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Interactive Diagram
The Environment Diagram for Function Composition

```
1. def square(x):
   return x * x

2. def make_adder(n):
   def adder(k):
      return k + n
   return adder

3. def compose1(f, g):
   def h(x):
      return f(g(x))
   return h

4. compose1(square, {make_adder(2)})(3)
```

Return value of make_adder is an argument to compose1
The Environment Diagram for Function Composition

```python
1  def square(x):
2      return x * x
3
4  def make_adder(n):
5      def adder(k):
6          return k + n
7          return adder
8
9  def compose1(f, g):
10     def h(x):
11        return f(g(x))
12        return h
13
14  compose1(square, {make_adder(2)})(3)
```

Return value of `make_adder` is an argument to `compose1`
The Environment Diagram for Function Composition

```python
1. def square(x):
   return x * x

2. def make_adder(n):
   def adder(k):
       return k + n
   return adder

3. def compose1(f, g):
   def h(x):
       return f(g(x))
   return h

4. compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1
```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1
The Environment Diagram for Function Composition

```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Return value of `make_adder` is an argument to `compose1`
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

c = compose1(square, make_adder(2))(3)

Return value of make_adder is an argument to compose1
The Environment Diagram for Function Composition

```
1 def square(x):
2     return x * x
3
4 def make_adder(n):
5     def adder(k):
6         return k + n
7     return adder
8
9 def compose1(f, g):
10    def h(x):
11        return f(g(x))
12    return h
13
14 compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1