Announcements
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  • [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html)
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html)
  ▪ 3 points; graded for correctness.
Announcements

- Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html)
  - 3 points; graded for correctness.
  - Submit in the same way that you submit homework assignments.
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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:* You can use the Python interpreter, watch course videos, and read the online text ([http://composingprograms.com](http://composingprograms.com)).
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  
  ▪ http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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: You can use the Python interpreter, watch course videos, and read the online text (http://composingprograms.com).

  ▪ No external resources: Please don't search for answers, talk to your classmates, etc.
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  • [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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:* You can use the Python interpreter, watch course videos, and read the online text ([http://composingprograms.com](http://composingprograms.com)).
  • *No external resources:* Please don't search for answers, talk to your classmates, etc.
• Homework 2 due Tuesday 9/17 at 5pm.
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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:* You can use the Python interpreter, watch course videos, and read the online text ([http://composingprograms.com](http://composingprograms.com)).
  ▪ *No external resources:* Please don't search for answers, talk to your classmates, etc.
• Homework 2 due Tuesday 9/17 at 5pm.
• Project 1 due Thursday 9/19 at 11:59pm.
Announcements

• Take-home quiz released Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/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:** You can use the Python interpreter, watch course videos, and read the online text ([http://composingprograms.com](http://composingprograms.com)).
  ▪ **No external resources:** Please don't search for answers, talk to your classmates, etc.

• Homework 2 due Tuesday 9/17 at 5pm.

• Project 1 due Thursday 9/19 at 11:59pm.

• Solutions to homeworks: [http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/solutions](http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/solutions)
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://inst.eecs.berkeley.edu/~cs61a/fa13/staff.html
The Purpose of Higher-Order Functions
The Purpose of Higher-Order Functions

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

**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

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

**Functions as arguments:**
Environments Enable Higher-Order Functions

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

**Functions as arguments**:

Our current evaluation rules handle that case already!
Environments Enable Higher-Order Functions

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

Functions as arguments:

Our current evaluation rules handle that case already!

We'll discuss an example today
**Environments Enable Higher-Order Functions**

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

**Functions as arguments:**

Our current evaluation rules handle that case already!

We'll discuss an example today

**Functions as return values:**
Environments Enable Higher-Order Functions

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

**Functions as arguments:**

Our current evaluation rules handle that case already!

We'll discuss an example today

**Functions as return values:**

We need to extend our rules a little
Environments Enable Higher-Order Functions

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

**Functions as arguments:**

Our current evaluation rules handle that case already!

We'll discuss an example today

**Functions as return values:**

We need to extend our rules a little

Functions need to know where they were defined
Environments Enable Higher-Order Functions

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

**Functions as arguments:**

Our current evaluation rules handle that case already!

We'll discuss an example today.

**Functions as return values:**

We need to extend our rules a little.

Functions need to know where they were defined.

Almost everything stays the same.
Environments Enable Higher-Order Functions

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

**Functions as arguments:**

Our current evaluation rules handle that case already!

We'll discuss an example today

**Functions as return values:**

We need to extend our rules a little

Functions need to know where they were defined

Almost everything stays the same

(demo)
Names can be Bound to Functional Arguments

```python
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Names can be Bound to Functional Arguments

```python
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Names can be Bound to Functional Arguments

```python
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Names can be Bound to Functional Arguments

```python
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Names can be Bound to Functional Arguments

```python
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)
```

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
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
  result = apply_twice(square, 2)

Example:  http://goo.gl/mwVuIF
```
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))`

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Names can be Bound to Functional Arguments

```python
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))`

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
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)

Example: http://goo.gl/mwVuIF

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

```python
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)
```

- Functions are values.

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
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)
```

• Functions are values.
• Names can refer to functions (just as they can refer to any values).

Example: http://goo.gl/mWVuIF
Names can be Bound to Functional Arguments

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

- Functions are values.
- Names can refer to functions (just as they can refer to any values).
- Multiple names can all refer to the same function, even in different frames.

Example: [http://goo.gl/mwVuIF](http://goo.gl/mwVuIF)
Discussion Question

What is the value of the final expression below?

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

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

repeat(g, 5)
```

Example: [http://goo.gl/EDiOIr](http://goo.gl/EDiOIr)
Discussion Question

What is the value of the final expression below?

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

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

repeat(g, 5)
```

Example: [http://goo.gl/EDiOIr](http://goo.gl/EDiOIr)
Discussion Question

What is the value of the final expression below?

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

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

repeat(g, 5)
```

If you think there's an error

Example: [http://goo.gl/EDiOlR](http://goo.gl/EDiOlR)
Environments for Nested Definitions

(Demo)
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)
```

Example:
Environment Diagrams for Nested Def Statements

Example:

```python
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

Example:

1. `def make_adder(n):`
   2. `def adder(k):
      3. return k + n
      4. return adder

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

Example:

```python
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

Example:

```python
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

Example:

```python
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

Example:

```python
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

Example:

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

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

```python
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

Every user-defined function has a parent frame (often global).

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

add_three = make_adder(3)
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

Example:
```python
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

- 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)

Example:

```python
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

• 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

Example:

```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)
```
An Environment is a Sequence of Frames

Global frame

- func make_adder(n)
- func adder(k) [parent=f1]

f1: make_adder

- n
- adder
- Return value

adder [parent=f1]

- k
- Return value
An Environment is a Sequence of Frames

Global frame

make_adder

add_three

func make_adder(n)

func adder(k) [parent=f1]

f1: make_adder

n 3

c
adder
Return value

adder [parent=f1]

k 4

Return value 7
An Environment is a Sequence of Frames

Global frame
  func make_adder(n)
    make_adder
    add_three
  func adder(k) [parent=f1]

f1: make_adder
  n 3
  adder
  Return
  value

adder [parent=f1]
  k 4
  Return
  value 7
An Environment is a Sequence of Frames

A three-frame environment

Global frame

func make_adder(n)

make_adder

func adder(k) [parent=f1]

adder

f1: make_adder

n 3

adder

Return value

adder [parent=f1]

k 4

Return value 7
An Environment is a Sequence of Frames

A two-frame environment

A three-frame environment

Global frame

\[ \text{func make_adder}(n) \]
\[ \text{func adder}(k) \ [\text{parent}=f1] \]

\[ f1: \text{make_adder} \]
\[ n \quad 3 \]
\[ \text{adder} \]
\[ \text{Return value} \]

\[ \text{adder} \ [\text{parent}=f1] \]
\[ k \quad 4 \]
\[ \text{Return value} \quad 7 \]
An Environment is a Sequence of Frames

- A three-frame environment
- A two-frame environment
- The global environment

Global frame
- func make_adder(n)
- func adder(k) [parent=f1]

f1: make_adder
- n 3
  - adder
  - Return
  - value

adder [parent=f1]
- k 4
  - Return
  - value
  - 7
An Environment is a Sequence of Frames

A local frame extends the environment that begins with its parent.
An Environment is a Sequence of Frames

A local frame extends the environment that begins with its parent.
An Environment is a Sequence of Frames

A local frame extends the environment that begins with its parent.

A three-frame environment

The global environment

Always extends

A two-frame environment

Global frame

make_adder

add_three

func make_adder(n)

func adder(k) [parent=f1]

When a frame or function has no parent label [parent=___], then its parent is the global frame.

f1: make_adder

adder

Return value

adder [parent=f1]

k 4

Return value 7
An Environment is a Sequence of Frames

A local frame extends the environment that begins with its parent.
An Environment is a Sequence of Frames

A local frame extends the environment that begins with its parent.

A two-frame environment

A three-frame environment

We don't bother to label frames that aren't parents

Always extends

The global environment

Global frame

make_adder

add_three

func make_adder(n)

func adder(k) [parent=f1]

When a frame or function has no parent label

[parent=___]

then its parent is the global frame

f1: make_adder

n 3

adder

Return value

adder [parent=f1]

k 4

Return value 7
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:

1. Create a function value: func <name>(<formal parameters>)
How to Draw an Environment Diagram

When a function is defined:

1. Create a function value: \( \text{func} \ <\text{name}> (<\text{formal parameters}>) \)

2. If the parent frame of that function is not the global frame, add matching labels to the parent frame and the function value (such as \( f1, f2, \) or \( f3 \)).
How to Draw an Environment Diagram

When a function is defined:

1. Create a function value: \texttt{func <name>(<formal parameters>)}

2. If the parent frame of that function is not the global frame, add matching labels to the parent frame and the function value (such as \texttt{f1}, \texttt{f2}, or \texttt{f3}).

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

When a function is defined:

1. Create a **function value**: \( \text{func } <\text{name}>(<\text{formal parameters}>) \)

2. If the **parent frame** of that function is not the global frame, add matching **labels** to the **parent frame** and the **function value** (such as \( f_1, f_2, \) or \( f_3 \)).

   \[
   \begin{align*}
   f_1: \text{make_adder} & \quad \text{func} \quad \text{adder}(k) \quad [\text{parent}=f_1]
   \end{align*}
   \]

3. Bind \(<\text{name}>\) to the **function value** in the first frame of the current environment.
How to Draw an Environment Diagram

When a function is defined:

1. Create a function value: func <name>(<formal parameters>)

2. If the parent frame of that function is not the global frame, add matching labels to the parent frame and the function value (such as f1, f2, or f3).

3. Bind <name> to the function value in the first frame of the current environment.

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

When a function is defined:

1. Create a **function value**:   \texttt{func <name>(<formal parameters>)}

2. If the **parent frame** of that function is not the global frame, add matching **labels** to the **parent frame** and the **function value** (such as \texttt{f1}, \texttt{f2}, or \texttt{f3}).

   \begin{align*}
   \texttt{f1: make_adder} & \quad \texttt{func adder(k) [parent=f1]} \\
   \end{align*}

3. Bind \texttt{<name>} to the **function value** in the first frame of the current environment.

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:

1. Create a function value: \texttt{func <name>(<formal parameters>)}

2. If the parent frame of that function is not the global frame, add matching labels to the parent frame and the function value (such as \texttt{f1, f2, or f3}).

3. Bind <name> to the function value in the first frame of the current environment.

When a function is called:

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

2. If the function has a parent label, copy it to the local frame: [parent=<label>]

\begin{verbatim}
\hline
f1: make_adder \quad \texttt{func adder(k) [parent=f1]}
\hline
\end{verbatim}
How to Draw an Environment Diagram

When a function is defined:

1. Create a **function value**: `func <name>(<formal parameters>)`

2. If the **parent frame** of that function is not the global frame, add matching **labels** to the **parent frame** and the **function value** (such as `f1`, `f2`, or `f3`).

3. Bind `<name>` to the **function value** in the first frame of the current environment.

When a function is called:

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

2. If the function has a parent label, copy it 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:
1. Create a **function value**: `func <name>(<formal parameters>)`

2. If the **parent frame** of that function is not the global frame, add matching **labels** to the **parent frame** and the **function value** (such as `f1`, `f2`, or `f3`).

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

3. Bind `<name>` to the **function value** in the first frame of the current environment.

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

2. If the function has a parent label, copy it 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

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

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
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)
```

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
Local Names are *not* Visible to Other (Non-Nested) Functions

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

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
Local Names are *not* Visible to Other (Non-Nested) Functions

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
Local Names are not Visible to Other (Non-Nested) Functions

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

4 def g(a):
   return a + y

7 result = f(1, 2)
```

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
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, again

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
Local Names are *not* Visible to Other (Non-Nested) Functions

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

2 def g(a):
    return a + y

3 result = f(1, 2)
```

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
Local Names are *not* Visible to Other (Non-Nested) Functions

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

- An environment is a sequence of frames.

Example: [http://goo.gl/b6WvUc](http://goo.gl/b6WvUc)
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, again

Global frame

| func f(x, y) |
| f |
| g |

Error

1

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Function Composition

(Demo)
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)
```

Example:
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)
```

Example:
The Environment Diagram for Function Composition

Example:

```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

Example:

```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)
```
The Environment Diagram for Function Composition

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
```
The Environment Diagram for Function Composition

Return value of make_adder is an argument to compose1

```
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)
```

Example:
The Environment Diagram for Function Composition

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

Example:

Return value of make_adder is an argument to compose1
Example:

```
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)
```

Example:

Return value of make_adder is an argument to compose1
Return value of make_adder is an argument to compose1
The Game of Hog

(Demo)