Lecture #3: Environments

- Substitution is not as simple as it might seem.
- For example:
  ```python
  def f(x):
      def g(x):
          return x + 10
      return g(5)
  f(3)
  ```
- When we call `f(3)`, we should not substitute 3 for the x in g!
- And there are other difficulties...

Names

- Evaluating expressions that are literals is easy: the literal’s text gives all the information needed.
- But how do I evaluate names like `add`, `mul`, or `print`?
  ```python
  x = 3
  print(x)
  x = 4
  print(x)  # After x = 3, does this x change to 3??!
  ```
- Deduction: there must be another source of information.
- We’ll use the concept of an environment to explain it.

Environments

- An environment is a mapping from names to values.
- We say that a name is bound to a value in this environment.
- Every expression is evaluated in an environment, which supplies the meanings of any names in it.
- Simplest environment consists of a single global environment frame:

  ```
  mul:
  print:
  radius: 10
  square:
  ```

  ```
  λ x, y: ≪ x × y ≫
  λ x: return mul(x, x)
  λ x: ≪ print x ≫
  ```

Evaluation of Names

- To evaluate a name (identifier) in an environment, look for what that name is bound to in that environment.
- For example, in this situation...

Evaluation of Names (II)

... We find the values for square and radius in the global frame (the big box with the globe on its upper right).

Evaluation of Names: More Complicated Environments

- In general, as we’ll see, environments consist of chains of frames.
- Here, we find the value of x in the small, "local frame"
- We don’t find mul, there, so we must follow the "environment link" looking for it.
More Complicated Environments (II)

Evaluating User-Defined Function Calls

• Consider the expression `square(mul(x, x))` in

```python
from operator import mul
def square(x):
    return mul(x, x)
x = -2
print(square(mul(x, 5)))
```

```
mul:  λ x, y: ≪ x × y ≫
print:  λ x: ≪ print x ≫
x: -2
square:  λ x: return mul(x, x)
```

```
λ x: mul(x, x)
λ x: print(x)
```

```
λ x: mul(x, x)
λ x: print(x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mul(x, x)
```

```
λ x: mu...
Evaluating User-Defined Functions Calls (VI)

When we evaluate \( \text{mul}(x, x) \) in this new environment, we get the same value as before for \( \text{mul} \), but the local value for \( x \).

\[
\lambda x, y: \langle x \times y \rangle \\
\lambda x: \langle \text{print } x \rangle
\]

\[\lambda x: \text{mul}(x, x)\]

\[\langle -10 \rangle\]

\[\langle -10 \times -10 \rangle\]

Evaluating User-Defined Functions Calls (VII)

Evaluate the primitive multiplication as before:

\[
\lambda x, y: \langle x \times y \rangle \\
\lambda x: \langle \text{print } x \rangle
\]

\[\lambda x: \text{mul}(x, x)\]

\[\langle -10 \rangle\]

\[\langle -10 \times -10 \rangle\]

Evaluating User-Defined Functions Calls (VIII)

And return the finished value...

\[
\lambda x, y: \langle x \times y \rangle \\
\lambda x: \langle \text{print } x \rangle
\]

\[\lambda x: \text{mul}(x, x)\]

\[\langle -10 \rangle\]

\[100\]

Evaluating User-Defined Functions Calls (IX)

...replacing the call to the user-defined function and yielding the final value:

\[
\lambda x, y: \langle x \times y \rangle \\
\lambda x: \langle \text{print } x \rangle
\]

\[\lambda x: \text{mul}(x, x)\]

\[\langle -10 \rangle\]

\[100\]

\[100\]

Summary: Environments

- **Environments** map names to values.
- They consist of chains of environment frames.
- An environment is either a global frame or a first (local) frame chained to a parent environment (which is itself either a global frame or ...).
- We say that a name is **bound** to a value in a frame.
- The value (or meaning) of a name in an environment is the value it is bound to in the first frame, if there is one, ...
- ...or if not, the meaning of the name in the parent environment

A Sample Environment Chain

<table>
<thead>
<tr>
<th>In</th>
<th>Value of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>1</td>
</tr>
<tr>
<td>Environ 1</td>
<td>2</td>
</tr>
<tr>
<td>Environ 2</td>
<td>3</td>
</tr>
</tbody>
</table>

- Environ 1’s parent
- Environ 1’s first frame
- Environ 2’s parent
- Environ 2’s first frame
Environments: Binding and Evaluation

- Every expression and statement is evaluated (executed) in an environment, which determines the meaning of its names.
- Subexpressions (pieces) of an expression are evaluated in the same environment as the expression.
- Assigning to a variable binds a value to it in (for now) the first frame of the environment in which the assignment is executed.
- Def statements bind a name to a function value in the first frame of the environment in which the def statement is executed.
- Calling a user-defined function creates a new local environment and binds the operand values in the call to the parameter names in that environment.

Example: Evaluation of a Call: `sum_square(3, 4)`

```
square: ...
mul, abs...

sum_square: \(x, y: \text{return square}(x)+\text{square}(y)\)

A
x: 3
y: 4

B
x: 3
y: 4

A
\text{square}(3)
\text{x*x}

B
\text{square}(4)
\text{x*x}
```

```
A
\text{sum_square}(3, 4)
\text{\text{x, y: return square}(x)+square}(y)\)

A
\text{square}(x)+\text{square}(y)

A
\text{square}(x)
\text{x*x}

B
\text{square}(y)
\text{x*x}
```

Last modified: Mon Mar 3 01:54:56 2014