Multiple Environments

Life Cycle of a User-Defined Function

Def statement:
Name: \texttt{square}(x):
return \texttt{mul}(x, x)

Calling/Applying:
def \texttt{square}(2+2)
A new function is created!
Name bound to that function in the current frame

What happens?
A new frame is created!
Parameters bound to arguments
Body is executed in that new environment

Operator & operands evaluated
Function (value of operator) called on arguments (values of operands)

A new frame is created!
Parameters bound to arguments
Body is executed in that new environment

Multiple Environments in One Diagram!
Multiple Environments in One Diagram!

An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Interactive Diagram

Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Interactive Diagram

Names Have Different Meanings in Different Environments

A call expression and the body of the function being called are evaluated in different environments.

Interactive Diagram

Environments for Higher-Order Functions

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Interactive Diagram
Environments Enable Higher-Order Functions

Functions are first-class: Functions are values in our programming language.

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

Environment diagrams describe how higher-order functions work!

Names can be Bound to Functional Arguments

Environment Diagrams for Nested Def Statements

Environments for Nested Definitions
How to Draw an Environment Diagram

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

\[ f_1: \text{make_adder} \quad \text{func adder(k) [parent=f_1]} \]

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:  \[ \text{[parent=<label>] } \]
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

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

- 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.
Lambda Expressions

>>> x = 10
>>> square = x * x

Also an expression: evaluates to a function

>>> square = lambda x: x * x

A function with formal parameter x
that returns the value of \(x^2\)

>>> square(4)
16

Lambda expressions are not common in Python, but important in general
Lambda expressions in Python cannot contain statements at all!

Lambda Expressions Versus Def Statements

\[
\text{square} = \lambda x: x^2 \quad \text{VS} \quad \text{def square}(x):
\]
\[
\quad \text{return } x^2
\]

• Both create a function with the same domain, range, and behavior.
• Both bind that function to the name square.
• Only the def statement gives the function an intrinsic name, which shows up in
environment diagrams but doesn’t affect execution (unless the function is printed).

Self-Reference

Returning a Function Using Its Own Name

http://pythontutor.com/composingprograms.html#code=def%20print_all(k):
  \\0A  \\0A  print(k)
  \\0A  return_all(1)(3)(5)

http://pythontutor.com/composingprograms.html#code=def%20print_sums(n):
  \\0A  print(n)
  \\0A  def%20next_sum(k):
  \\0A  \\0A  return%20print_sums(n+k)
  \\0A  return%20next_sum
  \\0A  \\0A  print_sums(1)(3)(5)
Review

What Would Python Display?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>print(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>delay(delay)()(6)</td>
<td>None</td>
<td>delayed</td>
</tr>
<tr>
<td>print(delays)(){}()</td>
<td>None</td>
<td>delayed</td>
</tr>
<tr>
<td>delay(print)()(4)</td>
<td>None</td>
<td>delayed</td>
</tr>
</tbody>
</table>

Names in nested def statements can refer to their enclosing scope.

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(square(1), 1)</td>
<td>17</td>
<td>Matey 17</td>
</tr>
<tr>
<td>pirate(pirate(pirate))()()()</td>
<td>Error</td>
<td>Matey Error</td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

```
from operator import add, mul

def square(x):
    return mul(x, x)

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg

    return plunder

pirate(pirate(pirate))(5)(7)
```

```
add(pirate(3)(square)(4), 1)
```

```
pirate(pirate(pirate))(5)(7)
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```