Announcements

• Lab 1 is due Wednesday 9/3 at 11:59pm

• Submitting labs and attending section may help your grade

• Homework 1 is due next Wednesday 9/10 at 11:59pm

• Office hours are a great place to ask questions about lab and homework assignments (demo)

• You can switch to sections with open space. http://goo.gl/nWfv7Z

• Michelle Hwang's sections (15, 18) are for students with little prior CS experience

• Videos are a mix of Fall 2013 and new content
Names, Assignment, and User-Defined Functions

(Demo)
Types of Expressions

**Primitive expressions:**

- 2
- `add`
- 'hello'

**Operator**  
**Operand**  
**Operand**  
Number or Numeral  
Name  
String

**Call expressions:**

- `max`  
  - (  
    - 2  
    - ,  
    - 3  
  )

An operand can also be a call expression

- `max(min(pow(3, 5), -4), min(1, -2))`
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)

???
```
Environment Diagrams
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):

Statements and expressions

Arrows indicate evaluation order

Frames (right):

Each name is bound to a value

Within a frame, a name cannot be repeated

(Demo)

Interactive Diagram
Assignment Statements

<table>
<thead>
<tr>
<th>Line</th>
<th>Expression</th>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a = 1</td>
<td>a 1</td>
</tr>
<tr>
<td>2</td>
<td>b = 2</td>
<td>b 2</td>
</tr>
<tr>
<td>3</td>
<td>b, a = a + b, b</td>
<td>a 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b 3</td>
</tr>
</tbody>
</table>

Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to the resulting values in the current frame.
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min, max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

`func min(...)`

`func max(...)`

(Demo)

`Global frame`

- \( f \)
- \( h \)
- \( g \)
- \( \text{max} \)

Interactive Diagram
Defining Functions
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Execution procedure for def statements:

1. Create a function with signature `<name>`(<formal parameters>)
2. Set the body of that function to be everything indented after the first line
3. Bind `<name>` to that function in the current frame
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```python
from operator import mul

def square(x):
    return mul(x, x)
square(-2)
```

A function’s signature has all the information needed to create a local frame

Interactive Diagram
Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:
• The global frame alone, or
• A local frame, followed by the global frame.

**Most important two things I’ll say all day:**

An environment is a sequence of frames.

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

E.g., to look up some name in the body of the square function:
• Look for that name in the local frame.
• If not found, look for it in the global frame.
  (Built-in names like “max” are in the global frame too, but we don’t draw them in environment diagrams.)

(Demo)
Print and None

(Demo)
None Indicates that Nothing is Returned

The special value `None` represents nothing in Python

A function that does not explicitly return a value will return `None`

*Careful*: `None` is *not displayed* by the interpreter as the value of an expression

```python
>>> def does_not_square(x):
...     x * x
... 
>>> does_not_square(4)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

The name `sixteen` is now bound to the value `None`
Pure Functions & Non-Pure Functions

**Pure Functions**
*just return values*

-2 \[\rightarrow\] abs \[\rightarrow\] 2

**Argument**

2, 100 \[\rightarrow\] pow \[\rightarrow\] 1267650600228229401496703205376

**Return value**

**Non-Pure Functions**
*have side effects*

-2 \[\rightarrow\] print \[\rightarrow\] None

**Returns None!**

*Python displays the output “-2”*

A side effect isn't a value; it's anything that happens as a consequence of calling a function.
Nested Expressions with Print

```python
None, None ➔

print(...):

None ➔

Does not get displayed

display “None None”

func print(...)

print(print(1), print(2)) ➔

None ➔

1
2
None None

1 ➔

print(...):

None ➔

display “1”

18

2 ➔

print(...):

None ➔

display “2”

None

>>> print(print(1), print(2))
1
2
None None
```