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

61A Lecture 2
Wednesday, September 3, 2014

Names, Assignment, and User-Defined Functions

(Demo)

Types of Expressions

Primitive expressions:

Number or Numeral
Name
String

Call expressions:

max | 2 , 3 |
Operator          Operand          Operand

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 visualize the interpreter’s process.

Code (left): Statements and expressions Arrows indicate evaluation order

Frames (right): Each name is bound to a value

Global frame

Just executed

Global frame

Just executed

Assignment Statements

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.
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.

Function signature indicates how many arguments a function takes.

Function body defines the computation performed when the function is applied.

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.

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.)

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.

>>> def does_not_square(x):
...     x * x
...     return None
...     return x
...     x * x

>>> does_not_square(4)
None

The name sixteen is in bound to the value None.

>>> sixteen = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
Pure Functions & Non-Pure Functions

Pure Functions
- just return values

Non-Pure Functions
- have side effects

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

None, None  
display "None None"

None  
print(print(1), print(2))

func print(...):

1  
print(1)

2  
print(2)

17  
Does not get displayed

18  
func print(1)

1  
func print(...)

display "3"

19  
func print(2)

2  
func print(...)

display "2"