Announcements

- Starting next week, submitting labs & attending section will provide a midterm safety net.

- Homework 1 is due next Wednesday 1/28
  - All homework is graded on effort; you must make progress on each problem to earn 2/2
  - Homework Party on Tuesday 1/27 5-6:30pm in 2050 VLSB

- Quiz 1 released next Wednesday 1/28 is due next Thursday 1/29 (graded on correctness)

- Ask questions about lab and homework assignments in office hours! [cs61a.org/weekly.html]
  - 2 locations in Bechtel Engineering Center (Map: http://goo.gl/dAcHXf)
  - 11-2 & 3-5 on Monday, 11-6 on Tuesday & Thursday, 11-2 & 3-4 on Wednesday, 11-1 on Friday

- You need to register a class account (Lab 0); that’s how we track assignments
  - Please register even if you’re on the waitlist or applying for concurrent enrollment

---

Names, Assignment, and User-Defined Functions

Types of Expressions

- Primitive expressions:
  - Number or Numeral
  - Name
  - String

- Call expressions:
  - Operator
  - Operand
  - Operand
  - An operand can also be a call expression

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.
- 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 those resulting values in the current frame.
Discussion Question 1 Solution

```
def min(**kwargs):
    max_val = None
    for key, value in kwargs.items():
        if max_val is None or value < max_val:
            max_val = value
    return max_val

f(2, g(h(1, 5), 3))
```

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.

```
def (formal parameters):
    return expression
```

Calling User-Defined Functions

```
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.

```
from operator import mul

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

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):
...    return None
...    x * x
```

Print and None

```
4
```
Pure Functions & Non-Pure Functions

Pure Functions
Just return values

-2 \rightarrow \text{abs} \rightarrow 2

Return value

2, 100 \rightarrow \text{pow} \rightarrow 126750080228220481496783205376

2 Arguments

Non-Pure Functions
Have side effects

-2 \rightarrow \text{print} \rightarrow \text{None}

Returns None!

A side effect isn’t a value; it’s anything that happens as a consequence of calling a function.

Python displays the output “-2”

Nested Expressions with Print

None, None \rightarrow \text{print(1)} \rightarrow \text{None} \rightarrow \text{None}

Displays “1”

None, None \rightarrow \text{print(2)} \rightarrow \text{None} \rightarrow \text{None}

Displays “2”

None, None \rightarrow \text{print(print(1), print(2))} \rightarrow \text{None} \rightarrow \text{None}

Displays “None None”

17

2 Arguments

Nested Expressions with Print

None, None \rightarrow \text{print(1)} \rightarrow \text{None} \rightarrow \text{None}

Displays “1”

None, None \rightarrow \text{print(2)} \rightarrow \text{None} \rightarrow \text{None}

Displays “2”

None, None \rightarrow \text{print(1), print(2)} \rightarrow \text{None} \rightarrow \text{None}

Displays “None None”