61A LECTURE 2 – NAMES, ENVIRONMENTS

Steven Tang and Eric Tzeng
June 25, 2013

Announcements

- Homework 1 is released!
  - Due Thursday at 11:59pm
  - Feel free to ask questions about the Python problems on Piazza
- Project 1 will be released today!
  - Due 7/3 at 11:59pm
  - Start looking for a partner…
- Office hours start today
  - Schedule on the website
  - Mine are right after this (9:30-10:30 AM)

Clarification on grading

- Labs and discussions are *not* graded
  - …but you really should go!
- The only things worth points are homeworks, projects, and exams (plus a few extra points here and there…)

The Course Staff - Lecturers

Steven Tang

Graduated L&S CS from Cal

Back for a PhD in Education

Eric Tzeng

Graduated EECS from Cal

Back for a PhD in Computer Science

Some applications...

Phones
Cars
Politics
Games
Education
Movies
Music
Sports
Anything connected to the Internet
…

Systems
Programming Languages
Graphics
Artificial Intelligence
Databases
Theory
Security
Parallel Computing
Quantum Computing

A few more acknowledgements…

- Thanks to Tom Magrino and Jon Kotker, for their advice and sage wisdom in preparing this course
- Thanks to Brian Harvey, without whom 61A wouldn't be what it is today!

Whew!

- On to Python and actual computer science now!
- Warning: this lecture is quite a bit more dense than the previous one!

The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language
- Means of Combination
  - Compound elements built from simpler ones
- Means of Abstraction
  - Elements can be named and manipulated as units

Today!

The key to abstraction

- **Names!**
- Names allow us to quickly reuse:
  - Data
  - Rules for manipulating that data (functions)
- Quick demo in Python

A disclaimer

- This lecture, I'm going to go over a lot of naming models that are flat out WRONG.
- Remember them, so that you don't make the same mistakes!

And now, a mystery…

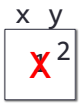
```
>>> x = 1
>>> y = x
>>> x = 2
>>> y
???
```

Variables as containers

- One way people sometimes think about variables is to think of them as containers
  - A variable "holds" a value, and when you assign to a variable, you're changing the value it "holds"

Variables as containers cont.

```
>>> x = 1
>>> y = x
>>> x = 2
>>> y
1    # wait, what?!
```

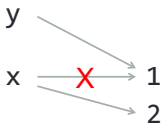


WRONG WRONG WRONG

Variables as references

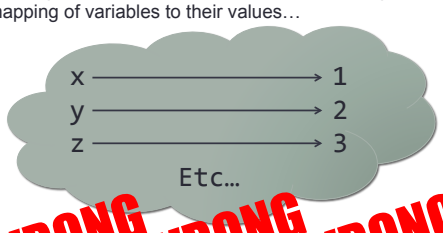
- The correct way to model this is to treat variables as references to values
- Some ground rules…
  - Assigning a variable changes the reference, never the value!
  - Variables "point to" values, never references!

```
>>> x = 1
>>> y = x
>>> x = 2
>>> y
1
```



So how do variables work?

- You might be tempted to think that there's a single mapping of variables to their values…



WRONG WRONG WRONG

Functions

- We already know how to give names to data
- Now let's give names to ways of manipulating that data!
- Done using a `def` statement (note: not an expression!)

Function name | Parameter 0 | Parameter 1

```
def convert_to_cents(dollars, cents):
    return dollars * 100 + cents
```

Return statement

Consider this…

```
>>> x = 3
>>> def f(x):
...     return x
...
>>> f(2)
2
>>> x
3
```

This x is 3…

But this x is 2!

What have we learned so far?

- Names are hard.
- Also, variables are references!
- Also, names are hard.

Break!

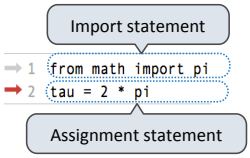
- When we come back, we discuss the solution to all of our naming woes!

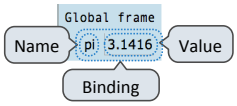
Environment diagrams

Diagram from NASA

Environment diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Next line is highlighted

Frames (right):

A name is bound to a value

In a frame, there is at most one binding per name

Back to user-defined functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function "signature" indicates how many parameters

>>> def <name>(<formal parameters>):
return <return expression>

Function "body" defines a computational process

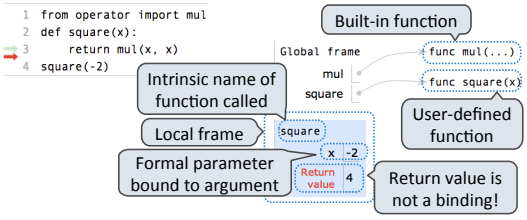
Execution procedure for def statements:

1. Create a function value with signature <name>(<formal parameters>)
2. Bind <name> to that value in the current frame

Calling user-defined functions

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

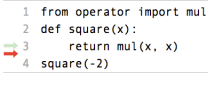
1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

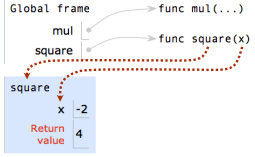


Calling user-defined functions

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

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



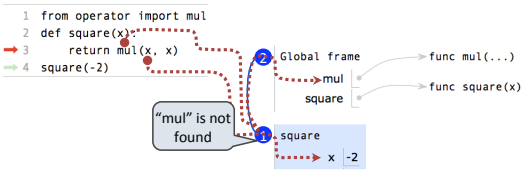


A function's signature has all the information to create a local frame

Looking up names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error



What's the point?

- 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
- **Important properties of environments:**
  - An environment is a sequence of frames
  - The earliest frame that contains a binding for a name determines the value that the name evaluates to
- The *scope* of a name is the region of code that has access to it

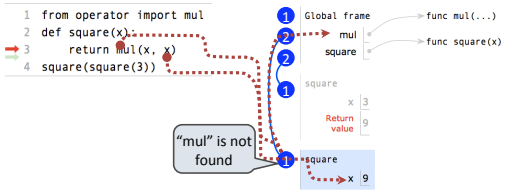
Break!

Multiple environments in one diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

mul(x, x)

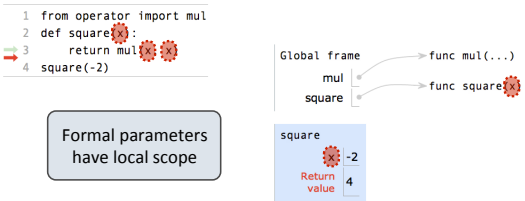


Formal parameters

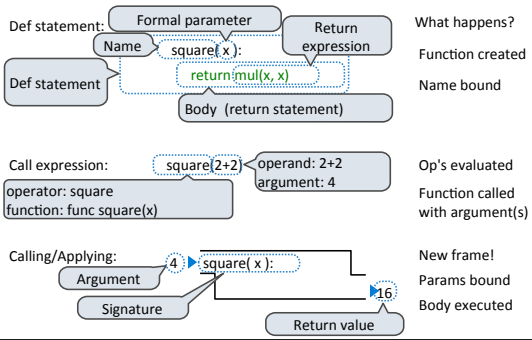
def square(x):
  return mul(x, x)

vs

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



Life cycle of a user-defined function



Closing remarks

- That was a lot to take in at once!
- It's okay if you're feeling a little overwhelmed right now
  - But practice makes perfect…
  - Draw these a lot (you'll get a chance in discussion today)
- Follow the rules, and you'll be okay
- We're going to make things a little more complicated in a couple of days, so make sure you get it ASAP!