Lecture 5: Environments

June 27, 2022

Richard Roggenkemper

Announcements

- Lectures have a Piazza thread
  - This can be used live or later when watching the recording
- Office Hours are starting this week
  - cs61a.org/office-hours
- Tutoring is also starting this week!
  - tutorials.cs61a.org
- Pace of this week will be similar to what you see in the future
- Lab01, HW01, and Hog (Project #1) are all out!

Programming Assignment FAQs

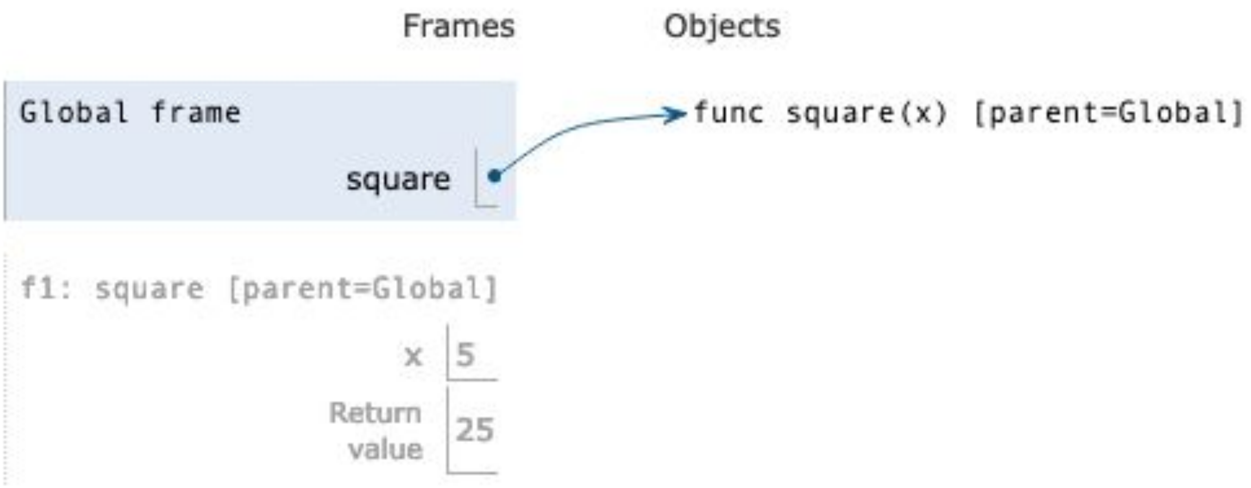
- Which assignments are programming assignments?
  - Labs, Homeworks, and Projects
- How many times can we test our code?
  - As many times as you want! You can keep running the autograder locally with `python3 ok`
- Are there any hidden tests?
  - No, unless explicitly noted
  - For most, maybe all programming assignments this semester, there will be no hidden tests
- When will I get my grade for these?
  - You can run `python3 ok --score` to see if you missed anything
  - We will release grades with an email announcement from Piazza some time after the assignment is due
- When will howamidoing.cs61a.org be updated?
  - After we release the first batch of grades

Environment Diagrams

What are Environment Diagrams?

A visual tool to keep track of bindings & state of a computer program

In this class, we use Python as our primary language, but environment diagrams can be used in many different programming languages



Why do we use Environment Diagrams?

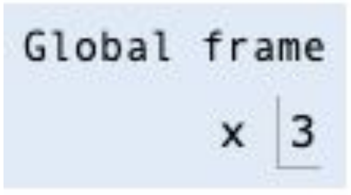
- Environment Diagrams are conceptual
  - understand *why* programs work the way they do
  - confidently predict how a program will behave

- Environment Diagrams are helpful for debugging
  - When you're really stuck,

What do we've seen so far

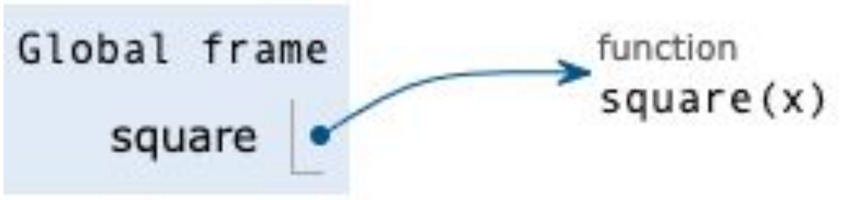
Assignment Statements

```
x = 1
x = x + x + x
```



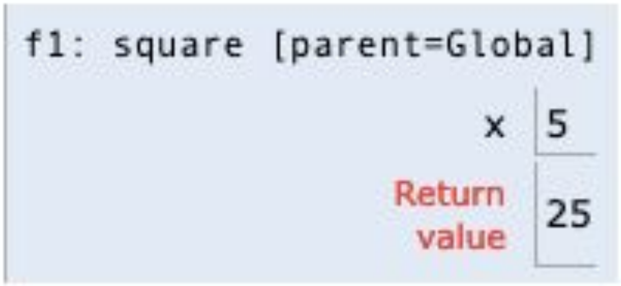
Def Statements

```
def square(x):
    return x * x
```



Call Expressions

```
square(4)
```



Frames

Demo

- A **frame** keeps track of variable-to-value bindings
  - Every call expression has a corresponding frame
- The **global frame**, is the starting frame
  - It doesn't correspond to a specific call expression
- **Parent frames**
  - The parent of a function is the frame in which it was defined
  - If you can't find a variable in the current frame, you check it's parent, and so on. If you can't find the variable, **NameError**

How to draw an Environment Diagram

When a function is defined:

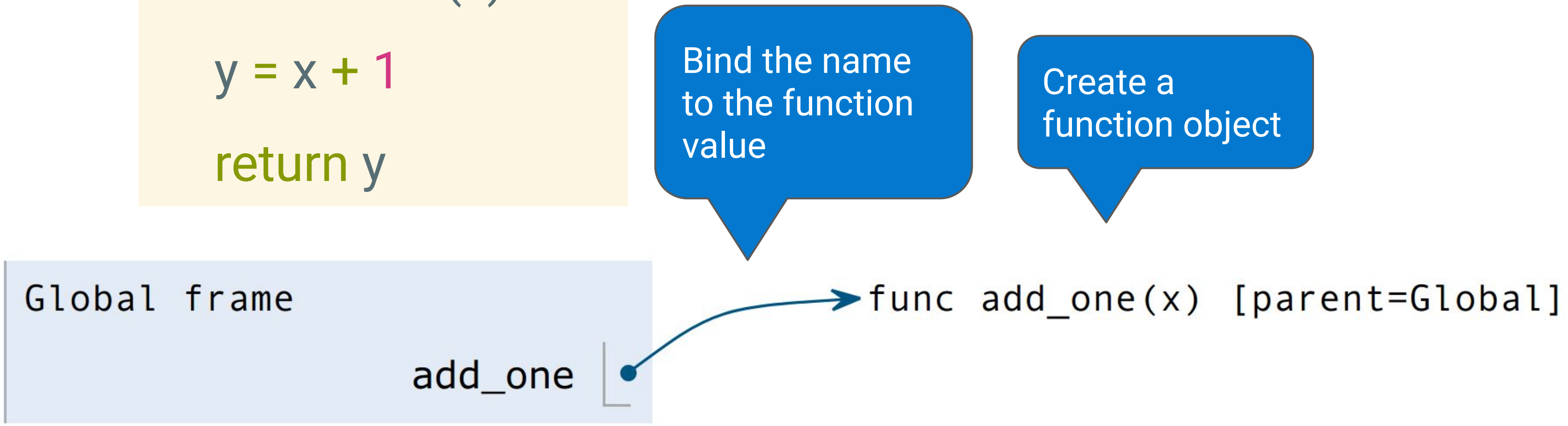
Create a function value:

Its parent is the current frame

func <name>(<formal parameters>) [parent=<frame>]

Bind <name> to the function value in the current frame

```
def add_one(x):
    y = x + 1
    return y
```



Bind the name to the function value

Create a function object

Global frame

add_one

func add_one(x) [parent=Global]

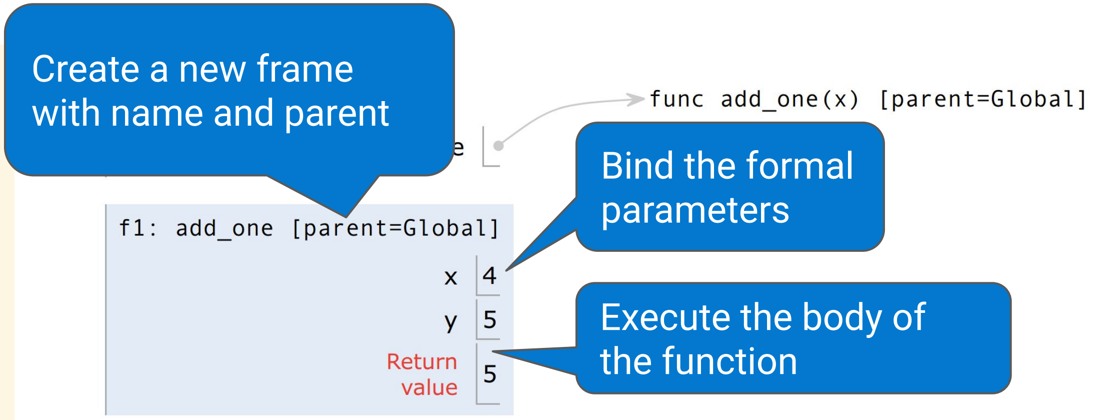
How to draw an Environment Diagram

Demo

When a function is applied:

1. Add a **local frame**, titled with the <name> of the function being applied.
2. Copy the parent of the **function** (not always the current frame) to the local frame: [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

```
def add_one(x):
    y = x + 1
    return y
add_one(4)
```



Create a new frame with name and parent

Bind the formal parameters

Execute the body of the function

Check Your Understanding

Demo

Draw the environment diagram

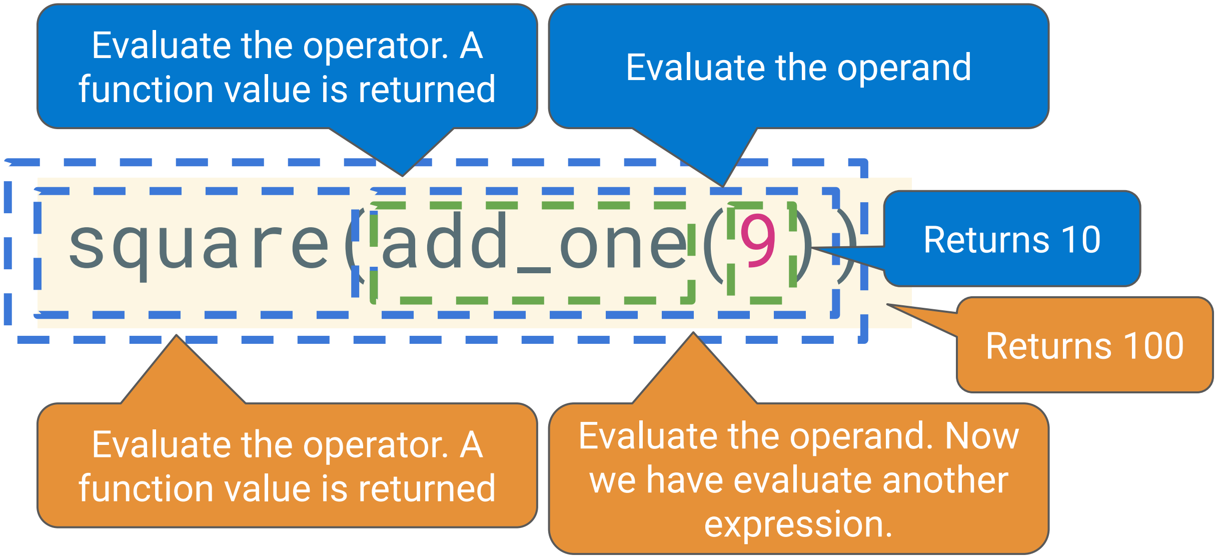
```
def square(x):
    return x * x

def sum_of_squares(x, y):
    return square(x) + square(y)

sum_of_squares(3, 4)
```

Demo

Evaluation Order

Remember to evaluate the **operator**, then the **operand(s)**, then apply the **operator** onto the **operand(s)**.

```
def add_one(x):
    y = x + 1
    return y

def square(x):
    return x * x
```



Evaluate the operator. A function value is returned

Evaluate the operand

```
square(add_one(9))
```

Returns 10

Returns 100

Evaluate the operator. A function value is returned

Evaluate the operand. Now we have evaluate another expression.

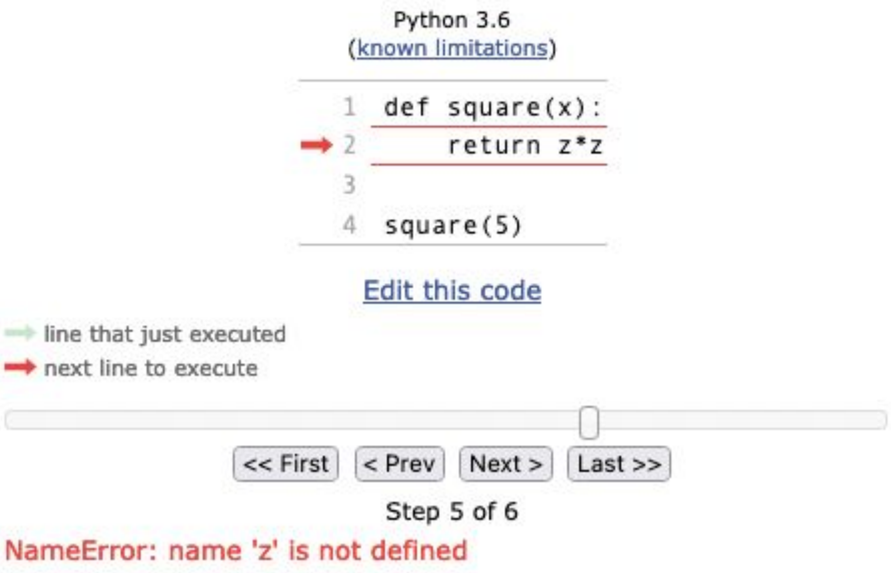
The environment diagram should reflect Python’s evaluation.

Variable Lookup

Demo

Variable Lookup

1. Lookup name in the current frame
2. Lookup name in parent frame, its parent frame, etc..
3. Stop at the global frame
4. If not found, an **NameError** is thrown



```
def square(x):
    return z*z

square(5)
```

```
NameError: name 'z' is not defined
```

Break

Lambda Expressions

Lambda Expressions

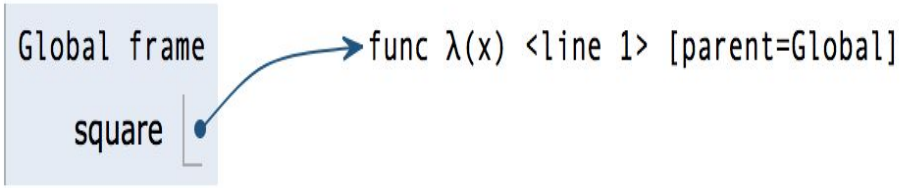
Expressions that evaluate to simple functions

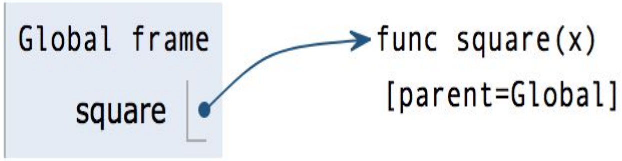
```
>>> square = lambda x: x * x

>>> square
<function <lambda> ... >
>>> square(4)
16
>>> x = square(5)
>>> x
25
```

Lambda Expressions vs def Statements

```
square = lambda x: x * x
```

```
def square(x):
    return x * x
```





- Both create a function with the same behavior
- The parent frame of each function is the frame in which they were defined
- Both bind the function to the same name
- Only the `def` statement gives the function an intrinsic name

Demo

Check Your Understanding

```
times = 2

def repeated(f, n, x):
    while n > 0:
        x = f(x)
        n -= 1
    return x

def square(x):
    return x * x

repeated(lambda x: x*x, times, 3)

repeated(square, times, 3)
```

Higher Order Functions

Higher Order Functions

A function that ...

- takes a function as an argument value or
- returns a function as a return value

```python
times = 2

def repeated(f, n, x):
    while n > 0:
        x = f(x)
        n -= 1
    return x

repeated(lambda x: x*x, times, 3)
```

Demo

Function Currying

What is **currying** ?

- Converting a function that takes multiple arguments into a single-argument higher-order function

Here is an example of a function that currys a two-argument function

```
def curry2(f):
    def g(x):
        def h(y):
            return f(x,y)
        return h
    return g
```

Summary

- **Environment Diagrams** formalize the evaluation procedure for Python
    - Understanding them will help you think deeply about how the code that you are writing actually works
- **Lambda** functions are similar to functions defined with `def`, but are nameless
- A **Higher Order Function** is a function that either takes in functions as an argument and/or returns a function as a return value