ENVIRONMENT DIAGRAMS

COMPUTER SCIENCE 61A

September 10, 2012

1 Drawing the Diagram

1.1 Background

Let's start with some definitions:

- **frame**: location where variable bindings are stored
- **binding**: connection between a name (variable) and a value (int, function, etc.)
- keep track of the **current frame** – the frame in which code is currently being executed (and where to start variable lookup).
- The **frame number** is a unique identifier we use to label frames (for example, "E1"). **NOTE**: some frames will not have numbers – only give a label to a frame if you need to refer to it later (e.g. with nested functions).

1.2 Starting Your Diagram

- First draw the global frame. Designate that it is the global frame by writing "global frame" in the upper-left corner. Note that the *current frame* is now this global frame.

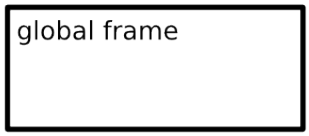

Figure 1: The global frame

- Step thorough the code evaluating one line at a time.

1.3 Binding values to names

This occurs for multiple types of statements:

Statement	Example	Name	Value
"=" sign	`x = 6`	x	6
"def"	`def square(x):` `  return x*x`	square	func square(x)
"import"	`from operator import add`	add	func add(...)

Here's what to do if you see one of the above:

1. **Evaluate the value**:
   - A complicated statement like `y = square(add(4, 5))` will require making function calls (see below), but will eventually evaluate to 81.
   - Function values are written like this: `func name(params)`. For example, `func square(x)` and `func add(...)` are function objects.
2. **Write the variable name**:
   - For "="sign statements, this is the name on the left-hand side of the "=".
   - For "def" statements, this is the name of the function. For example, "square" is the name of the function.
3. **Associate the name with the value**:
   - For primitives (like numbers and booleans), just put the value right next to where you wrote the name of the variable
   - For other objects (like functions), draw an arrow from the name to the function object (which should be floating nearby outside of the frame).

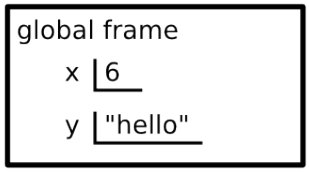

Figure 2: The result of evaluating `x, y = 6, "hello"` in the global frame

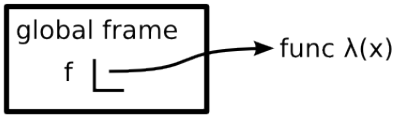

Figure 3: The result of evaluating `f = lambda x: x*x` in the global frame

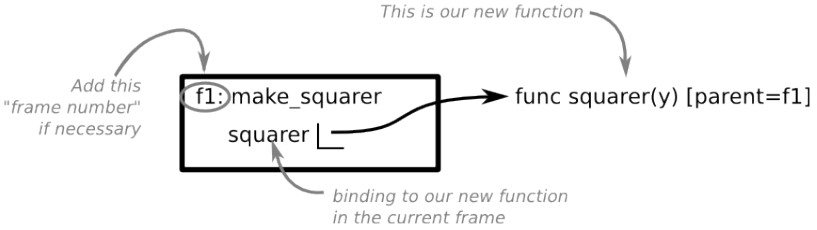

Figure 4: The result of evaluating a function definition for a function named squarer from within a frame named `make_squarer`.

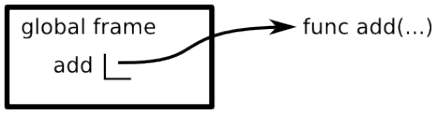

Figure 5: The result of evaluating "`from operator import add`" in the global frame.

Note that the argument list for built in functions is usually represented as "`...`" even if the function only accepts a finite number of arguments.

1.4 Calling User-defined Functions

You can tell when a function is being called when you see a variable name being used with parentheses (for example, "square(3)").

1. **Draw new frame**:

   This new frame is called a "local frame". Label the top left corner with the name of the function. If the function has a parent frame, write the parent frame's label in the top right corner, like this:

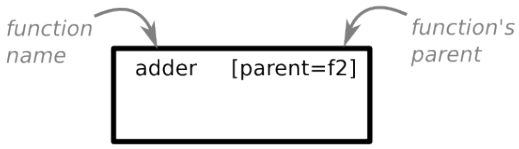

Figure 6: The frame resulting from calling a function whose name is `adder` parent is `f2` i.e. a function whose representation in the environment diagram is something like `func adder(y) [parent=f2]`. Note that in this diagram, the parameters are not yet bound

2. **Bind formal parameters**:

   First, evaluate the operands from left to right (this might require drawing even more frames!). Then, assign the computed values to the formal parameters (see "Binding names to values")

3. **Execute the function body**:

Starting at the first line of the function (after the def statement), execute each line.

4. **Write the return value**:

   Write the value that the function returns at the bottom of the frame. **NOTE**: if the function has no "return" statement, the return value is "None".

2 Reading the Diagram

2.1 Looking up a name:

1. Start in the current frame.
2. If you can't find the name in the current frame, look in its parent (the parent's frame number is written in the top-right corner – if not, then the parent is "global").
3. If still can't find the name in the parent frame, look in the parent's parent frame (i.e. the grandparent frame?). Continue until:
   (a) you find the name: then use the value associated with the frame
   (b) there are no more parents (by now you should be in "global"): Python throws a NameError!

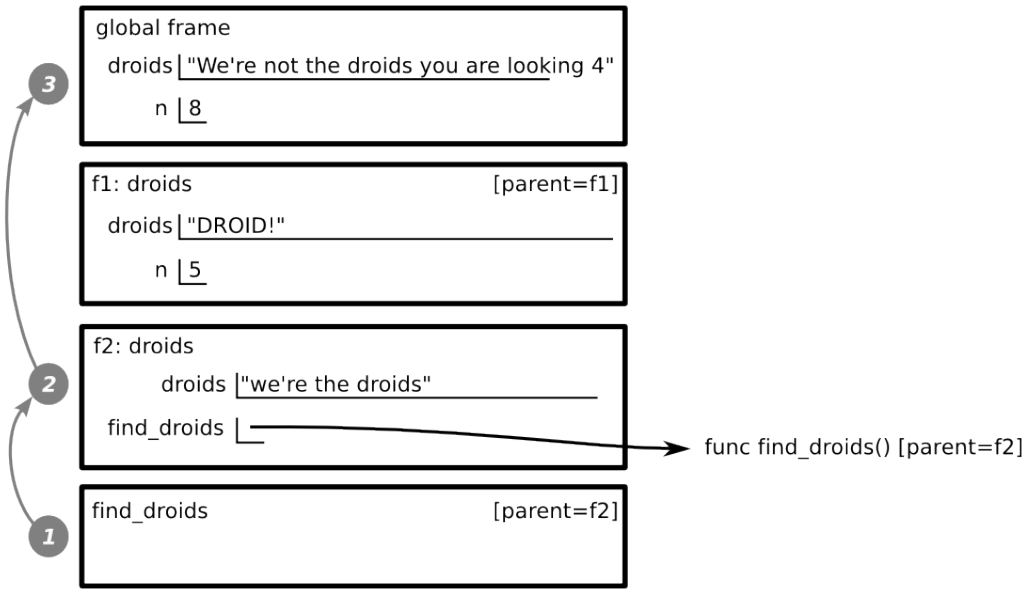

Figure 7: Note that in this diagram the result of looking up `droids` in the `find_droids` frame is `we're the droids`, and the result of looking up `n` in the same frame is `8`. The lookup order passes from parent to child moving from the bottom frame to f2 to the global frame.

2.2 Dos and Don'ts

- Variables should NOT point to other variables!
- Do NOT draw frames for built-in functions, like `sum(...)` or **`print`**`(...)`.
- Do NOT execute the body of a function when you are just defining the function.
- Remember to follow the sequence of frames from parent to child—not simply from bottom to top.
- Remember to write down the parent of every function or frame whenever this parent is not the global frame.