Lab 7: Memory

CS 1109, 2023SU

Zachary J. Susag

July 14, 2023

These exercises are designed to give you practice *reading* and *tracing* Python code. For each program below, draw the memory diagram (global space and call frames) that corresponds to running the program.

There is nothing to submit for this lab. Instead, **remember that the in-person exam takes place next Friday, July 21st**. This exercise is excellent practice for the exam, and there is a high probability that a similar type of question will appear on it. If you have questions, please ask a course staff member or one of your fellow classmates!

Drawing Memory Diagrams

Use the following set of steps to draw your memory diagrams.

- A variable is either in the global space (*global variable*) or in a call frame (*local variable*)
- **Assigning to a *new* variable:**
  1. Write the name of the variable in the appropriate place in memory (i.e., in the global space if it is a global variable or in a call frame if it is a local variable).
  2. Draw a box next to the name.
  3. Inside the box write the value that the variable stores.
- **Assigning to an *existing* variable:**
  1. Find the variable name in the appropriate place in memory. Remember that global variables can only be written to in the *global space.*
  2. Cross out the old value with a single slash. The crossed out value should still be visible.
  3. Write the new value next to the old, crossed out value.
- Draw a call frame **for each function call**. When a function ends, cross out the call frame.
- A call frame should have the following components:
  1. The **function name** in the top-left corner.
  2. The **line number that the function is currently executing** in the top-right corner.
  3. All **parameters** and **local variables**.
  4. If a function has a `return` statement, draw a **RETURN** variable to store the return value.
- Draw call frames in a **stack** growing downward.

Exercise 1

```python
def foo(s):
  return s / 2

def bar(s):
  g = foo(s)
  t = s
  s = g
  g = t
  return g

def baz():
  g = bar(1)
  foo(g)
  return g

s = 4
x = foo(s)
baz()
```

Exercise 2

```
def f(x):
  return g(x) // 2

def g(s):
  x = h(s)
  x = 2 * x
  return x

def h(z):
  z = z + 2
  return z % 3

x = 1
y = f(x + 1)
```