CS 1110 Spring 2022, Assignment 2: Eathiopia (Object and Call Frame Diagrams)*

1 Estimate of Time Required

We planned this assignment to take only 2-5 hours, including doing some worked examples beforehand, so that completing this assignment and submitting revisions of A1 during overlapping time periods is definitely doable.

2 Availability of Worked Examples

We strongly recommend that you go through worked examples of diagramming variables, objects, and call frames out beforehand, and will be very happy to go over these with you at consulting/office hours[1]. Besides lectures slides, see prior years' A2s and solutions in the "Archive" section of our assignment advice and archive page[2].

Contents

3 New Rules

3.1 One-shot Submission From Now On

There is no revise-and-resubmit for this or any subsequent assignment unless otherwise noted.

*Authors: Lillian Lee. Indebted to previous iterations by Anne Bracy, Lillian Lee, Steve Marschner, Stephen McDowell, Walker White, and David Gries.

[1] http://www.cs.cornell.edu/courses/cs1110/2022sp/staff

[2] https://www.cs.cornell.edu/courses/cs1110/2022sp/resources/doing-assignments.html

3.2 Groups Not Preserved Across Assignments; Re-Grouping Needed

You may work alone or with just one other person, who can be someone you've grouped with before in CS1110, or a different person. If you are partnering, see the directions for how this works in Gradescope in Section 8.

3.3 Submit on Gradescope, not CMS; Different Partnering Protocol

To facilitate grading for this visually-oriented assignment, we will use Gradescope[3] instead of CMS. This does mean that partnering on the submission server is done differently; read Timeline and Deadlines carefully for instructions.

4 Rules From Before That Still Apply

Sections 1.2-1.3 of Assignment 1[4].

5 Learning Objectives (and Mistakes to Avoid)

You will practice executing Python code on "paper" using the notation we have introduced in class. This notation constitutes a precise visual language for describing and understanding exactly what Python is doing when it executes code. In complex coding situations, we use these kinds of diagrams ourselves to figure out what's going on.

Concepts tested:

1. What statements create variables or change the values of which variables (including global variables, local variables, and object attributes).
2. That the result of a constructor expression is the ID of the new object that is created.
3. That frames summarize the state of the process of executing a function call. The variables they contain store information local to the corresponding function, and indicate what that function can affect; the program counter records what line number should be executed next.
    - *Parameters* are local variables whose purpose is to hold the input values that the function is supplied when called.
    - *Arguments* are the input values that are supplied to a function when the function is called, and are stored in the corresponding parameter variables.
4. Which statements of an if-(elif-else) block are executed.

Given these learning objectives, **some seemingly minor but actually tragic mistakes you can expect to lose points for committing** are:

I. Drawing fewer or more objects than the number of constructor expressions that are evaluated during execution.

II. In the frame for a call to a function, not drawing a correspondingly named box for each parameter in that function's header.

III. Drawing non-existent variables (indicating that you believe in their existence).

IV. Writing the name of a variable, say `x`, inside of the box for another variable, say a box named `y`, instead of a value [the problem: if `x`'s value is subsequently changed, you would predict the wrong value for `y`].

V. Writing a variable name on the tab of an object instead of a value [the problem: if the variable's value changes, you would incorrectly predict that the object's ID changes too].

Some **seemingly minor notational mistakes that *could* indicate deeper misunderstandings and thus risk point deductions** are:

A. Not having the correct sequence of crossed-out line numbers in the frame's program counter [the problem: you might be misunderstanding where the flow of execution goes next].

[3] https://www.gradescope.com/courses/373248
[4] https://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment1/a1.pdf

6 Notational Conventions (Some Differ From Previous Semesters)

1. Do not try to "show animation" in your diagrams.[5] That is, there should be one call-frame on your paper for one function call, no matter how many individual lines of that function are executed. As another example: each global variable should show up exactly once, not multiple times, on your paper.

2. Do not erase any values, objects, or frames. For values that are changed, the old value should be neatly crossed out such that we can see what the old value was; and the new value should be written next to it. Similarly, frames should be crossed out rather than erased, and objects should never be removed.

3. In the program counter (the place in call frames where you put line numbers and cross them out), do not enter numbers for lines corresponding to comments or docstrings.

4. When function execution ends, cross out the final value of the program counter. The series of crossed-out program-counter values is a record of which lines were executed during the function call.

5. If a function call returns some value v, write "RETURN $\boxed{v}$" in the frame, e.g., "RETURN $\boxed{3}$" for a function call that returns the integer 3. (Be sure you are writing a value, not a variable name.) If a function call explicitly or implicitly returns `None`, write "RETURN $\boxed{\text{None}}$" in the frame.

6. Place your frames so that their position reflects the order in which they were created; we recommend starting at the top and drawing each subsequent frame below the previous one.

7. Don't draw the objects (folders) for imported modules or for function definitions.

8. Don't draw call frames for built-in functions, such as `int()`.

9. Since we haven't covered class definitions in detail yet, assume that the creation of a new object and initialization of its attributes happen in one step, and no call frame is generated.[6]

10. `else`-statements and `elif`-statements should be treated just like other lines, and so may or may not show up in the program counter.[7]

11. It is optional to draw parameter boxes as projecting out of the left side of a call frame (visually indicating that they receive their values from the "outside" caller). Similarly, it is optional to draw RETURN boxes as projecting out of the right side of a call frame (visually indicating that the RETURN value is given to the "outside" caller.) But if you adopt this "some variable boxes stick out of the frames" convention, *make sure you don't have local variables that aren't parameters sticking out.*

12. Don't write the line numbers of blank lines in the instruction/program counter of your call frames.

7 Your Task: Restaurant Eathiopia

7.1 Motivation

The situation modeled by this assignment's code is a restaurant where each table of customers is served communal dishes, one course at a time. Each course comes from one of the restaurant's pots. If there isn't enough servings in a specified pot for all the customers at a table, then the restaurant tries to serve the table from a designated *substitute* pot instead (leaving the first pot entirely alone). But if the second pot wouldn't have enough, we fall back to the second pot's substitute, and so on. At our restaurant, there is always the last resort of Injera bread, of which we assume we have an infinite supply.

The restaurant earns money from serving a particular dish to a table based on how many people were at the table and what the price per serving was for that dish.

[5] This is a significant difference between Fall CS1110 and Spring CS1110 call-frame notation.

[6] That is, for now, don't worry about the `__init__` method in a class or draw a frame for it, even though Python Tutor does.

[7] This may be a significant difference between Fall CS1110 and Spring CS1110 call-frame notation, or a deviation from what Python Tutor displays.

7.2 Instructions

In Section 9, and also in files a2_calls.py[8] and a2_objects.py[9] (both of which import from pot.py[10]), is the code to work with.

The definition for class Pot[11] means that a constructor expression like `Pot("Lentils", 100, 1.0, None)` creates a new `Pot` object with `dish` attribute having the value `"Lentils"`, `servings` attribute having the value `100`, `pps` attribute having the value `1.0`, and `sub` attribute having the value `None`.

The definition for class Table[12] means that a constructor expression like `Table(13)` creates a new `Table` object with `seated` attribute having the value `13` and `course` attribute having the value `""`.

Your submission should contain:

- A diagram (compliant with the notational conventions given in Section 6, lecture, and the worked examples referenced in Section 2) showing what happens during the execution of `a2_calls.py`.
- A diagram (compliant with the notational conventions given in Section 6, lecture, and the worked examples referenced in Section 2) showing what happens during the execution of `a2_objects.py`.
- Your answers to Q1-Q4, where these questions appear as comments in `a2_objects.py`.
  - The questions are contextual and consecutive, so for example, Q1 is asking what happens when lines 1-9 are executed, Q2 is asking what happens when lines 1-25 are executed, etc.

In your diagrams, **use the line numbers given in Section 9**, and **choose consecutive ids for the objects you create.** This is *different* than what Python Tutor does! (More on this below.)

You are welcome to draw your diagrams by hand (and will need to do so for exams), but we also provide a template for depicting folders for Powerpoint[13] and Google Slides[14].

7.3 Need Help? Try Python Tutor

You are allowed and encouraged to use the CS1110 version of Python Tutor[15]. But, first try the worked examples by hand, and attempt to do this assignment manually before checking with Python Tutor: One often learns more from making mistakes and being corrected than by just seeing someone else or some program solve a problem for us.

Be aware that some of Python Tutor's notational and display conventions differ from what we require on assignments and exams.

8 Turning in the Assignment (Plan Ahead To Make Time for Scanning to PDF)

We highly recommend that before submission, you double-check your answers against the "Learning Objectives (and Mistakes to Avoid)" section above.

Put the netids of *all* group members at the top of the page(s) you submit.

If you work on paper, there are scanners in Olin and/or Uris Library[16]. Other possibilities[17] and computing labs equipped with scanners can be found by selecting "Peripherals" in the sidebar and then "Scanners" here[18].[19] You can also scan your homework with a mobile device[20].

Your solution must be a single PDF file.[21] Your file must be less than 100MB in size, and ideally less than 10MB. This should not be a problem as long as the resolution is reasonable.[22]

[8] http://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment2/a2_calls.py
[9] http://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment2/a2_objects.py
[10] http://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment2/pot.py
[11] This is given in file `pot.py`, but you don't need to look.
[12] This is given in file `pot.py`, but you don't need to look.
[13] http://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment2/diagram_template.pptx
[14] https://docs.google.com/presentation/d/1n45XTEeqMIR1qAb778RvXMiwB5Co8aAuezrwcmEch58/edit?usp=sharing
[15] http://cs1110.cs.cornell.edu/tutor
[16] https://olinuris.library.cornell.edu/collections/maps/services
[17] https://www.library.cornell.edu/about/collections/visual-resources/digitization
[18] http://mapping.cit.cornell.edu/publiclabs/map/alternate_search_view.cfm
[19] Disclaimer: the CS1110 staff cannot ascertain for you or guarantee that a particular scanner is available or in working order.
[20] https://help.gradescope.com/article/0chl25eed3
[21] For pdf file merging, there is the program PDFtk for Windows (https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/) and the built-in application Preview for OS X.
[22] If your file is still too large, try SmallPDF (https://smallpdf.com/compress-pdf) to compress it.

Not Legible To Us? No Credit, Unfortunately A caution for those who plan to take a photo of your work and convert that to pdf: We unfortunately must reserve the right to judge a submission to be illegible and thus assign zero credit, and have had to do so in the past. *Please* make sure the pdf file you submit can be read by the graders.[23] You can also download your submission from Gradescope to see how it looks, and re-upload if necessary[24].

8.1 Timeline and Deadlines

1. If you are partnering: well before Thu Mar 3, decide which of you will make the submission for the 2pm checkpoint on Thu Mar 3. **This is important because on Gradescope, unlike CMS, partnerships are declared *after* first submission!**

2. By 2pm on Thu Mar 3, whoever is responsible for submitting — and only that person — makes the initial submission on Gradescope[25]. **If partnering,** then that person adds the partner to the submission; instructions at `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members`.

3. By 11:59pm Ithaca time on Thu Mar 3, make your final submission on Gradescope[26].

The 2pm checkpoint provides you a chance to alert us during business hours of any submission problems. **Since you've been warned to submit early, do not expect that we will accept work that isn't accepted by the submission server on time** for whatever reason, including server delays stemming from many other students trying to submit at the same time as you.[27]

Of course, if some special circumstances arise, contact the instructor(s) immediately.

(Code begins on next page. This big blank space is to prevent pagebreaks within the code.)

[23] Former TA Anthony Poon recommends Genius Scan (`https://www.thegrizzlylabs.com/genius-scan/`), which processes images into a flat, perspective-corrected, and evenly-lit PDF.

[24] `https://help.gradescope.com/article/axm932lptr-student-view-submission`

[25] `https://www.gradescope.com/courses/373248`

[26] `https://www.gradescope.com/courses/373248`

[27] There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy.

9 Code to diagram

9.1 a2_calls.py

You can assume that execution starts at line 46; that is, assume that the expression in line 45 evaluates to True.

```
1  # a2_calls.py
2  # Lillian Lee, Feb 21, 2022
3
4  from pot import Pot, Table # the Pot class
5
6
7  def serve(t, p):
8      """Serve the appropriate number of servings from Pot `p` as the course for
9      Table `t`.
10     Returns:
11       the cost of the food served (as USD with arbitrary fractions), as a float.
12     """
13     avail = p.servings
14     if avail == "infinite" or avail >= t.seated:
15         if avail != "infinite":
16             p.servings = p.servings - t.seated
17         t.course = p.dish
18         return p.pps*t.seated
19     else:
20         return serve2(t, p.sub)
21
22 def serve2(t, p):
23     """Does the same as serve()."""
24     if p.servings == "infinite" or p.servings >= t.seated:
25         if p.servings != "infinite":
26             new = p.servings - t.seated
27             p.servings = new
28         t.course = p.dish
29         return p.pps*t.seated
30     else:
31         redirect = serve3(t, p.sub)
32         return redirect
33
34 def serve3(t3, p3):
35     """Does the same as serve(), except won't serve anything if p doesn't
36     have the capacity."""
37     if p3.servings == "infinite" or p3.servings >= t3.seated:
38         if p3.servings != "infinite":
39             p3.servings = p3.servings - t3.seated
40         t3.course = p3.dish
41         return p3.pps*t3.seated
42     else:
43         return 0
44
45 if __name__ == '__main__':
46     front = Table(5)
47     back = Table(13)
48
49     p1 = Pot("Injera", "infinite", .01, None)
50     p2 = Pot("Misir Wat", 14, 12.55, p1)
51     p3 = Pot("Doro Wat", 7, 15.99, p2)
52
53     earned = serve(front, p2)
54     earned = earned + serve(back, p3)
```

9.2 a2_objects.py

```
# a2_objects.py
# Lillian Lee, Feb 21, 2022

from pot import Pot

p1 = Pot("Injera", "infinite", .01, None)
p2 = Pot("Misir Wat", 14, 12.55, p1)
p3 = Pot("Doro Wat", 7, 15.99, p2)
p4 = Pot("Misir Wat", 10, 12.55, p2)

# STUDENTS: What are p3 and p4's dishes and the dishes
# of their substitutes?  Answer this by writing on your answer sheet:
#
#   Q1
#   p3.dish: ...
#   p3.sub.dish: ...
#   p4.dish: ...
#   p4.sub.dish:  ...
#
# and replace each "..." with the corresponding value


p2.dish = "Tibs Wat"
p2.servings = 10
p2.pps = 16.99
# STUDENTS: Did p3 or p4's dishes or those of their substitutes change?
# Answer this by writing on your answer sheet:
#
#   Q2
#   p3.dish: ...
#   p3.sub.dish: ...
#   p4.dish: ...
#   p4.sub.dish:  ...
#
# and replace each "..." with the corresponding value

p4.sub = Pot("Tibs Wat", 10, 16.99, None)
# STUDENTS: NOW did p3 or p4's dishes or those of their substitutes change?
# Answer this by writing on your answer sheet:
#
#   Q3
#   p3.dish: ...
#   p3.sub.dish: ...
#   p4.dish: ...
#   p4.sub.dish:  ...
#
# and replace each "..." with the corresponding value

p4.sub = Pot("Shiro Wat", 10, 13.99, None)
# STUDENTS: What about now?
# Answer this by writing on your answer sheet:
#
#   Q4
#   p3.dish: ...
#   p3.sub.dish: ...
#   p4.dish: ...
#   p4.sub.dish:  ...
#
# and replace each "..." with the corresponding value
```