CS 1110 Spring 2017, Assignment 2: Frame and Object Notation*

Due on CMS **March 7th, 2017, 11:59pm**. This assignment should take you only two or three hours.

We strongly recommend that you submit by 30-60 minutes before the deadline. Besides the usual reasons[1], this assignment involves work that you might do by hand and then need to scan in order to upload to CMS.

Contents

1 Rules

1.1 One-shot Submission From Now On

This assignment, and all subsequent ones unless otherwise noted, will receive a single grade; there is no revise-and-resubmit like there was for Assignment 1.

1.2 How To Partner (You Only Get One)

You may do this assignment in groups of size **one or two**. You need not have the same group as you had in A1.

If you are going to work together with someone, then *form your group on CMS* before submitting.[2]

Reminder: If your partnership dissolves, see the course Academic Integrity description about "group divorce" on what to do.

1.3 Reminder: What Collaborations Are (Dis-)Allowed And How To Document Them

Our policies are laid out in full on the course Academic Integrity page, but we re-state here **the main rules**: where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's work in any form.

*Authors: Walker White, Lillian Lee, Steve Marschner, Stephen McDowell, Erik Andersen

[1]Reminders: you can replace older submissions with improved ones up to the actual deadline. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time — by CMS's definition of time, which may differ from your watch's or your computer's — for whatever reason, including server delays stemming from many other students trying to submit at the same time. No so-called "slipdays" and this time we really mean it, no "you get to submit late at the price of a late penalty" for this assignment.

[2]Reminder: This requires actions on the part of both parties. Brief instructions for how to form a CMS group are on the course Assignment page.

2. Never show or share any portion of your work in any form to anyone except a member of the course staff.
3. Never request solutions from outside sources; for example, on online services like StackOverflow.
4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

2 Learning Objectives — Which Have Grading Implications

The purpose of this assignment is to give you practice executing Python code on paper using the notation we have introduced in class. You need to be able to do this quickly and confidently—uncertainty about what to draw is a sign that you don't really understand what the programs you write are going to do.

The importance of the notation we have introduced in class is that it gives us a precise visual language for describing and understanding exactly what Python is doing when it executes code. In this assignment, you will use this notation to demonstrate your understanding of what happens during the execution of a given sequence of statements, and on exams you will likewise use this same language to prove your understanding. 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 what 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 that serve the purpose of holding 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 assigned to the corresponding parameter variables.
4. Which statements of an if-statement are executed in a given setting.

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

1. Drawing fewer or more objects than the number of constructor expressions that are evaluated during execution.
2. In the frame for a call to a function, not drawing a correspondingly named box for each parameter in that function's header.
3. Drawing non-existent variables (indicating that you believe in their existence).

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

1. 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`].
2. 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].
3. 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 Notational Conventions (Some of These Differ From Previous Semesters)

When you are using the frames-and-objects notation on paper:

1. 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.
2. 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.
3. If a function call returns some value v, write "Return $\boxed{v}$" in the frame, with a box around the value. For example, "Return $\boxed{3}$" for a function call that returned the integer value 3. (Be sure you are writing a value, not a variable name.)
4. Place your frames so that their position reflects the order in which they were created; we recommend starting at the top and drawing each frame below the previous one.

3.1 Depicting the Creation of New Objects

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.[3]

3.2 Worked Example

You can see the Spring 2014 version of this course for a worked example. The example draws from a piece of code called `a2_example.py`. Two diagrams of the execution of this code are provided, one by Prof. Lillian Lee and one by Prof. Steve Marschner.

Do note that Spring 2014 had a different convention about where to indicate the return value than what we have instructed you in Section 3 item (3) above.

We provide two solutions to give an idea about variations in notation we don't care about, such as whether or not you draw a box around the class name in the upper-right of an object.

4 The Code Whose Execution You Are To Diagram

On page 5 of this handout, and also in the file a2.py, is the code you are to work with. Your submission should consist of a diagram, compliant with the notational conventions given in Section 3, showing what happens during the execution of lines 51–57 when you run this code by typing "`python a2.py`".

The class definition for `Acct` in lines 5–8 means that a constructor call like `Acct(16.0)` creates a new `Acct` object with a `balance` attribute holding the value 16.0.

Employing our notation correctly is a way of proving that you know what `a1`, `a2`, `a3`, `x1`, and `x2` store or refer to, and that you know whether or not each function does what its specification claims it does. We'll tell you that our solution depicts 5 global variables, between 2 and 8 objects (inclusive), and between 5 and 8 crossed-out frames (inclusive). Our solution also indicates to us that at least one of the function specifications is misleading.

4.1 Using Python Tutor

We highly recommend that you run a2.py on your computer, and add some print statements to show you what values ended up where. You can catch most mistakes you might make by comparing your on-paper results to the results of actual execution.

You are also highly encouraged to use Python Tutor to step through the code above line by line and thus get a clearer picture of what Python does during execution. If you want to use Python Tutor, copy the contents of the file a2.py into it. (Don't copy from this document, because you don't want to include the line numbers in Python Tutor.) You may want to change Python Tutor's "draw references using arrows" option to "use text labels for references".

Be aware that some of Python Tutor's notational and display conventions differ from what we use in this class and are expecting on this assignment. Examples: it removes "lost" objects, which we don't; we are not including

[3] That is, for now, don't worry about the `__init__` method in a class or draw a frame for it.

class folders or function objects (unless necessary—and it's not necessary in this assignment); Python Tutor produces a frame for `__init__` whereas we're not asking you to do this for this assignment, as noted in Section 3.1.

And note that on exams, you will need to understand what objects, frames, and variables are created and altered *without* having Python Tutor by your side. Just as an accomplished violinist can perform a concerto without the score and a neurosurgeon can identify the hippocampus without referring to a diagram, a proficient programmer can understand code by reading and diagramming it, without having to try it out to see what it does. The ability to flawlessly predict the execution of programs without access to a computer will tell you (and us!) that you truly know how Python programs work.

Moreover, as we may have mentioned with respect to iClicker questions in class, you often learn more from making mistakes and being corrected than from being right the first time. We thus recommend that you try doing this assignment a bit at a time by hand, each time checking your partial answers against what Python Tutor produces.

5 Turning in the Assignment

5.1 How to Document Your Name(s) and NetID(s)

When you finish the assignment, **put the names and netids of *all* group members, listed alphabetically by last name, last names underlined, first names *not* underlined**, at the top of the page.[4]

We highly recommend that before submission, you double-check your answers against the "Learning Objectives — Which Have Grading Implications" section above.

5.2 How To Create An Uploadable Version

5.2.1 PDF (Perhaps Of A Scan Of Handwritten Work) Highly Preferred

You can work on paper and then scan your paper as a PDF. There are scanners in Olin and Uris Library to help you with this.

Your solution must be a single PDF file. If your scanner makes more than one file, combine them together. On Windows, the program PDFtk can merge multiple PDF files into one file. In OS X, the built-in application Preview can be used to merge PDF files.

Your final file must be less than 100MB in size; ideally it should be less than 10MB. This should not be a problem as long as the resolution is reasonable. Do not scan at a higher resolution than 300 dpi; we do not need that level of detail. If your file is too large, use the web page SmallPDF to reduce your file size.

5.2.2 Other File Formats

We will be printing out your submissions to grade. We worry that photos of your work taken from your phone might not print out well enough for us to read, and if we can't read your assignment, we can't give you credit for it. That being said, if you want to risk this, CMS will allow you to upload a PNG or JPG file.

[4] We will be printing out the submissions for grading. Without your names on the paper, we will not know that the assignment is yours, and we cannot give you credit for your work; and if we can't tell your first name(s) from your last name(s), we may misfile your work or your grade.

```
# a2.py
# Lillian Lee (LJL2) and Erik Andersen (ELA63)
# Feb 27, 2017

class Acct(object):
    """Accts have a balance attribute that is a float (can be negative or 0)"""
    def __init__(self, amt):
        self.balance = amt

def transfer(source, to, amt):
    """Move float <amt> from balance of Acct <source> to balance of Acct <to>"""
    source.balance = source.balance - amt
    to.balance = to.balance + amt

def transfer_to_new(a):
    """Returns new Acct with balance of Acct <a> moved to the new Acct's balance"""
    new = Acct(0.0)
    transfer(a,new,a.balance)
    return new

def equalize(a1, a2):
    """Change balance of both Acct a1 and Acct a2 to the average balance of the
    two Accts."""
    avg = (a1.balance + a2.balance)/2.0
    a1.balance = avg
    a2.balance = a1.balance

def smaller_into_bigger(a1,a2):
    """(1) Empties the smaller Acct of a1 and a2 into the bigger one, unless
           the smaller has a balance less than 10, in which case no change is
           made.
       (2) If a change is made, a1 becomes the bigger Acct and a2 becomes
           the smaller Acct.
       (2) Returns -1 if no change was made; otherwise, returns the amount that
           was moved."""
    if a1.balance < 10 or a2.balance < 10:
        return -1
    elif a1.balance < a2.balance:
        smaller = a1
        bigger = a2
    else:
        smaller = a2
        bigger = a1

    output = smaller.balance
    transfer(smaller, bigger, output)
    a1 = bigger
    a2 = smaller
    return output

a1 = Acct(10.0)
a2 = Acct(30.0)
a3 = transfer_to_new(a2)
equalize(a2,a1)
transfer(a3,a1,10.0)
x1 = smaller_into_bigger(a1, a2)
x2 = smaller_into_bigger(a1,a3)
```