CS 1110 Spring 2018, Assignment 2: Frame and Object Notation, Public Goods Games*

March 5, 2018

1 Worked Examples

First, to help solidify the concepts that are exercised in this assignment, we provide some worked examples of diagramming variables, objects, and call frames. We strongly recommend that you try these examples out as soon as possible, and will be very happy to go over these with you at consulting/office hours.

Spring 2014 That semester had a different convention about where to indicate the return value, but otherwise the notation is the same.

Here is a small piece of code and three different corresponding diagrams: one done as a video by Prof. Bracy, one by Prof. Lillian Lee and one by Prof. Steve Marschner. We gave independent solutions to indicate the kinds of variations in notation we don't care about.[1]

Spring 2017 Here is a previous assignment A2 — focus on section 4 and on page 5 — and the solutions.

2 Motivation

We seek to simulate a *public goods game*. Here's a quick definition from the Wikipedia article[2], with some adaptations to match the code we give you.

> The *public goods game* is a standard of experimental economics. In the basic game, [players] secretly choose how many of their private tokens [from their *holdings*] to put into a public pot. The tokens in this pot are multiplied by a factor (greater than one and less than the number of players . . .) and this "public good" payoff is evenly divided among players. Each [player] also keeps the tokens they do not contribute. . . . The group's total payoff is maximized when everyone contributes all of their tokens to the public pool. However, the Nash equilibrium in this game is simply zero contributions by all . . . [In a two-player game, if you contribute nothing but the other player contributes something, you the "free rider" gain tokens and the other player *loses* tokens.]

If you were playing this game, what would you do? What do you think most people do? Would your answer change if there were multiple rounds of the game, or if the factor were changed, or . . . ? We could program a simulation of this game to explore some of these questions.

Contents

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

[1] Examples: whether or not you draw a box around the class name in the upper-right of an object; or whether you put your global variables in a column, a row, or in a cluster.

[2] version consulted on Mar 1, 2018.

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.

3.2 CMS Groups Are Not Preserved Across Assignments — You Need To Re-group (So To Speak)

If you are partnering, regardless of whether you were grouped in a previous assignment, **the two of you must still form a new group *specifically for this assignment* on CMS before submitting**.[3]

4 Rules that are the same as the previous assignment

4.1 How To Partner (You Only Get One)

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 your partnership dissolves, see the course Academic Integrity description about "group divorce" for what to do.

4.2 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.
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).

5 Learning Objectives, Which Have Grading Implications

In this assignment, 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:

[3] This links your submission "portals".

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 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 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].

6 Notational Conventions (Some Differ From Previous Semesters)

1. Do not draw multiple versions of the same thing.[4] So, for example, there should only be one call frame on your paper for one function call, no matter how many individual lines of that function are executed.
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. 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.
4. If a function call returns some value v, write "Return $\boxed{v}$" in the frame, with a box around the value, e.g., "Return $\boxed{3}$" for a function call that returned the integer value 3. (Be sure you are writing a value, not a variable name.)
5. 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.
6. Don't draw the objects (folders) for imported modules or for function definitions.
7. Don't draw call frames for built-in functions, such as `int()`.
8. 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.[5]

[4] This is the one significant difference between Fall CS1110 and Spring 1110 call-frame notation.
[5] 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 Your Task

On page 5 of this handout, and also in file a2.py (which imports player.py), is the code you are to work with.

The definition for class `Player` (which can be found in the file `player.py`) means that a constructor expression like `Player(16)` creates a new `Player` object with a `holdings` attribute having the value `16`.

Your submission should consist of a diagram, compliant with the notational conventions given in Section 6 and class, showing what happens during the execution of it. Use the line numbers given on page 5 of this handout.

We'll tell you that our solution depicts 3 global variables, between 2 and 8 objects (inclusive), and between 5 and 12 crossed-out frames (possibly inclusive).

7.1 Need Help? Try Python Tutor

You can catch many errors by comparing your on-paper results to the results of Python Tutor. But, first try the worked examples by hand, and try doing this assignment manually, beforehand. One often learns more from making mistakes and being corrected than by just seeing someone else or some program solve a problem for us.

You can copy the `a2.py` code into the main tab of Python Tutor and the contents of any files it imports into separate tabs, making sure to change the tab names to the corresponding module names. 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

8.1 Documenting Your Name(s) and NetID(s)

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.[6]

8.2 Creating a PDF From Handwritten Work (Plan Ahead To Make Time For This)

There are scanners in Olin and Uris Library, other possibilities and computing labs equipped with scanners can be found by selecting "Peripherals" in the sidebar and then "Scanners" here.[7]

Your solution must be a single PDF file.[8] It 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. Do not scan at a higher resolution than 300 dpi.[9]

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.[10]

8.3 Due dates

1. If you are partnering: well before submission, follow the instructions in the "How to form a group" instructions on the course Assignments page. Both parties need to act on CMS in order for the grouping to take effect.

2. By 2pm on Thu March 8, submit whatever you have done at that point to CMS, following steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of the course Assignments page. It is OK if you haven't finished working on the files yet.

3. By **11:59pm on Thu March 8**, make your final submission, again following the aforementioned steps 1-3.[11]

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

[6] We will be printing out the submissions for grading. Given this, there are significant administrative difficulties caused by non-compliance; so be warned that we typically deduct some points for failure to follow these directions.

[7] Disclaimer: the CS1110 staff cannot ascertain for you or guarantee that a particular scanner is available or in working order.

[8] For pdf file merging, there is the program PDFtk for Windows and the built-in application Preview for OS X.

[9] If your file is still too large, try SmallPDF to compress it.

[10] TA Anthony Poon recommends Genius Scan, which processes images into a flat, perspective-corrected, and evenly-lit PDF.

[11] The 2pm checkpoint on Thu March 8 provides you a chance to alert us during business hours if any problems arise. 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, for whatever reason. There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy. Of course, if some special circumstances arise, contact the instructor(s) immediately.

```python
# a2.py
# Prof. Lillian Lee (LJL2)
# Feb 27, 2018

""" Code for A2 diagramming exercise."""

from player import Player


def change(p, delta):
    """Adds delta to p's holdings unless doing so would make p's holdings
        negative, in which case nothing is added.
        Returns True if adding delta keeps p's holdings non-negative, False o.w.

    p is a Player.
    delta is an int, possibly negative or zero."""

    if delta < -p.holdings:
        return False
    else:
        p.holdings = p.holdings + delta
        return True


def redistr(p1, contrib1, p2, contrib2, factor):
    """Play a public goods game.

    Players p1 and p2 contribute contrib1 and contrib2 tokens, respectively,
    with the pot multiplied by `factor` before fair division between p1 and p2
    (truncated to an int).

    However, no change is made to if p1's holdings are less than contrib1 or
    p2's holdings are less than contrib2.

    p1 and p2 are Players.
    contrib1 and contrib2 are non-negative ints.
    factor is a positive float between 1 and 2 exclusive.
    """
    if not change(p1, -contrib1):
        return
    elif not change(p2, -contrib2):
        change(p1, contrib1)
        return

    get_back = int((contrib1+contrib2)*factor/2) # float division
    change(p1, get_back)
    change(p2, get_back)


p1 = Player(20)
p2 = Player(30)
p3 = Player(2)
redistr(p1, 4, p2, 6, 1.5)
redistr(p3, 2, p1, 30, 1.7)
```