id #: 2668657 name: Ashlesha Bhagat

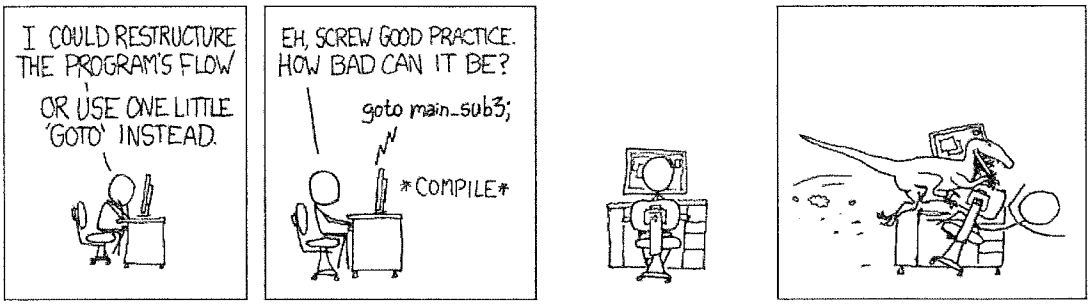
PROGRAMMING ON PURPOSE – MIDTERM

CSC 120, Winter 2023

Aaron G. Cass
Department of Computer Science
Union College

INSTRUCTIONS

1. This exam is opened-notes. You may use your own projects and labs, and your personal notes from this term.
2. You have 65 minutes to work on these problems. Please use the time wisely. Please first read and understand all questions to determine which ones will require more of your time.
3. Some of the questions refer to "Listings". These code listings are provided at the end of the exam. You may remove those pages from the exam.
4. There are a total of 100 points available.
5. Please write clearly and neatly. You may use scratch paper to work out preliminary solutions.
6. Write your answers in the provided spaces. If you need additional space, you may attach additional sheets – please clearly indicate if you have done so.
7. Please make your answer clear – if you have multiple solutions, I will not choose the best one for you. Please cross out solutions that you decide are faulty.
8. Please put your name and id # in the provided spaces at the top of this page and your id # at the top of the subsequent pages.
9. When you are done with the exam, please sign the Honor Code Pledge below.
10. **Good luck.**



HONOR CODE PLEDGE

I affirm that I have carried out my academic endeavors with full academic honesty.

Ashlesha Bhagat

Signature

ASHLESHA BHAGAT

Printed name

id #: 26695 7

1. **Tracing and parameter passing modes.** Consider the code in Listing 1, which you will be tracing to answer the following subquestions (you might find it useful to draw memory diagrams):

   (a) [3 pts] When we call `c(a_list, b(x)), a_list` is:

   Circle one: **Passed by value** **Passed by Reference**

   (b) [3 pts] When we call `c(a_list, b(x))`, after `b(x)` is computed, it is:

   Circle one: **Passed by value** **Passed by Reference**

   (c) [3 pts] When we call `advance(c1), c1` is:

   Circle one: **Passed by value** **Passed by Reference**

   (d) [2 pts] What, precisely, is the **first** line of output printed by this program?

   3

   (e) [2 pts] What, precisely, is the **second** line of output printed by this program?

   5

   (f) [2 pts] What, precisely, is the **third** line of output printed by this program?

   '[1, 2, 3, 5' -- 5'

   (g) [2 pts] What, precisely, is the **fourth** line of output printed by this program?

   []

   (h) [2 pts] What, precisely, is the **fifth** line of output printed by this program?

   4

   (i) [2 pts] What, precisely, is the **sixth** line of output printed by this program?

   2

id #: 266%57

2. **Representing.** Consider the python code in Listing 2 for managing an inventory that keeps track of items in rooms:

   (a) [10 pts] What, exactly, will be printed when this program is run?

   Inventory for deck of cards
   in living room : 3
   in kitchen : 5
   TOTAL : 8
   Inventory for deck of cards
   in living room : 6
   in kitchen : 0
   TOTAL : 2
   Inventory for dice
   in kitchen : 5
   TOTAL : 5

   (b) [2 pts] Are there any **global variables** in this code that are manipulated inside any functions? If so, what global variable are manipulated?

   YES there are global variables, "inventory" variable has been manipulated globally. ~~[illegible]~~

   (c) [2 pts] Does this program use **magic numbers**? If so, what numbers are used in a "magic" way? → YES

   -1 has been used as a magic number. -1 basically means that the item is not there in the inventory and needs to be added at the end of program. So we can establish a constant that say is named "NOT_THERE" which has value -1 so that we know item is not there just by reading code.

   (d) [3 pts] If we were to replace the number 2 with a named constant, what would be a good name for it?

   Name

   COUNT/NUM_ITEM = 2

   could be (?)

id #: 2669857

(e) [5 pts] Write a good docstring for the `remove` function.

"""Removes the given number of given items from the room which it is kept in
@ name: the name of item to be removed
@ room_name: the name of room from which item needs to be removed
@ count: the no. of items to be removed
"""

(f) [3 pts] In this program, what representation have we chosen to model information about an item, including what room it's stored in and how many of those items are in that room? Be as specific as possible.

Item is represented as a string value, room as a string value, number items as an integer value together one type of item in a particular room has been represented as a list.

(g) [3 pts] In this program, what representation have we chosen to model the full collection of inventory information – i.e. information about all the items in all of the rooms. Be as specific as possible.

inventory is a list containing two nested lists(nested list) where each nested list has a str value representing item, string representing room, an int value representing number of item.

(h) [3 pts] Does `add_item` rely on our choice of representation for a single inventory item?

Circle one: **Yes** **No**

(i) [3 pts] Does `add_item` rely on our choice of representation for the full collection of inventory information?

Circle one: **Yes** **No**

(j) [3 pts] If we were to re-write this program following the principles of information hiding, what modules would we have (in addition to `main`)?

id #: 2668 9657

3. **OOPs I did it again!.**

   Now imagine you are re-writing the previous program in an object-oriented manner and we already have the two modules shown in Listing 3.

   (a) [2 pts] What is the name of the class used in this program to represent inventory information about a single item?

   InventoryItem

   And in what module is that class expected to be written?

   inventory_item.py

   (b) [15 pts] Write the code for the constructor of that class. Here's the signature and a docstring explaining the desired behavior:

```
def __init__(self, room_name, item_name, count):
  '''
  Stores inventory information about a given item in a given room.

  :param room_name: The name of the room the item is found in.
  :param item_name: The name of the item.
  :param count: The number of the given item found in the given room.
  '''
```

~~self.__room_name = room_name~~
~~self.__item_name = item_name~~
~~self.__count = count~~

```
self.__room_name = room_name
self.__item_name = item_name
self.__count = count
```

id #: 2669657

(c) [5 pts] Write the code for the `get_count` method from that same class.

```
def get_count(self):
    return self.__count
```

(d) [5 pts] Write the code for the `increment` method from that same class.

```
def increment(self, count):
    self.__count += count
```

(e) [15 pts] Write a unit test (using `assert_equals`) to confirm that `decrement` and `get_count` work correctly.

```
inventory_item.py
import test_cute as ts
if __name__ == "__main__":
    test = ts.create()
    item = InventoryItem("deck of cards", "kitchen", 4)
    ts.assert_equals(test, "Checking if get_count is
      working as expected", str(item.get_count()), "4")
    item.decrement(2)
    ts.assert_equals(test, "Checking if decrement is
      working as expected", str(item.get_count()), "2")

    ts.print_summary(test)
```

note: if the functions worked as expected, final line of output will be

2 Tests executed, 2 Passed, 0 Failed

complete, defined in b/w

2669657

1.(e)

7

2, 1, 0 has also been used as magic no.s here because they have a constant meaning throughout & represent a particular item so we can use constants to define

2.(j) We can have two more modules
→ inventory.py : stores info abt inventory as a whole
→ item.py : stores info abt one single item