CS 1110 Prelim 1 Solutions, March 2024

1. [10 points] **Strings.** Implement the following function. *Hint:* see `rfind` on the reference sheet.

```
def inside_markers(text, marker):
    """
    Returns: the substring of `text` inside the 1st and last instance of `marker`
             If the marker exists only once within `text`, returns all of `text`

    Preconditions
       text   [str]: contains at least 1 instance of `marker`
       marker [str]: has length AT LEAST 1 (see examples)

    Examples:
      inside_markers("ab+c+d+e+f+g", "+")  returns "c+d+e+f"
      inside_markers("ab++c++d++e++f+g", "++")  returns "c++d++e"
      inside_markers("hello world", " ")    returns "hello world"
      inside_markers("blah blah", "a")     returns "h bl"
      inside_markers("blah blah blah blah", "blah")     returns " blah blah "
    """

    if text.count(marker) == 1:
        return text
    pad = len(marker)
    start_index = text.find(marker) + pad
    end_index = text.rfind(marker)
    return text[start_index:end_index]

    # Alternate Solution
    first = text.find(marker)
    last = text.rfind(marker)
    if first == last:
        return text
    else:
        return text[first + len(marker): last]
```

2. [12 points] **Lists.** Implement the following function.

```
def outside_in(list1, list2):
    """
    Given input lists `list1` and `list2`, removes the first and last elements
    from `list1` and places them in the _middle_ of `list2` (the first element
    goes before the last element). If `list2` has an odd length, the middle
    element is REPLACED by the two elements from `list1` instead.

    Does not return anything! Just modifies the input lists.
    Remember: list1 is the identifier of a folder on the heap. you should
        be modifying THAT folder on the heap, NOT re-assigning list1 the
        value of a new identifier.

    Examples:
      outside_in([1, 2, 3, 4], [5, 6, 7, 8]) modifies the lists to be:
            -->    [2, 3], [5, 6, 1, 4, 7, 8]
      outside_in(['apple', 'bee'], ['cat', 'DOG', 'egg']) modifies lists to be:
            -->    [], ['cat', 'apple', 'bee', 'egg']
      outside_in(['first1', 'mid1', 'last1'], ['mid2']) modifies lists to be:
            -->    ['mid1'], ['first1', last1']
      outside_in(['a', 1, True],[] )  modifies the lists to be:
            -->    [1], ['a', True] )

    Preconditions: list1 and list2 are lists
                   list1 has length at least 2

    """

    first = list1.pop(0)
    last = list1.pop(-1)
    list2len = len(list2)
    list2_middle = list2len // 2
    if list2len % 2 == 0:
        list2.insert(list2_middle, first)
        list2.insert(list2_middle + 1, last)
    else:
        list2[list2_middle] = first
        list2.insert(list2_middle + 1, last)

    # Alternate Solution
    l2len = len(list2)
    first = list1.pop(0)
    last = list1.pop(len(list1)-1)
    l2mid = l2len // 2
    if l2len % 2:
        list2.pop(l2mid)
    list2.insert(l2mid, last)
    list2.insert(l2mid, first)
```

3. [12 points] **Test cases.** Consider the following function specification, which an online horoscope provider might use when asking for a user's birth date.

```
def is_valid_date(d):
    """  Returns True if `d` is a valid date and False otherwise.

    A valid date is a string in the format of 'MMDD'. The month (MM) must be a
    2-digit number representing one of twelve possible months. The day (DD)
    must be a 2-digit number representing the day of the month. Note that the
    date must be one that exists. The leap date of Feburary 29 is valid.

    Preconditions: `d` is a string containing only digits ('0'-'9')
    """
```

Here is an example of one set of sample inputs and an expected output:

Test Case	Input `m`	Expected Output / `return value`	What the test covers:
1	`"1231"`	`True`	a valid input (a string of the correct format that also represents a date that exists)
2	`"1310"`	`False`	13 is not a valid month

Provide three more conceptually distinct test cases. (Your cases should be distinct from each other and from Test Cases 1-2.) Include a short statement (∼1 sentence) explaining what situation your test cases cover.

Test Case	Input `m`	Expected Output / `return value`	What the test covers:
3	"0229"	`True`	edge case: make sure it accepts leap date
4	"115"	`False`	wrong format, not 4 digits
5	"1241"	`False`	date does not exist 41 is not a valid day

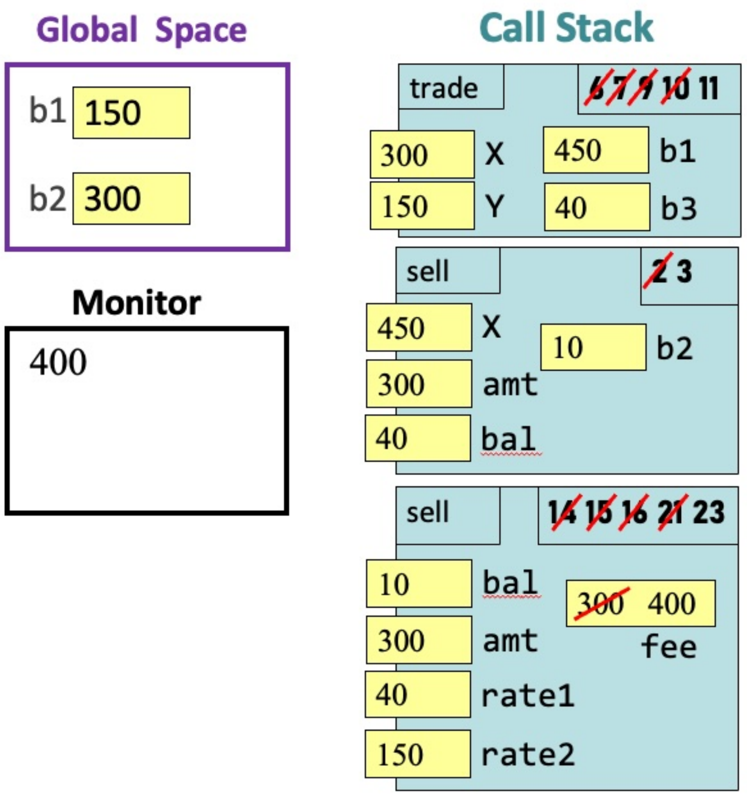
Also acceptable: "0230" , `False`, date does not exist. there is no February 30th

Not acceptable: violating the precondition. Example: `125`, an `int`, is not acceptable

4. **Drawing Time!**

   (a) [14 points] Simulate running the code below (which runs to completion without errors) until Python executes line 21 and reaches the comment `# ! STOP SIMULATION HERE !`. At the stopping point, the instruction counter should have the value 23. Draw the memory diagram as shown in class and for A2. Remember there are 4 possible "regions" in which you might draw program elements: **Global Space**, **Call Stack**, **Heap**, or **Monitor**. (The first 3 are regions in memory, the last one is something that can be observed by a user.) Make sure you label whatever you draw so that it is clear which region your drawn components belong to. As usual, do *not* draw the objects/folders for function definitions. Also, do *not* draw a call frame for `print()`.

```
1   def sell(X, amt, bal):
2       b2 = 10
3       fee = add_fee(b2, amt, bal, b1)
4       return X + amt - fee

5   def trade(X, Y):
6       b1 = X + Y
7       if X < Y:
8           b3 = 20
9       elif X > Y:
10          b3 = 40
11      bal = sell(b1, b2, b3)

13  def add_fee(bal, amt, rate1, rate2):
14      fee = b2
15      if amt > 100 or balance > 10000:
16          fee = fee + 100
17      elif amt > 50:
18          fee = fee - 100
19      else:
20          fee = 5
21      print(fee)
22      # ! STOP SIMULATION HERE!
23      return fee

24  b1 = 150
25  b2 = 300
26  trade(b2, b1)
```



(b) [2 points] Take a closer look at the function `sell`. It is possible to give `sell` an argument for the parameter `X` that makes Python throw an Error while executing one of the lines of `sell` (lines 2-4). Provide a first argument to `sell` that would make this happen.

```
sell(  ____________ ,  ....)
```

The way to make Python throw an error while executing `sell` is to give `X` a value with a type that cannot be added to amt on line 4. Example: `'hello'`.

Not acceptable: something that would make Python throw an error when the argument is evaluated on line 11.

(c) [2 points] Take a closer look at the function `add_fee`. It is possible to give `add_fee` four `int` arguments that make Python throw an Error while executing one of the lines of `add_fee` (lines 14-23). Provide four integer arguments that would make this happen.

```
add_fee(  ____________ ,  ____________ ,   ____________ ,  ____________ )
```

The way to make Python throw an error while executing `add_fee` is to give `amt` (the second argument) a value less than or equal to 100. This way, on line 15, the expression `amt > 100` will evaluate to `False` and so the code `balance > 10000` be evaluated. At this point, Python throws an error because there is no such variable `balance`. The concept here is short circuit evaluation (from the lab).

(d) [2 points] Take a closer look at the function `trade`. It is possible to give `trade` two `int` arguments that make Python throw an Error while executing one of the lines of `trade` (lines 6-11). Provide two integer arguments that would make this happen.

```
trade(  ____________ ,  ____________ )
```

The way to make Python throw an error while executing `trade` is to give values for `X` and `Y` that are equal. If so, neither line 8 nor line 10 will be executed. This means `b3` will not be created. This will make Python throw an error when executing line 11. See the lecture notes on conditionals about making sure variables are created across all possible clauses in cascading `if-elif-else` statements.

5. **Objects.** You may want to look at part (b) as you work through part (a). In particular, drawing the state of memory after the first line of code may help you work on part (a).

Objects of class `Library` have 3 attributes:

- `shelf` [`list` or `None`]: unique `int` IDs of books available in the Library
- `cap` [`int`]: maximum number of books that can be stored in the library's bookshelf
- `accept` [`boolean`]: whether the library can store more books in the bookshelf

A call of the form `Library()` returns the identifier of a new `Library` object in an "initial state": `shelf` has the value `None`, `cap` has the value `0`, and `accept` has the value `False`.

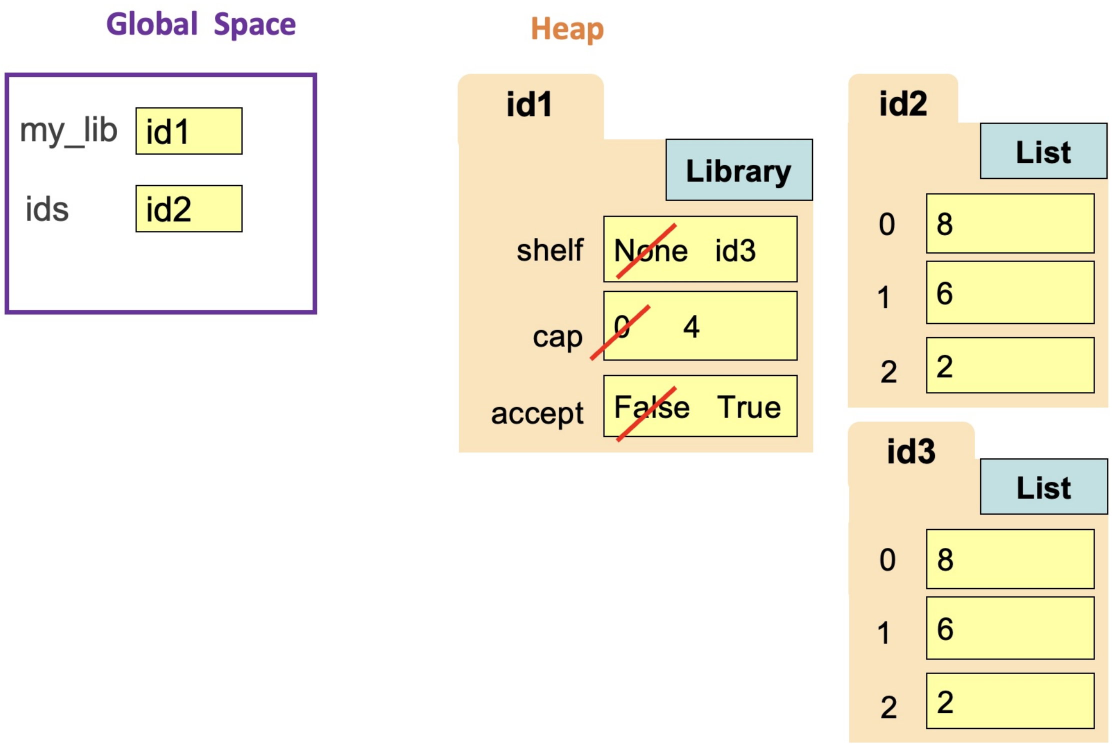
(a) [6 points] Implement the following function according to its specification.

```
def open_library(lib, books, c):
    """ Assigns Library `lib` attributes as follows:
      - shelf: set to a new list containing the first `c` IDs in `books`
      - cap: set to `c`
      - accept: set to True if the shelf has room for more books

    Preconditions: lib [Library]: a Library object in an "initial state"
                   books [list]: a non-empty list of book IDs (ints)
                   c [int]: a positive number

    the function modifies `lib`, does not modify `books`, returns nothing
    Examples:
        after calling open_library(lib1, [0, 1, 2, 3, 4], 8),
            lib1 should have the attributes:
            - shelf is a list with elements 0, 1, 2, 3, 4
            - cap is 8
            - accept is True

        after calling open_library(lib2, [100, 3, 7, 98, 34, 2, 45], 5),
            lib2 should have the attributes:
            - shelf is a list with elements 100, 3, 7, 98, 34
            - cap is 5
            - accept is False                                              """
    lib.shelf = books[:c] # works even if c is larger than len
    lib.cap = c
    lib.accept = c > len(books)

    # Alternate Solution
    lib.shelf = books[:min(len(books), c)]
    lib.cap = c
    if c > len(books):
        lib.accept = True
    else:
        lib.accept = False
```

(b) [10 points] Assume the class `Library` and the function `open_library` are accessible within the given code. Draw the memory diagram (as shown in class and for A2) after the following lines of code have been executed. Remember there are 4 possible "regions" in which you might draw program elements: **Global Space**, **Call Stack**, **Heap**, or **Monitor**. (The first 3 are regions in memory, the last one is something that can be observed by a user.) However, **do not draw *any* call frames**. Make sure you label whatever you draw so that it is clear which region your drawn components belong to.

Note: even if you did not complete the previous part, you can still do this part based on what the functions `Library()` and `open_library()` should do. Your drawings may even help you complete part (a) if you "see" what your code should be doing.)

```
my_lib = Library()
ids = [8,6,2]
open_library(my_lib, ids, 4)
```



(c) [10 points] Implement the following function (which simulates someone borrowing a book from the library) according to its specification:

```
def borrow(lib, id):
    """
    Given a object `lib` and int `id`, a successful borrow:
       Moves `id` out of `lib`'s bookshelf
       Updates accept accordingly
       Returns True (to indicate success)

    One situation would prevent a successful borrowing:
       `id` might _not_ be on `lib`'s shelf
       The return value should reflect the failure

    Preconditions:
        lib [Library]: a Library object. `lib` could be in the "initial state"
                       (newly constructed) or it could be open (having already
                       had open_library called for it)
        id [int]: an integer representing the ID of a book to borrow
    """

    if lib.shelf == None:
        return False
    if id in lib.shelf:
            lib.shelf.remove(id)
            lib.accept = (lib.cap > len(lib.shelf))
            return True
    return False

    # Alternate Solution #1:
    # ----------------------
    if lib.shelf == None or not id in lib.shelf:
        return False
    lib.shelf.remove(id)
    lib.accept = True
    return True

    # Alternate Solution #2:
    # ----------------------
    if lib.shelf == None:
        return False
    if id in lib.shelf:
        lib.shelf.remove(id)
        if lib.cap > len(lib.shelf):
            lib.accept = True
        return True
    return False
```

6. [8 points] **Understanding Python.** Consider the `Point3` class as it was defined in lecture. A call to the constructor of the form `Point3(1,2,3)` will set attributes `x`, `y`, and `z` to have values `1`, `2`, and `3`, respectively. Assume the class `Point3` is accessible to the code below. What is printed out when each code snippet below is executed? Write `ERROR` as shorthand for any error output.

```
p1 = Point3(8,9,10)
p1 = Point3(2,4,6)
p2 = p1
p2.x = 10
print(print(p1.x))
```

Your Answer:

Correct Answer: **10**
None

```
p1 = Point3(1,2,3)
p2 = Point3(4,5,6)
h = [p2,p1,p1,p2]
h[1].x = h[0].z
h[3].x = h[2].x
h[0].x = h[1].y
print(h[3].x)
```

Your Answer:

Correct Answer: **2**

```
p1 = Point3(5,2,8)
p2 = Point3(1,4,7)
p1.x = p2.x
p2.y = p1.y
p2 = p1.x
print(p2.y)
```

Your Answer:

Correct Answer: **ERROR**

```
p1 = Point3(1,2,3)
p2 = Point3(4,5,6)
h = [p1,p2]
a = h[h[0].x].y
print(a)
```

Your Answer:

Correct Answer: **5**