CS 1110 Prelim 1, March 2024

This 90-minute (1.5 hour) closed-book, closed-notes exam has 6 questions worth a total of roughly 88 points (some point-total adjustment may occur during grading). Pace yourself accordingly. You may separate the pages while working on the exam; we have a stapler available.

Note: iteration / `for`-loops are not allowed or needed for any question of this exam.

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any reference material besides the reference provided in the exam itself, or to otherwise give or receive unauthorized help. We also ask that you not discuss this exam with students who are scheduled to take a later makeup.

Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: ______________________________ Date ____________

Name (First Last):

Cornell NetID, all caps:

This is a comprehensive reference sheet that might include functions or methods not needed for your exam.

String methods	
`s[i:j]`	Returns: if `i` and `j` are non-negative indices and `i` $\leq$ `j-1`, a new string containing the characters in `s` from index `i` to index `j-1`, or the substring of `s` starting at `i` if `j` $\geq$ `len(s)`
`s.count(s1)`	Returns: the number of times `s1` occurs in string `s`
`s.find(s1)`	Returns: index of first occurrence of string `s1` in string `s` (-1 if not found)
`s.find(s1,n)`	Returns: index of first occurrence of string `s1` in string `s` STARTING at position n. (-1 if `s1` not found in `s` from this position)
`s.index(s1)`	Returns: index of first occurrence of string `s1` in string `s`; raises an error if `s1` is not found in `s`.
`s.index(s1,n)`	Returns: index of first occurrence of string `s1` in string `s` STARTING at position n; raises an error if `s1` is not found in `s` from this position
`s.isalpha()`	Returns: `True` if `s` is *not empty* and its elements are all letters; it returns `False` otherwise.
`s.isdigit()`	Returns: `True` if `s` is *not empty* and its elements are all numbers; it returns `False` otherwise.
`s.islower()`	Returns: `True` if `s` is has at least one letter and all letters are lower case; returns `False` otherwise (*e.g.*, 'a123' is `True` but '123' is `False`).
`s.isupper()`	Returns: `True` if `s` is has at least one letter and all letters are upper case; returns `False` otherwise (*e.g.*, 'A123' is `True` but '123' is `False`).
`s.lower()`	Returns: a copy of `s`, all letters converted to lower case.
`s.join(slist)`	Returns: a string that is the concatenation of the strings in list `slist` separated by string `s`
`s.replace(a,b)`	Returns: a *copy* of `s` where all instances of `a` are replaced with `b`
`s.rfind(s1)`	Returns: the highest index in string `s` where string `s1` is found (-1 if not found)
`s.split(sep)`	Returns: a list of the "words" in string `s`, using `sep` as the word delimiter (whitespace if `sep` not given)
`s.strip()`	Returns: copy of string `s` where all whitespace has been removed from the beginning and the end of `s`. Whitespace not at the ends is preserved.
`s.upper()`	Returns: a copy of `s`, all letters converted to upper case.

List methods	
`lt[i:j]`	Returns: if `i` and `j` are non-negative indices and `i` $\leq$ `j-1`, a new list containing the elements in `lt` from index `i` to index `j-1`, or the sublist of `lt` starting at `i` if `j` $\geq$ `len(s)`
`lt.append(item)`	Adds `item` to the end of list `lt`
`lt.count(item)`	Returns: count of how many times `item` occurs in list `lt`
`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` is not found. (There's no "find()" for lists.)
`lt.index(y, n)`	Returns: index of first occurrence of `item` in list `lt` STARTING at position `n`; raises an error if `item` does not occur in `lt`.
`lt.insert(i,item)`	Insert `item` into list `lt` at position `i`
`lt.pop(i)`	Returns: element of list `lt` at index `i` **and also removes that element from the list** `lt`. Raises an error if `i` is an invalid index.
`lt.remove(item)`	Removes the first occurrence of `item` from list `lt`; raises an error if `item` not found.
`lt.reverse()`	Reverses the list `lt` in place (so, `lt` is modified)
`lt.sort()`	Rearranges the elements of `x` to be in ascending order.

Other useful functions	
`s1 in s`	Returns: `True` if the substring `s1` is in string `s`; `False` otherwise.
`elem in lt`	Returns: `True` if the element `elem` is in list `lt`; `False` otherwise.
`input(s)`	prompts user for a response using string `s`; returns the user's response as a string.
`isinstance(o, c)`	Returns: `True` if `o` is an instance of class `c`; `False` otherwise.
`len(s)`	Returns: number of characters in string `s`; it can be 0.
`len(lt)`	Returns: number of items in list `lt`; it can be 0.
`list(range(n))`	Returns: the list `[0 .. n-1]`

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 "
    """
```

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

    """
```

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			
4			
5			

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(  ____________ ,  ....)
```

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

(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(  ____________ ,  ____________ )
```

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                                          """
```

(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
    """
```

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:

```
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:

```
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:

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

Your Answer: