CS 1110 Prelim 1, March 2023

This 90-minute (1.5 hour) closed-book, closed-notes exam has 6 questions worth a total of roughly 90 points (some point-total adjustment may occur during grading).

You may separate the pages while working on the exam; we have a stapler available.

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 $j \ge len(s)$	
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 <i>copy</i> of s where all instances of a are replaced with b	
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 1t	
	from index i to index j-1, or the sublist of lt starting at i if $j \ge len(s)$	
<pre>lt.append(item)</pre>	Adds item to the end of list lt	
<pre>lt.count(item)</pre>	Returns: count of how many times item occurs in list 1t	
<pre>lt.index(item)</pre>	Returns: index of first occurrence of item in list lt; raises an error if item is not found. (There's no	
	"find()" for lists.)	
<pre>lt.index(y, n)</pre>	Returns: index of first occurrence of item in list 1t STARTING at position n; raises an error if item	
	does not occur in lt.	
<pre>lt.insert(i,item)</pre>	Insert item into list 1t at position i	
lt.pop(i)	Returns: element of list 1t at index i and also removes that element from the list 1t. 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 1t in place (so, 1t 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 \mathbf{s} ; it can be 0.	
len(lt)	Returns: number of items in list 1t; it can be 0.	
list(range(n))	Returns: the list [0 n-1]	

1. [8 points] **Strings.** Implement the following function.

```
def before_first_inclusive(text, marker):
   11 11 11
   Returns: substring of `text` up to and including the first
   occurrence of `marker`.
   If `marker` is the empty string then return all of `text`
   Examples:
      before_first_inclusive( "abc", "abc" )
                                              --> "abc"
      before_first_inclusive( "abcabc", "abc" ) --> "abc"
      before_first_inclusive( "abc", "a" )
                                               -->
                                                     "a"
      before_first_inclusive( "a", "a" )
                                                     "a"
                                               -->
      before_first_inclusive( "abba", "b" )
before_first_inclusive( "abba", "" )
                                               --> "ab"
                                               --> "abba"
   Preconditions:
      text: non-empty string containing at least one instance of `marker`
      marker: string found in `text`
   0.00
```

2. [12 points] Lists. Implement the following function. *Hint:* see reverse on the reference sheet.

```
def mirror_me(mylist, i):
    11 11 11
    Given an input list `mylist` and an index `i`, returns a new list which
    is a copy of `mylist` except all elements after index `i` are reversed.
    `mylist` should not be modified
    Examples:
        mirror_me([1,2,3,4,5,6], 2) returns [1,2,3,6,5,4]
        mirror_me([9,2,1,8,4], 4) returns [9,2,1,8,4]
        mirror_me([5,8], 1) returns [5,8]
        mirror_me([5], 0) returns [5]
        mirror_me([], 0) returns []
    Preconditions: 0 <= i < len(mylist)</pre>
                   mylist is not None but could be empty
    \Pi/\Pi/\Pi
    # STUDENTS: loops are NOT ALLOWED (or needed)
    # STUDENTS: conditionals (if-else) are NOT ALLOWED (or needed)
```

3. [10 points] **Test cases.** Consider the following function specification, which one might use if to transmit messages one character at a time.

```
def is_valid_message(m):
```

""" Returns True if `m` is a valid message and False otherwise.

A valid message is a list whose elements consist of only characters (strings of length one). Additionally, a message must contain at least two elements in order to be valid.

```
Pre-condition: m is a list
```

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

Test Case	Input m	Expected Output / return value
1	["n","o","","d","i","c","e"]	False

Test Case 1 covers the following situation:

m is not valid because it has an element that is a string whose length is not 1.

Complete the table for **three** more conceptually distinct test cases, using the same format. (Your cases should be distinct from each other and from Test Case 1.)

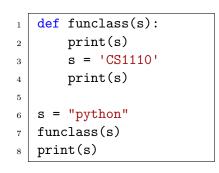
Test Case	Input m	Expected Output / return value
2		True
3		False
4		False

Include a short statement (1-2 sentences) explaining what situation **Test Cases 3 and 4** cover.

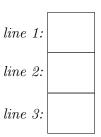
Test Case 3 covers the following situation:

Test Case 4 covers the following situation:

4. [12 points] Understanding Python. For each snippet of code below, at most 3 lines will be printed. What are these three lines? Put a single letter (A-E) in each box.



Your Answer:



CHOICES:

- (A) python
- (B) CS1110
- (C) None
- (D) Error
- (E) Nothing printed due to an Error in a previous box.

```
def funclass(s):
    print(s)
    s = 'CS1110'
    return s

s = "python"
    s = funclass(print(s))
    print(s)
```

Your Answer:

line 1:	
line 2:	
line 3:	

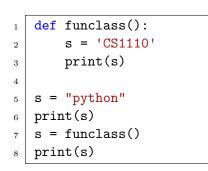
```
def funclass():
    s = 'CS1110'
    print(s)
    return s

s = "python"
    print(funclass())
    print(s)
```

Your Answer:

line 1:	
line 2:	
line 3:	

Your Answer:



line 1:	
line 2:	
line 3:	

5. [28 points] More widgets, Edna! Store is an object with 3 attributes: name, widgets, and sister. A call of the form Store(n,w,s) creates a new Store object with attribute name set to n, widgets set to w, and sister set to s. Assume the class Store is accessible within the given code. Simulate running all the code. Draw the memory diagram as seen in class and Assignment 2. (As usual, do not draw the objects/folders for imported modules or for function definitions.) Pay attention to what goes in the Global Space / Call Stack / Heap.

```
def make_sale(n, s):
                                                       if s.widgets >= n:
                                                11
  def borrow(num, og):
                                                           s.widgets -= n
                                                12
      sister = og.sister
2
                                                           return True
      mine = og.widgets
3
                                                       elif s.sister is not None:
     hers = sister.widgets
                                                           return borrow(n, s)
      if mine + hers >= num:
                                                16
                                                       else:
          hers -= num - mine
6
                                                           return False
                                                17
          og.widgets = 0
          return True
                                                    s1 = Store("Getty's", 15, None)
      return False
9
                                                    s2 = Store("WonderPetz", 25, s1)
                                                    s1.sister = s2
                                                    success = make_sale(30, s1)
```

Global Space Call Stack Heap

- 6. **Object-**ively speaking. Objects of class Warehouse have 4 attributes:
 - pending [int list]: unique ids of packages not yet delivered to the warehouse
 - delivered [int list]: unique ids of packages that have been delivered to the warehouse
 - goal [int]: the number of deliveries this warehouse will try to complete
 - met_goal [boolean]: whether warehouse has delivered at least goal # of packages
 - (a) [8 points] Implement the following function according to its specification:

```
def prep_warehouse(w, g, n_ids, packages):
    Initializes Warehouse `w` attributes:
      - pending: set as a list of the last `n_ids` in the `packages` list
      - delivered: set as the empty list
      - goal: set to `g`
      - met_goal: set to False
    'packages' should not be modified
    Examples:
    prep_warehouse(w, 2, 3, [3,5,8,13,5,4]) --> w should have the attributes:
          pending is the list [13,5,4]
          delivered is the empty list
          goal is 2
         met_goal is False
    prep_warehouse(w, 7, 1, [1,9,4,3]) --> w should have the attributes:
          pending is the list [3]
          delivered is the empty list
          goal is 7
          met_goal is False
    Preconditions: n_ids: a positive integer
                   packages: an int list with >= n_ids elements
    11 11 11
    # STUDENTS: loops are NOT ALLOWED (or needed)
```

(b) [12 points] Implement the following function (which simulates the delivery of a package to the warehouse) according to its specification:

- (1) `id` might _not_ be on w's pending list.
 In this case, return False. (delivery failed)
- (2) `id` might _already_ be on w's delivered list.
 In this case, return False (deliver failed)
 If a delivery fails, none of w's attributes should be modified.

Preconditions:

```
w: a warehouse with attributes pending, delivered, goal, met_goal
id: a positive integer
```