Cornell NetID, all caps: _____

# CS 1110 Regular Final Solutions May 2021

**Skeletons/testing code available at**
https://www.cs.cornell.edu/courses/cs1110/2021sp/exams/prelim1/2021_spring_prelim1_testcode.py.

1. [5 points] **String processing.** At the bottom of the page are specifications for some string methods you can use for this question.

   Consider strings of the following format, where the number of spaces can vary (but you can assume at least one space immediately in front of and immediately behind each `>>`, `:` and `##`):

   ```
   tag  >> college1 : outcome1 ## college2 : outcome2 ## ...  ## enr : enrolledCollege
   ```

   Two example strings:

   ```
   3 >> OSU : Accepted ## Yale_U : Accepted ## BU : Accepted ## enr : BU
   ```

   and (notice the leading and extra spaces)

   ```
           3       >> OSU : Accepted ## Yale_U : Accepted ## ISU : Accepted ## enr : ISU
   ```

   Let variable `sline` store some string in the format above.

   Write roughly 1-5 lines of Python that store in variable `tag` the int that is the "tag" portion of `sline`.

   For either example above, the value stored in `tag` would be the **int** `3`, **not** the string `'3'`.

   Don't assume that the tag has only one digit.

   Multiple solutions are possible; here are two.

   ```
   tag = int(sline[:sline.index(">>")].strip())
   ```

   or

   ```
   tag = int(sline.split('>>')[0].strip())
   ```

   The function `int()` ignores leading and trailing spaces, so calling string method `strip()` was not, strictly speaking, necessary. We had originally had a point allocated to the call to `strip()` (or equivalent handling of extra spaces) in the grading guide, but removed it, bringing the exam to 118 points.

   ---

   Potentially useful string methods:

   `s.index(target)`: returns the index of the first (i.e., leftmost) occurrence of `target` in string `s`. Raises an error if `target` isn't in `s`.

   `s.rindex(target)`: returns the index of the last (i.e., rightmost) occurrence of `target` in string `s`. Raises an error if `target` isn't in `s`.

   `s.split(splitter)`: Precondition: `s` is a string and `splitter` is a string. Returns a list of strings that are the parts of `s` that were separated by `splitter`. Example: `'a!b!c'.split('!')` returns `['a', 'b', 'c']`.

   `s.strip()`: Returns a version of `s` with leading and trailing (but not internal) spaces removed.

2. [9 points] **Nested lists.** Implement the following function.

```
def diminish_rows(matrix, thresholds):
    """ Preconditions (no need to assert these):
        matrix: non-empty list of non-empty lists of ints.
        The sublists all have the same length.
        thresholds: a list of ints, one for each row of `matrix`.

    Modifies `matrix` as follows:
    Letting row = matrix[i],
        Every value in row that is <= thresholds[i] stays the same.
        Every value in row that is > thresholds[i] is changed to threshold[i]

    Ex: matrix = [ [5], [10], [15], [3]]
        thresholds = [6, 6, 6, 6]
        Then change matrix to   [[5], [6], [6], [3]]


    Ex: matrix = [ [1, 2, 3], [5, 2, 9], [-8, -1, 6], [2, 10, 6] ],
        thresholds = [2, 7, -3, 1]
        modified matrix = [ [1, 2, 2], [5, 2, 7], [-8, -3, -3], [1, 1, 1] ]"""

    for row in range(len(matrix)):
        thresh = thresholds[row]
        for col in range(len(matrix[0])): # all rows have same length
            if matrix[row][col] > thresh:
                matrix[row][col] = thresh
```

Alternate "while-like" solution:

```
    irow = 0 # needed to find right entry in thresholds
    for row in matrix:
        thresh = thresholds[irow]
        for icol in range(len(row)):
            if row[icol] > thresh:
                row[icol] = thresh
        irow += 1
```

Notes:

1. It was necessary to make sure to match the index of each item `thresholds` to the index of the corresponding row in `matrix`. Some solutions were incorrect because the lost track of whether they were indexing the rows or the columns.
2. The specification of a function is not executed. So, one could not assume that variables `i` or `row` existed.

3. [16 points] **While Loops and Linear Search**

Consider the specification of the class `WishItem` and its initializer. Do not implement the initializer, just know from its specification how to create a new `WishItem` instance.

```
class WishItem:
    """ An instance represents an item in a person's wish list at a shop

    Instance attributes:
        code [non-empty str]: unique string identifying the product wanted
        num_units [positive int]: number of units of the product wanted """

    def __init__(self, p_code, p_num):
        """Creates a new WishItem with attributes set as follows:
            code: set to `p_code`, where `p_code` is a non-empty string
            num_units: set to `p_num`, where `p_num` is a positive int """
```

Implement the following function, making effective use of a while loop. Your solution *must use a while loop* to receive points.

```
def add_to_wish_list(wlist, prod, u):
    """ Modify `wlist`, a list of WishItems, by adding `u` units of the product
    with the product code `prod`.  If the product is already in the list, increase
    the number of units wanted for that product by `u`.  If the product is not in
    the list, update `wlist` to include a new WishItem with `u` units of that
    product.  This function modifies the list; it does not create a new list.

    Preconditions (no need to assert these):
        wlist: a list of WishItems or an empty list. The elements in wlist have
                distinct product codes.
        prod: a non-empty string identifying the product wanted
        u: the number of units of the product to be added, a positive int"""
```

Notes:

- A statement like `if prod in wlist` will not work as (presumably) intended. `prod` is a string, whereas the items in `wlist` are WishItems, according to the preconditions; so `prod` is never `in wlist`.

```
k= 0
while k < len(wlist)  and  wlist[k].code != prod:
    k= k + 1

if k==len(wlist):
    wlist.append(WishItem(prod, u))
else:
    wlist[k].num_units += u
```

Alternate solution with sentinel variable in loop guard

```
        k= 0
        found= False
        while k < len(wlist) and not found:
            if wlist[k].code == prod:
                wlist[k].num_units += u
                found= True
            else:
                k= k + 1

        if not found:
            wlist.append(WishItem(prod, u))
```

4. **Testing and Debugging**. The function `date_of_birth` is meant to return a string representing a birthday. But there are multiple bugs in the code below. Read the specifications carefully; then, on the next page, identify and fix the bugs.

```
def date_of_birth(name, m, d, y):
    """
    Returns date of birth with month `m`, day `d` and year `y`
    of a person with name `name`.

    Examples:
        date_of_birth('Jiwon', 9, 13, 2000)
        --> "Jiwon was born on Sept. 13th, 2000!"

        date_of_birth('Rene Descartes', 3, 31, 1596)
        --> "Rene Descartes was born on Mar. 31st, 1596!"

    Preconditions:
        name: string.
        m: int. 1 <= m <= 12
        d: int. 1 <= d <= 31
        y: int. 1 <= y <= 2022
    """
    month = get_month(m)
    day = get_day(d)
    year = y

    date_of_birth = month + " " + day + ", " + year

    message = name + " was born on " + date_of_birth + "!"
    return message

def get_month(m):
    """Returns month name for month number `m`.
    The 1st month name is "Jan."
    """
    month_list = ["Jan.", "Feb.", "Mar.", "Apr.", "May",
    "June", "July", "Aug.", "Sept.", "Oct.", "Nov.", "Dec."]

    month = month_list[m]
    return month
```

```
def get_day(d):
    """
    Returns a string representation of day `d`
    with appropriate suffixes using the followi

    1. Days 11, 12, 13: use "th".
    2. Any other days ending in 1,2,3: use "st"
    3. Any other remaining days: use "th".
```

```
46
47      Examples:
48          get_day(11) --> "11th"
49          get_day(1) --> "1st"
50          get_day(28) --> "28th"
51
52      """
53      ending = d % 10
54
55      if d in [11, 12, 13]:
56          day = day_th(d)
57      elif ending <= 3:
58          day = day_st_nd_rd(d)
59      else:
60          day = day_th(d)
61
62      return day
63
64  def day_st_nd_rd(d):
65      """
66      precondition: d: int, where last digit
67        is between 1 and 3 inclusive.
68      """
69      ending = d % 10
70
71      if ending == 1:
72          suffix = "st"
73      elif ending == 2:
74          suffix = "nd"
75      else:
76          suffix = "rd"
77
78      day = str(d) + suffix
79      return day
80
81  def day_th(d):
82      day = str(d) + "th"
83      return day
```

(a) [5 points] First Bug: Consider the following call to `date_of_birth` and the Python error it triggers.

```
>>> date_of_birth('Jiwon', 9, 13, 2000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "birthday.py", line 23, in date_of_birth
    date_of_birth = month + " " + day + ", " + year
TypeError: can only concatenate str (not "int") to str
```

Below, explain where (a single line number) and why this error is triggered. **And**, write below the correct version of the line.

Need to convert year to str before concatenating. line 21: `year = str(y)`
or line 23: `date_of_birth = month` + " " + day + ", " + str(y)+ or
line 23: `date_of_birth = month` + " " + day + ", " + str(year)+

(b) [5 points] Second Bug: After the first bug (above) is fixed, the call

```
>>> date_of_birth('Jiwon', 9, 13, 2000)
```

should return the following string:

```
"Jiwon was born on Sept. 13th, 2000!"
```

Instead, it returns

```
"Jiwon was born on Oct. 13th, 2000!"
```

Below, explain where (a single line number) and why this problem is triggered. **And**, write below the correct version of the line.

indexing mistake for getting the correct month name (off by 1).
line 35: `month = month_list[m - 1]` or, line 19: `month = get_month(m-1)`

(c) [5 points] Third Bug: After the two bugs above are fixed, the call

```
>>> date_of_birth('Jason', 12, 20, 1988)
```

should return the following string:

```
'Jason was born on Dec. 20th, 1988!'
```

but instead returns

```
'Jason was born on Dec. 20rd, 1988!'
```

Below, explain where (a single line number) and why this problem is triggered. **And**, write below the correct version of the line.

Line 57 incorrectly includes days ending in 0 (e.g. day 20). Change line 57 to `elif 1 <= ending <= 3:` or `elif 1 <= ending and ending <=3:` Student answer that attempts to handle days ending in 0 in `day_st_nd_rd` is incorrect, because the precondition of day_st_nd_rd requires that the last digit be `d: 1 <= d <= 3.`

5. [20 points] **Memory Model**

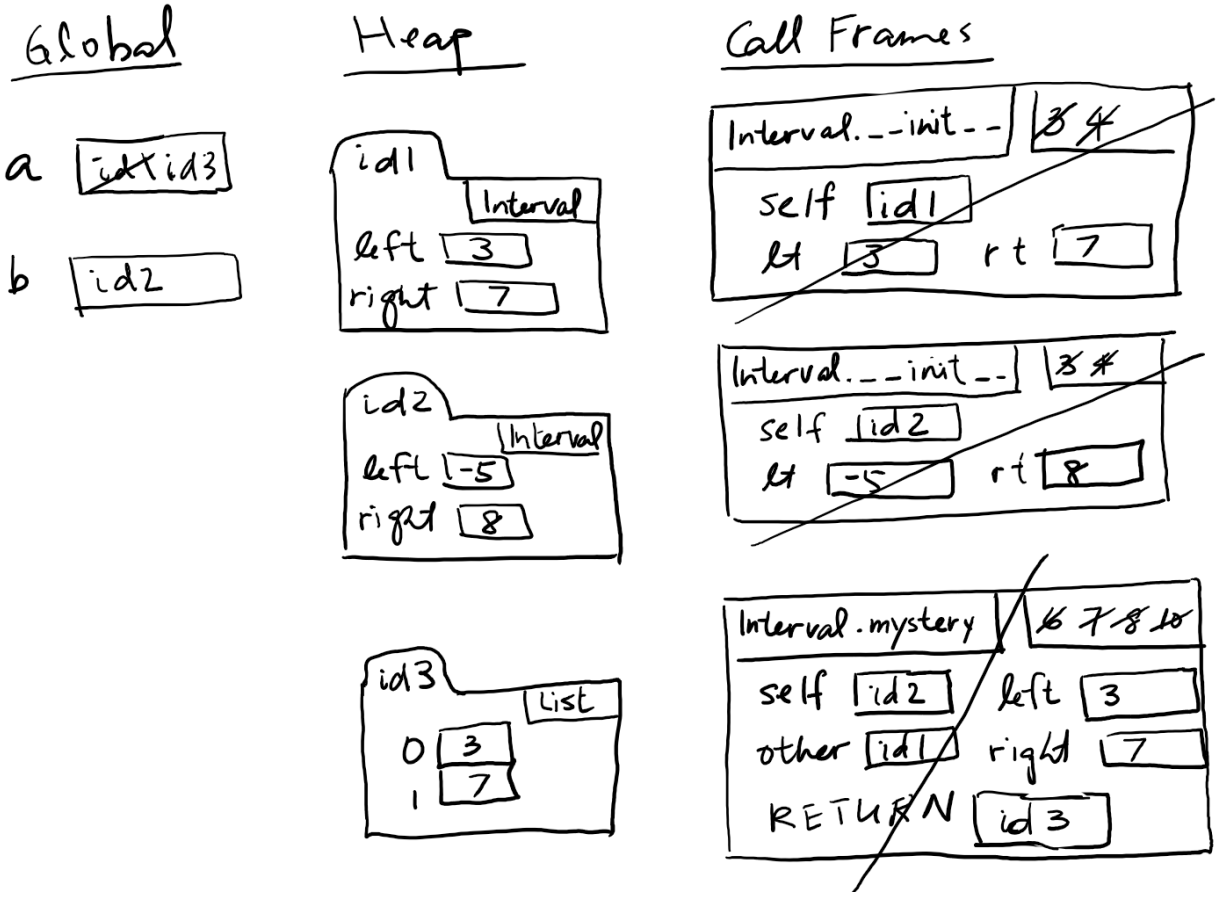
Execute the following script and draw the call frames, the heap space, and the global space. Do not draw any class folders, but don't forget to draw the call frames for `__init__`. Call frames for methods should follow the same conventions as call frames for other functions, except also indicate the name of the class next to the name of the method.

```
class Interval:

    def __init__(self, lt, rt):
        self.left= lt
        self.right= rt

    def mystery(self, other):
        left= max(self.left, other.left)
        right= min(self.right, other.right)
        if right <= left:
            return None
        return [left, right]

a= Interval(3,7)
b= Interval(-5,8)
a= b.mystery(a)
```




Notes:

- Since the instructions said to include the class name by the name of a method in the call frame, it was counted as incorrect to omit the word "Interval" in the method call-frame

top-left corner.

- It was not counted as an error to include the line for a method header in the program counter for the corresponding call frame.

6. [16 points] **Recursion.**

Let Employee be a class whose objects have the following two attributes:

```
name [str] - unique non-empty name of employee
employees [list of Employee] - employees reporting directly to this employee.
   *** LENGTH IS AT MOST 2 ***. (The length can be 0.)
```

Implement the following **function** (*not* a method), making effective use of recursion. For-loops are not required, although they are allowed as long as your solution is fundamentally recursive.

```
def get_tree(person):
    """Returns list of names of ALL employees that are *underneath*
    `person` (at any level). Order of list items does not matter; duplicates OK.

    Example:
    engineer = Employee('E', [])      means get_tree(engineer) --> []
    cto = Employee('T', [engineer])   means get_tree(cto) --> ['E']
    coo = Employee('O', [])           means get_tree(coo) --> []
    ceo = Employee('CEO', [coo, cto]) means get_tree(cea) --> ['O','T','E']

    Precondition: `person` is an Employee."""
```

Note: the original final exam was incompletely edited, so that there was some text that indicated that `get_tree` was a method, and other text that indicated that it was a non-method function.

In the end, we did not deduct points for either kind of solution as long as the treatment (method vs non-method function) was consistent. Here is a solution assuming the function is not a method.

```
    out = []
    for sub in person.employees:
        out += [sub.name] + get_tree(sub)
    return out
```

Alternate solution

```
    if len(person.employees)==0:
        return []
    elif len(person.employees)==1:
        sub = person.employees[0]
        return get_tree(sub) + [sub.name]
    else:
        left = person.employees[0]
        right = person.employees[1]
        return get_tree(left) + get_tree(right) + [right.name, left.name]
```

7. [15 points] **Classes.** This question simulates posting announcements on a course in Canvas: students can opt to additionally have announcements emailed to them.

   Here is the docstring for a new class Student.

```
class Student:
    """Instance attributes:
        netid [non-empty str]: netID, unique to this Student
        opted_in [bool]: Whether this Student should be emailed announcements.
        courses [list of Courses]: Courses this Student is enrolled in.
            Can be empty.
        inbox [list of strings]: messages for this Student.  Can be empty."""
```

   And here is the docstring a new class Course.

```
class Course:
    """Instance attributes:
        enrolled: List of unique Students enrolled in this Course.
                  Can be empty.
        announcements: possibly empty list of strings. """
```

   Implement the following *method* of class Course so that it meets its specification.

```
    def announce(self, a):
        """Adds `a` to this Course's announcements.
        Also adds `a` to the end of the inbox of every Student enrolled in this
            Course that has opted in.
            (Does *not* do so for Students that did *not* opt in.)

        Returns: a list of the netids of the Students that had the announcement
            put in their inbox.

        Precondition: `a` is a nonempty string."""

        self.announcements.append(a)
        emailed = []
        for s in self.enrolled:
            if s.opted_in:
                s.inbox.append(a)
                emailed.append(s.netid)
        return emailed
```

8. **Classes and subclasses.**

   (a) [15 points] Suppose the following code (both columns) were executed.

```
6  class A:
7      num_As = 0
8
9      def __init__(self):
10         A.num_As += 1
11
12     def __str__(self):
13         # Assume a reasonable format for type()
14         return "I am type " + str(type(self))
15
16 class B(A):
17     pass
18     def __str__(self):
19         return "I am a B"
20
```

```
21 class C(A):
22     num_Cs = 0
23     def __init__(self):
24         super().__init__()
25         self.num_Cs = 23
26
27 a1 = A()
28 a2 = A()
29 a3 = A()
30 # b = B()
31 c1 = C()
32 c2 = C()
33 print(A.num_As)
34 print(C.num_Cs)
35 print(c2.num_Cs)
36 print(c2)
```

1. What is the output of line 33? (Write "Error" if an error would occur.) 5
2. What is the output of line 34? (Write "Error" if an error would occur.) 0
3. What is the output of line 35? (Write "Error" if an error would occur.) 23
4. What is the output of line 36? (Write "Error" if an error would occur.) I am type C (or something similar. But NOT "I am type A")
5. True or False? Explain your answer in 1-2 sentences; no credit without explanation.

   Class B must have a definition of an `__init__` method; otherwise, if line 30 were uncommented, an error would occur.

   False; class B inherits the initializer of its superclass.

(b) [6 points] Consider the following code.

```
1 class Z:
2     def __init__(self,n):
3         self.label = n
4 z = Z("first Z object")
```

For each statement below, write whether it is "True", "False", or "Not enough info to decide". No other explanation needed.

Note that the line numbers were meant to indicate that there was no code "before" this code, so, for instance, there would not be a global variable `label` available.

1. Changing line 3 to "`self.n = label`" would *not* cause an error when the code is executed. False
2. Changing line 3 to "`self.n = label`" would cause an error when the code is executed, because `label` is not defined. True

3. Changing line 3 to "`self.n = label`" would cause an error when the code is executed, because `n` is not defined. False
4. Changing line 3 to "`self.n = n`" would cause an error when the code is executed, because `label` is not defined. False

9. [1 point] **Fill in your Cornell NetID (ex: LJL2, *not* your Cornell ID number) at the top of each page.**

   **Also, don't discuss this exam with students who are scheduled to take a later makeup.**

HAVE A GREAT SUMMER!