CSCA08 Fall 2015 Final Exam
Duration — 180 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|

Markus Login: _____

Last Name: _____ **First Name:** _____

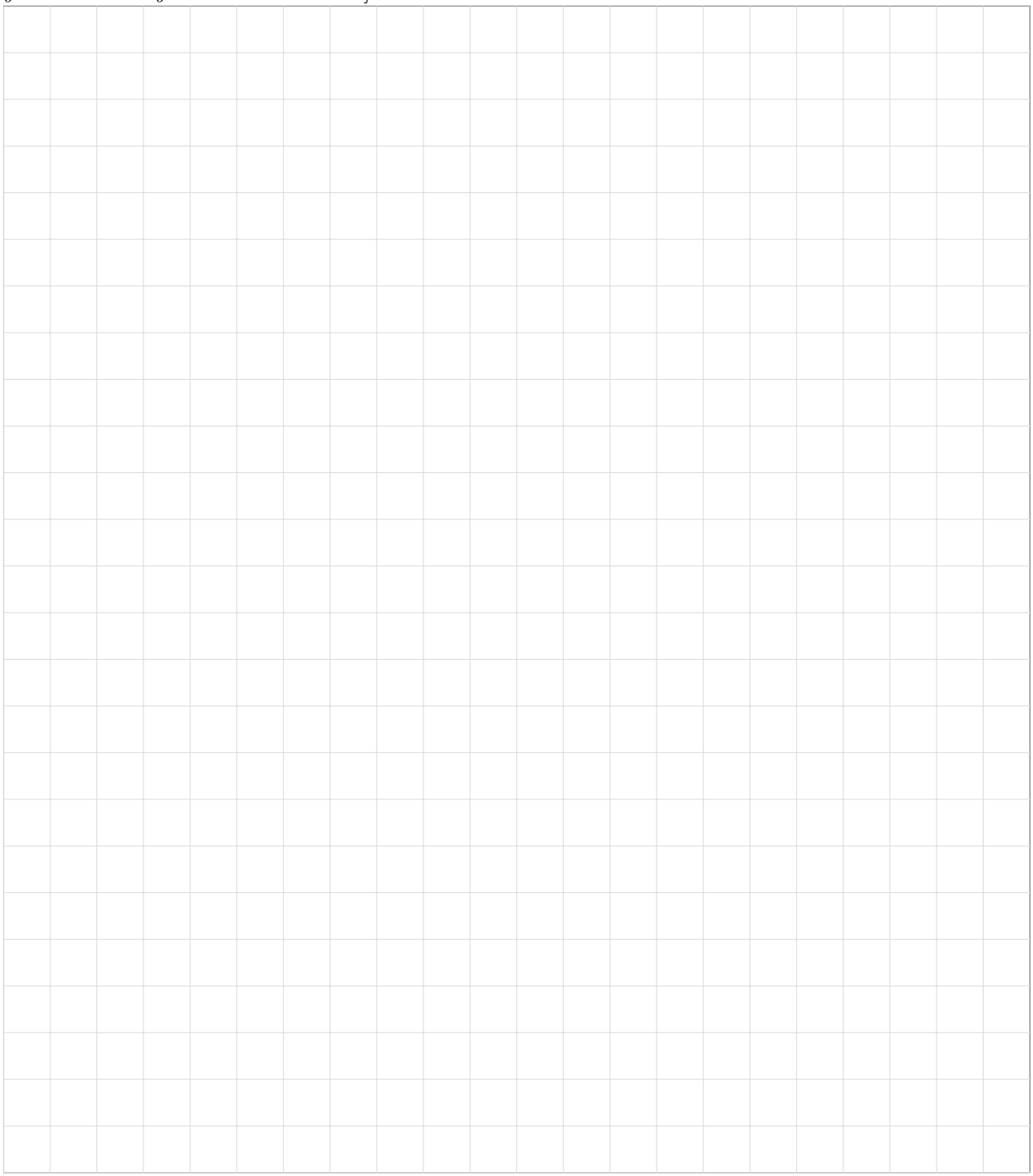
*Do **not** turn this page until you have received the signal to start.*

This exam consists of 5 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*
Proper documentation is required for all functions and code blocks. If you use any space for rough work, indicate clearly what you want marked. Any pages not attached to this cover page will not be marked. Please read all questions thoroughly before starting on any work.

The University of Toronto's Code of Behaviour on Academic Matters applies to all University of Toronto Scarborough students. The Code prohibits all forms of academic dishonesty including, but not limited to, cheating, plagiarism, and the use of unauthorized aids. Students violating the Code may be subject to penalties up to and including suspension or expulsion from the University.

# 1:	_____/ 5
# 2:	_____/ 5
# 3:	_____/10
# 4:	_____/10
# 5:	_____/20
TOTAL:	_____/50

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

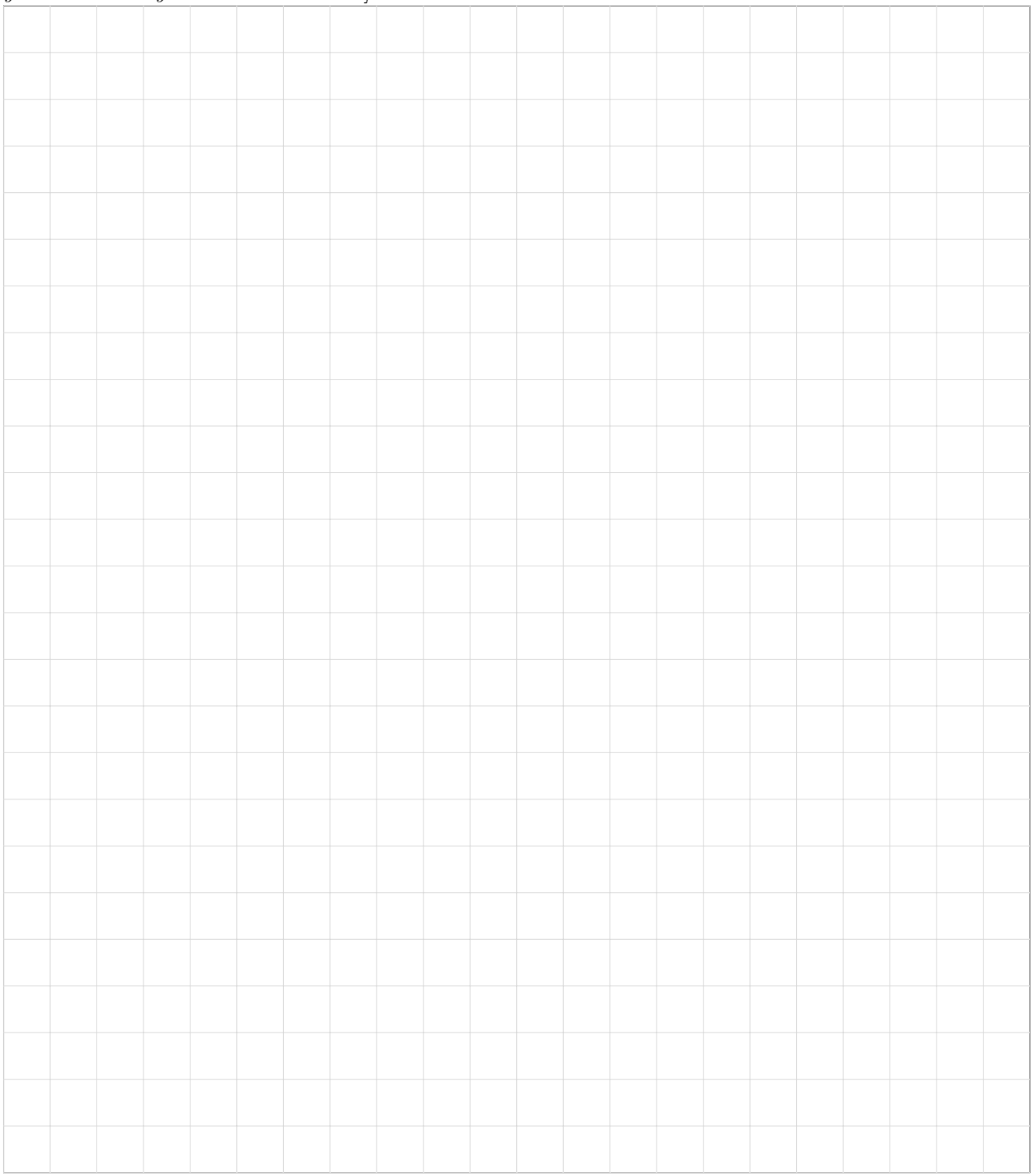


Question 1. [5 MARKS]

Write the output of the following code in the space provided.

```
class GrandParent():
    def __init__(self, a, b):
        self._a = a
        self._b = b

    def blah(self):
        return "GP:" + self._a + self._b

class Parent1(GrandParent):
    def __init__(self, a, b, c):
        GrandParent.__init__(self, a, b)
        self._c = c

    def blah(self):
        return ("P1:" + self._a +
                self._b + self._c)

class Parent2(GrandParent):
    def __init__(self, a, b, c):
        self._a = b
        self._b = a
        self._c = self.blah()

class Child1(Parent2):
    def __init__(self, a, b, c, d):
        Parent1.__init__(self, a, b, c)
        self._d = d

class Child2(Parent1):
    def __init__(self, a, b, c, d):
        Child1.__init__(self, b, c, d, a)
        Parent2.__init__(self, a, b, c)


    def blah(self):
        return ("C2: " + self._a + self._b
                + self._c + self._d)

gp = GrandParent("A", "B")
print(gp.blah())
p1 = Parent1("A", "B", "C")
print(p1.blah())
p2 = Parent2("A", "B", "C")
print(p2.blah())
c1 = Child1("A", "B", "C", "D")
print(c1.blah())
c2 = Child2("A", "B", "C", "D")
print(c2.blah())
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

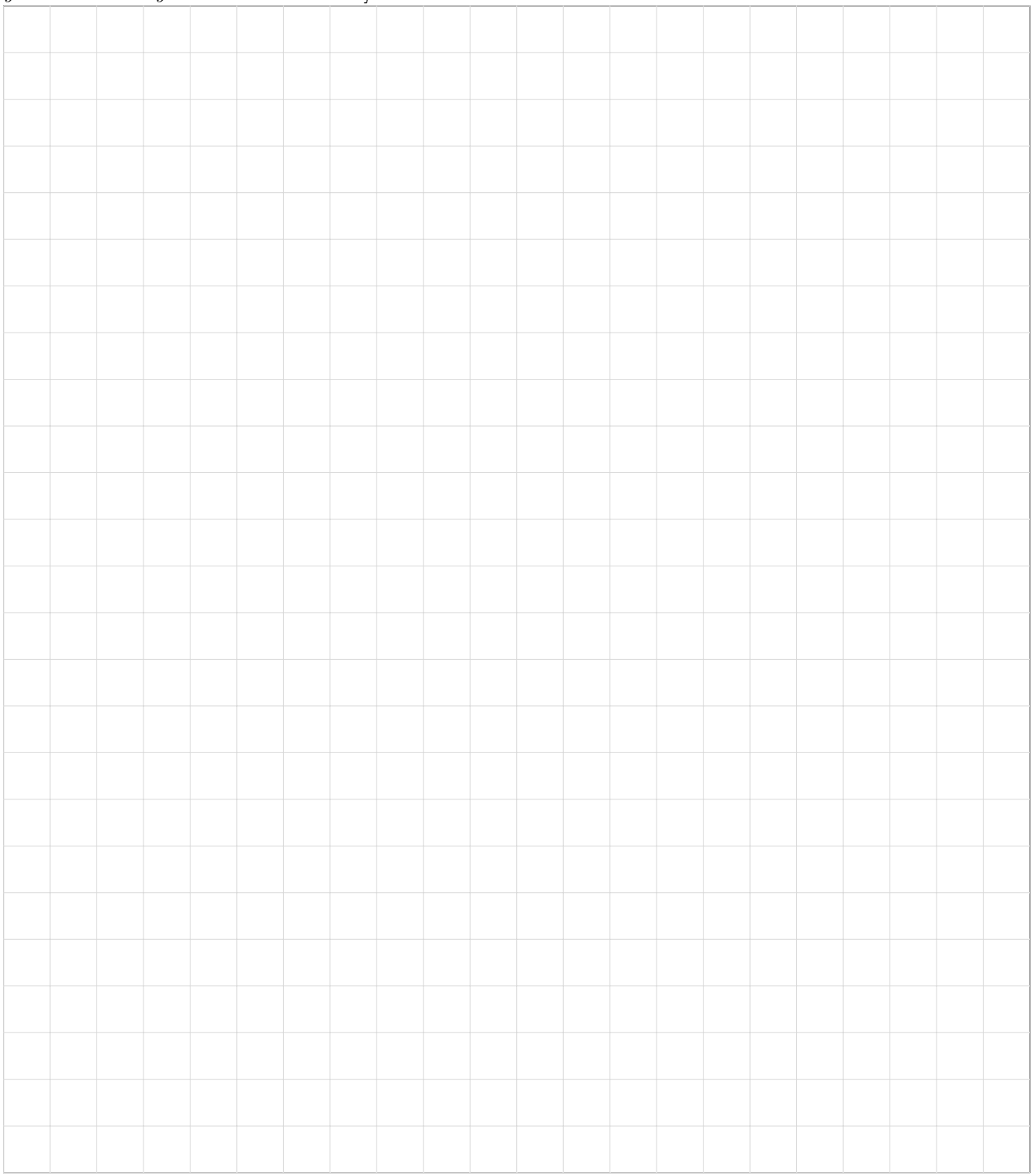


Question 2. [5 MARKS]

Write the output of the following code in the space provided. Assume that dictionaries print their keys in alphabetical/numerical order

```
s1 = "CSCA08 2015"
s2 = "INTRO TO COMPUTER SCIENCE"
d1 = {}
d2 = {}
i = 0
j = len(s2) - 1
while(i < j):
    d1[s1[i]] = s2[j]
    if(s1[i] in s2):
        d2[i] = s2[j-i]
    i += 1
    j -= i
print(d1)
print(d2)
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Question 3. [10 MARKS]

Write the output of the following code in the space provided [1].

```
def func1(list1):
    list1[0] = "A"
    print(list1)

def func2(list1, list2):
    list1[0] = list2[0]
    list3 = list1[1:3]
    list3[0] = "B"
    list2[0] = list3[0]
    print(list1, list2, list3)

def func3(list1, list2):
    list3 = list1[:]
    list1[0] = list2
    list1[0][0] = "C"
    list2[1] = "D"
    print(list1, list2, list3)

list1 = [1, 2, 3]
list2 = [4, 5, 6]
func1(list1)
func2(list1,list2)
list1 = [1, 2, 3]
list2 = [4, 5, 6]
func3(list1, list2)
list1 = [[1, 2], 3]
list2 = [[4], [5, [6]]]
func3(list1, list2)
```

[1] Boy, this sure does look familiar... at least this time we can't mis-read any lists as integers. Good thing we learn from our mistakes!

MANGLED CODE:

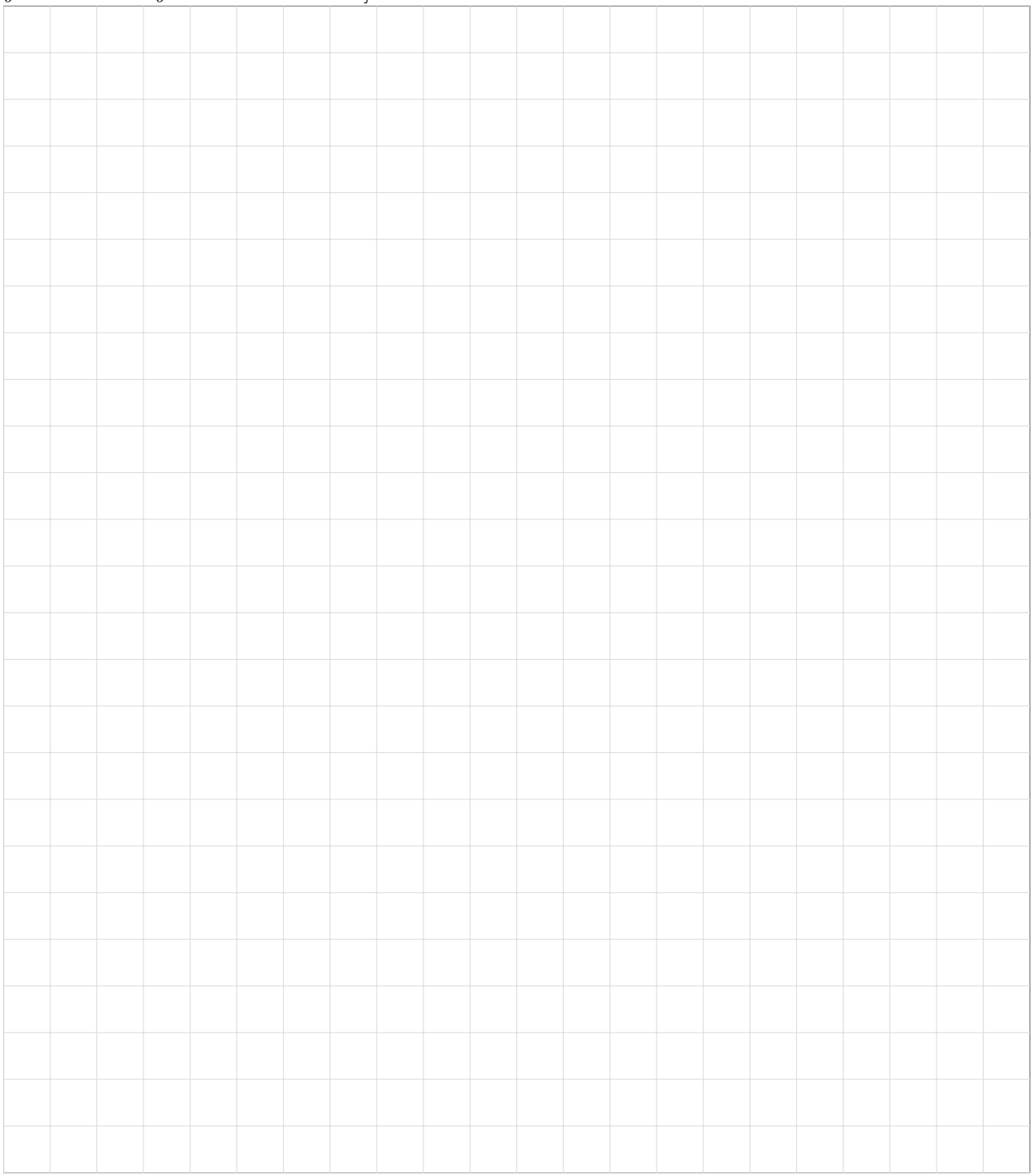
```
BICYCLE_RIDERS = 1
BICYCLE_WHEELS = 2
CAR_WHEELS = 4
SPORTS_CAR_SEATS=2
SPORTS_CAR_MAX_SPEED=300
SPORTS_CAR_MAX_SPEED
SPORTS_CAR_SEATS
BICYCLE_RIDERS
BICYCLE_WHEELS
CAR_WHEELS
Bicycle
Car
SportsCar
Vehicle
max_passengers
max_speed
num_seats
num_wheels
max_speed
self
self._max_passengers = max_passengers
self._max_passengers = num_seats
self._max_passengers = SPORTS_CAR_SEATS
self._max_passengers = BICYCLE_RIDERS
self._num_wheels = num_wheels
self._num_wheels = CAR_WHEELS
self._num_wheels = BICYCLE_WHEELS
self._max_speed = max_speed
self._max_speed = SPORTS_CAR_MAX_SPEED
Vehicle.__init__(               )
Car.__init__(              )
Bicycle.__init__(               )
SportsCar.__init__(               )
```

Question 4. [10 MARKS]

Nick was building some classes to represent various modes of transportation. He managed to get as far as writing all of the initializers, but then he stepped out of his office for just a minute... and guess who struck? The **Code Mangler**! This time, the code mangler didn't just swap the lines of code around, he/she took out the various parameters, but left the structure intact. Duplicate lines were deleted, and it looks as though a few extra lines of code may have even been placed in there to further confuse matters. Nick has copies of the DocStrings(somewhere), so no need to re-write those. Please re-assemble Nick's code in the space below.

```
class Vehicle(          ):

    def __init__(                                          ):




class Car(          ):

    def __init__(                                          ):




class SportsCar(          ):

    def __init__(                                          ):




class Bicycle(          ):

    def __init__(                                          ):
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

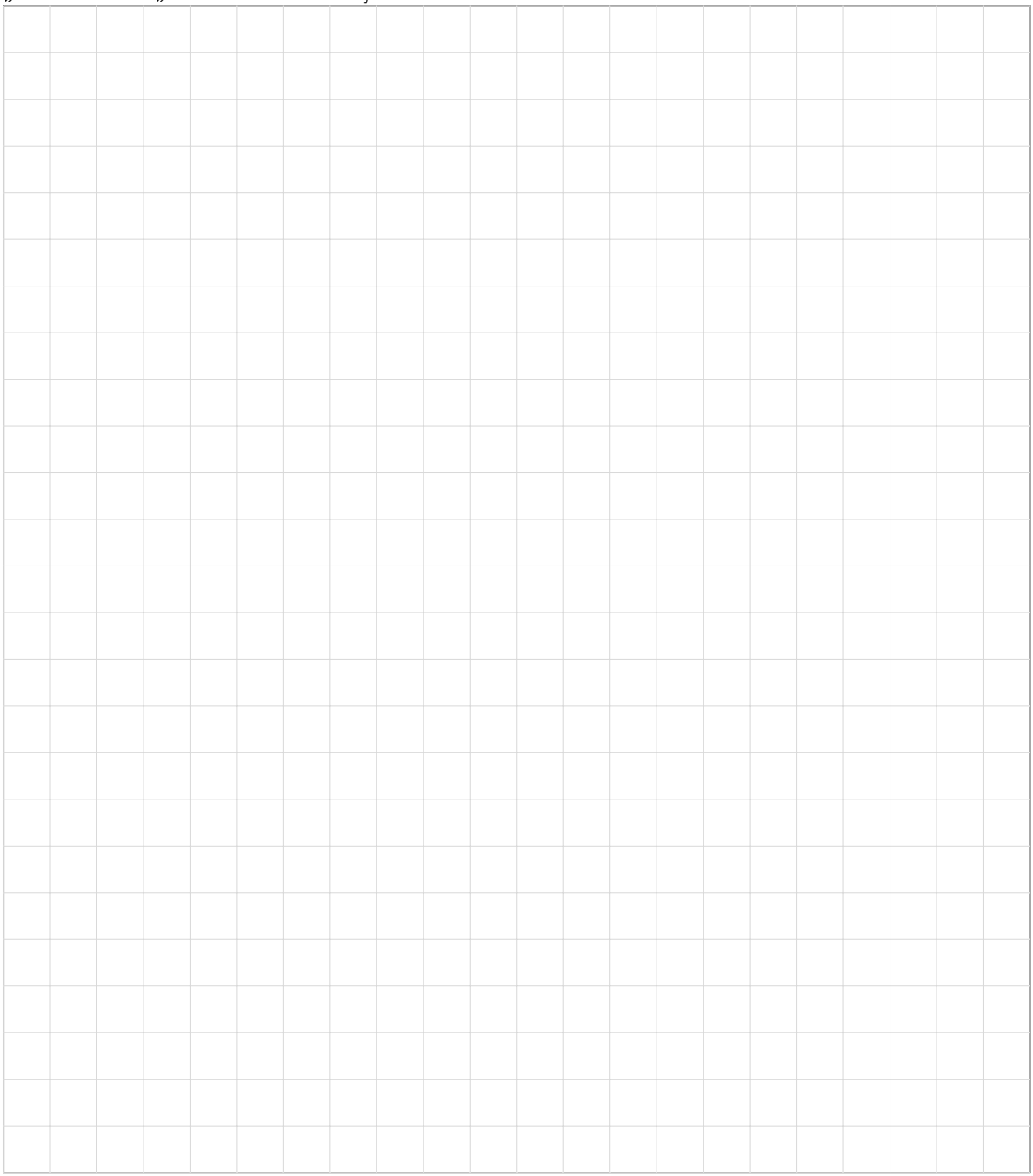


Question 5. [20 MARKS]

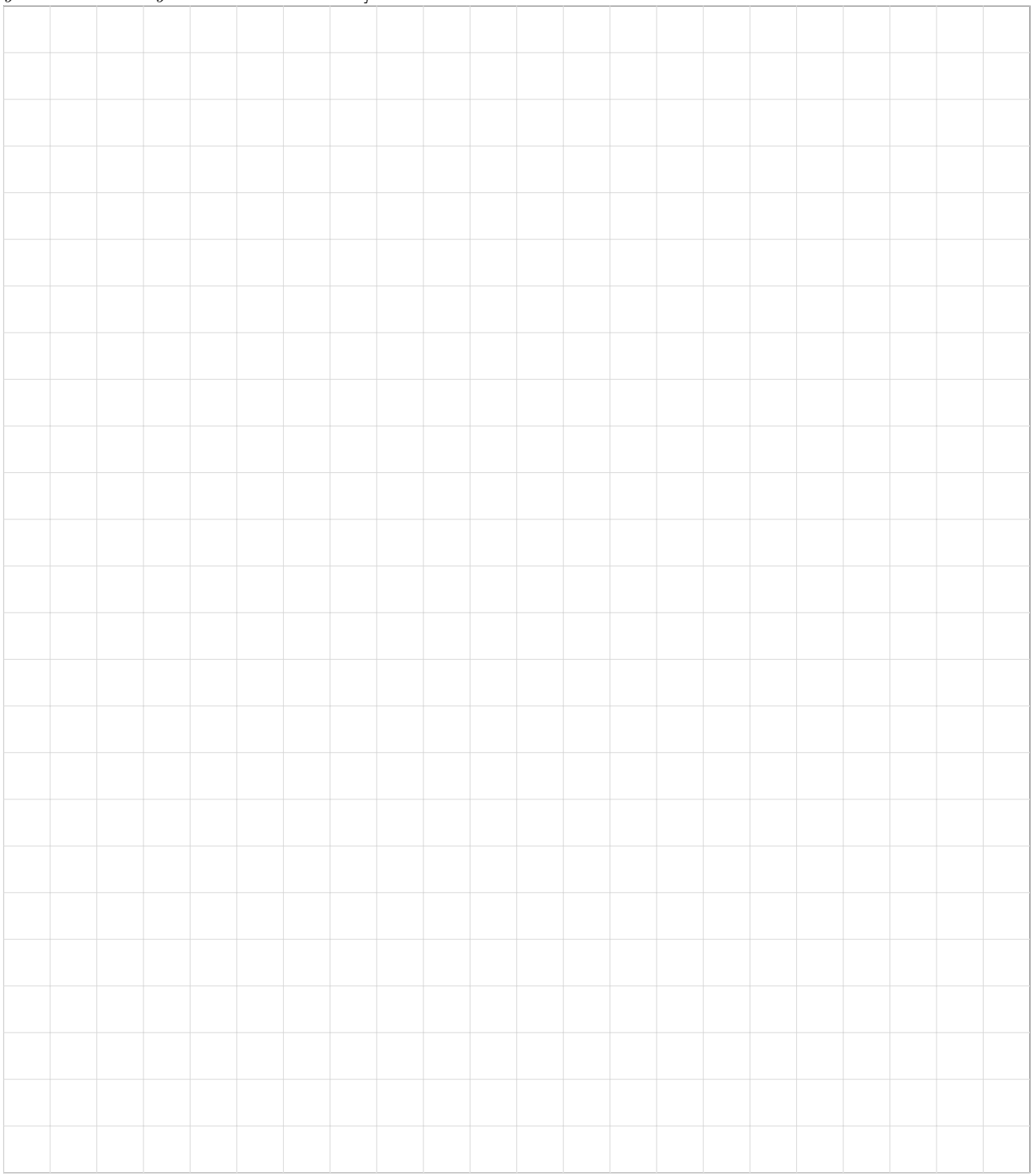
Santa needs help managing all of his operations at the North Pole. Your job is to write a program to simulate giving presents to all the good children. You will read 2 files, 'naughty.csv' and 'nice.csv'. The nice file contains records in the form of `name, present`, where name is the name of a child and present is the present that they want. The naughty file contains records of the form `name, infraction`, where name is the child's name and infraction is how they got on the naughty list. Santa should only give presents to the children who aren't on the naughty list. Santa's elves (interesting fact: all elves are named "Elife the Elf", and are eternally 6 years old) make the presents and put them into his sack, and Santa can then get the presents from his sack to give to the children.

Santa wants you to write a function called `give_presents`, that takes in a set of children, a sack of presents, and an elf who knows the info for Santa's naughty list, and gives presents to the children. When printed, children should say what they wanted for christmas, and what they got (or if they didn't get anything, they'll say they didn't get anything). You may assume that all children have unique names. You should then write some global code with a bunch of assert statements (they don't need to be inside proper UnitTests, just bare assertEqual statements will suffice) to test the function. If all goes well, Santa may want to add other features in future, so you should build all of your code in a way that it will be easy to expand upon later, and use good OOP principles to help your code interface with Santa's other systems.

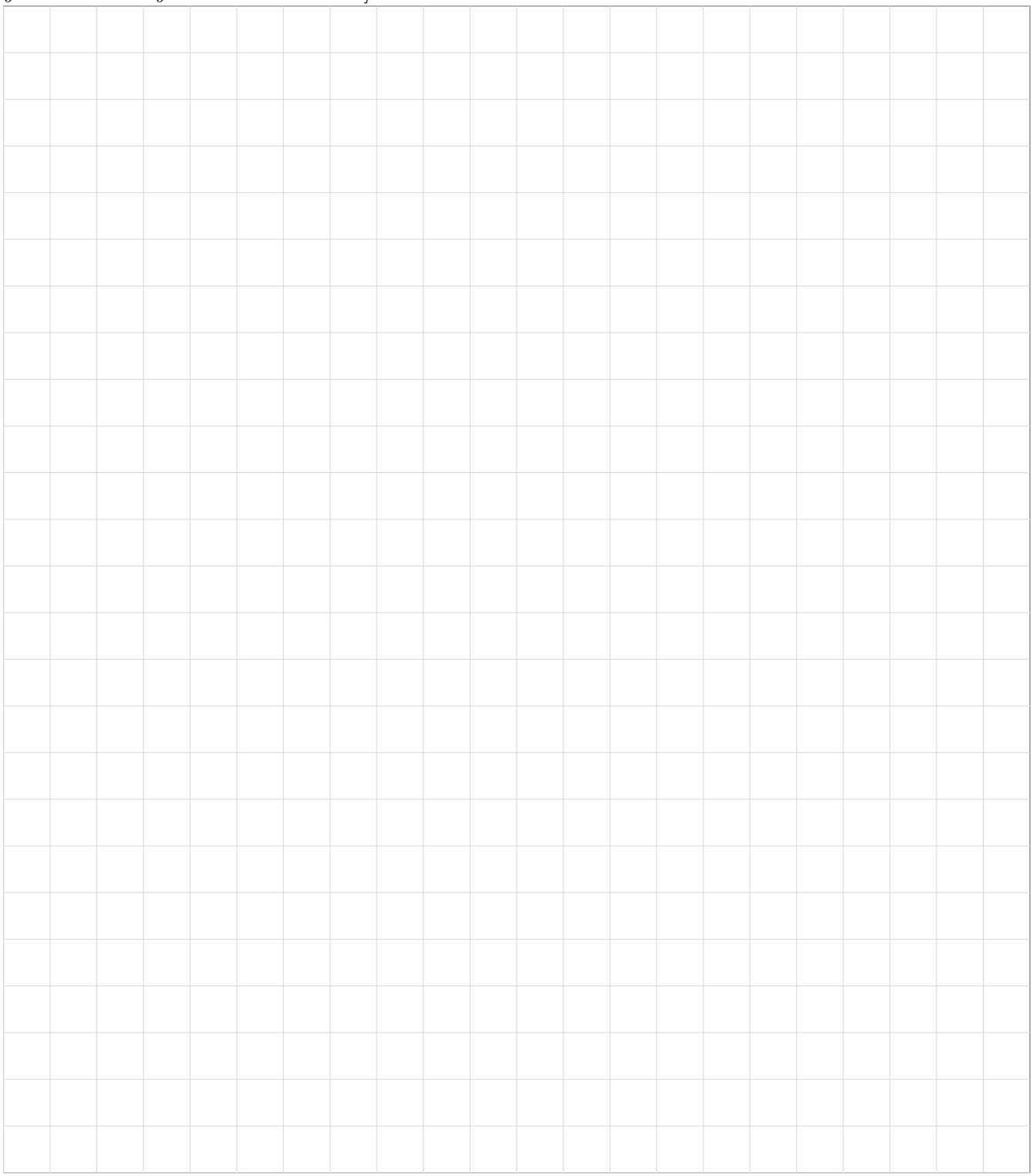
[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Short Python function/method descriptions:

You may tear this page off, but if you do so, you must not include any work on it (front or back) that you wish to have marked.

```
__builtins__:
  abs(number) -> number
    Return the absolute value of the given number.
  max(a, b, c, ...) -> value
    With two or more arguments, return the largest argument.
  min(a, b, c, ...) -> value
    With two or more arguments, return the smallest argument.
  isinstance(object, class-or-type-or-tuple) -> bool
    Return whether an object is an instance of a class or of a subclass thereof.
    With a type as second argument, return whether that is the object's type.
  int(x) -> int
    Convert a string or number to an integer, if possible.  A floating point argument
    will be truncated towards zero.
  str(x) -> str
    Convert an object into a string representation.

str:
  S.count(sub[, start[, end]]) -> int
    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
    interpreted as in slice notation.
  S.find(sub[,i]) -> int
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.isalpha() --> bool
    Return True if and only if all characters in S are alphabetic
    and there is at least one character in S.
  S.isdigit() --> bool
    Return True if and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() --> bool
    Return True if and only if all cased characters in S are lowercase
    and there is at least one cased character in S.
  S.isupper() --> bool
    Return True if and only if all cased characters in S are uppercase
    and there is at least one cased character in S.
  S.lower() --> str
    Return a copy of S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.split([sep]) -> list of str
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.startswith(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  S.strip() --> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() --> str
    Return a copy of S converted to uppercase.
```

```
list:
  append(...)
    L.append(object) -- append object to end
  count(...)
    L.count(value) -> integer -- return number of occurrences of value
  index(...)
    L.index(value, [start, [stop]]) -> integer -- return first index of value.
    Raises ValueError if the value is not present.
  insert(...)
    L.insert(index, object) -- insert object before index
  pop(...)
    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
  remove(...)
    L.remove(value) -- remove first occurrence of value.
    Raises ValueError if the value is not present.

math:
   ceil(...)
        Return the ceiling of x as an int.
        This is the smallest integral value >= x.
    cos(...)
        Return the cosine of x (measured in radians).
    floor(...)
         Return the floor of x as an int.
        This is the largest integral value <= x.
    pow(...)
        Return x**y (x to the power of y).
    sin(...)
        Return the sine of x (measured in radians).
    sqrt(...)
        Return the square root of x.
    tan(...)
        Return the tangent of x (measured in radians).
set:
  pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.
dict:
  keys(...)
    D.keys() -> a set-like object containing all of D's keys
  get(...)
     D.get(k[,d]) -> returns D[k] if k is in D, otherwise returns d.  d defaults to None.
object:
  __init__(...)
    x.__init__(...) initializes x; called automatically when a new object is created
  __str__(...)
    x.__str__() <==> str(x)
other:
   x // y = integer divide x by y (i.e., how many times does x divide evenly into y). 5 // 3 = 1
   x % y = the remainder when x is integer divided by y. 5 % 3 = 2
```