ICS 31 • UC Irvine
Fall 2015 • David G. Kay
********** KEY **********

Your Name _____
Your Student ID (8 digits) _____
Your UCInet ID _____

Your Lab Section (circle one):

1. 8:00a Prateek B. Swamy
2. 10:00 Prateek B. Swamy
3. 12:00 Rohit Malhotra
4. 2:00 Rohit Malhotra
5. 4:00 Roeland Singer-Heinze
6. 6:00 Roeland Singer-Heinze
7. 8:00p Akshat Amrish Patel
8. 8:00a Shibani Konchady
9. 10:00 Shibani Konchady
10. 12:00 Vignesh Raghunathan
11. 2:00 Vignesh Raghunathan
12. 4:00 Akshat Amrish Patel
13. 8:00a Kartic Saxena
14. 10:00 Kartic Saxena
15. 12:00 Archit Dey
16. 2:00 Archit Dey

Second Midterm

You have 75 minutes (until the end of the class period) to complete this exam. There are 60 points possible, so allow approximately one minute per point and you'll have plenty of time left over.

Please read all the problems carefully. If you have a question on what a problem means or what it calls for, ask us. Unless a problem specifically asks about errors, you should assume that each problem is correct and solvable; ask us if you believe otherwise.

In answering these questions, you may use any Python 3 features we have covered in class, in the text, in the lab assignments, or earlier on the exam, unless a problem says otherwise. Use more advanced features at your own risk; you must use them correctly. If a question asks for a single item (e.g., one word, identifier, or constant), supplying more than one will probably not receive credit.

Remember, stay cool! If you run into trouble on a problem, go on to the next one. Later on, you can go back if you have time. Don't let yourself get stuck on any one problem.

You may not share any information or materials with classmates during the exam and you may not use any electronic devices.

Please write your answers clearly and neatly—we can't give you credit if we can't decipher what you've written.

We'll give partial credit for partially correct answers, so writing something is better than writing nothing. But be sure to answer just what the question asks.

Good luck!

Problem 1 (9 points)	
Problem 2 (10 points)	
Problem 3 (5 points)	
Problem 4 (7 points)	
Problem 5 (5 points)	
Problem 6 (5 points)	
Problem 7 (19 points)	
Total (60 points)	

Problem 1 (9 points) **Topic: processing lists of namedtuples**

The Registrar has asked you to build a student enrollment system. You start by representing a Student object as

```
Student = namedtuple('Student', 'ID name level major')
```

where all four fields are strings: the student's ID number, name, class level (FR, SO, JR, SR, GR), and major. You create for testing these sample Students:

```
pp1 = Student('11112222', 'Programmer, Paula', 'FR', 'CS')
pp2 = Student('22223333', 'Programmer, Peter', 'SR', 'FILM')
aa1 = Student('33334444', 'Anteater, Andrea', 'SR', 'CS')
aa2 = Student('55556666', 'Aardvark, Aaron', 'SR', 'BIO')

SL = [pp1, pp2, aa1, aa2]
```

(a) (5 points) Complete the definition of the function below, consistent with its header, docstring comment, and assertions, by filling each blank with exactly one identifier, operator, or constant.

```
def collect_by_level (L: 'list of Student', lev: str) -> 'list of Student':
    ''' Return a list of all Students in L with the specified class level
        (FR, SO, JR, SR, or GRAD)
    '''
    result = []

    for s _____________  _____________:          # in L

        if s._____________ == _____________:    # level   lev

            result.append(_____________)        # s

    return _____________                        # result

assert collect_by_level(SL, 'SR') == [pp2, aa1, aa2]
assert collect_by_level(SL, 'JR') == []
assert collect_by_level(SL, 'FR') == [pp1]
```

(b) (4 points) Complete the definition of the function below, consistent with its header, docstring comment, and assertions, by filling each blank with exactly one identifier, operator, or constant.

```
def collect_majors (L: 'list of Student', ML: 'list of str') -> 'list of Student':
    ''' Return a list of all Students in L with a major on the list ML
    '''

    ### The body of this function is the same as the body of collect_by_level above,
    ### except for the following line; fill in the blanks.

        if _____________ . _____________  _____________  _____________:
             #     s                major             in            ML
    ### The rest of the body is the same as above.
```

The key here is using the "in" operator to look something up in a list

Problem 2 (10 points) **Topic: Expressions with lists and namedtuples**

Continuing your development of the student enrollment system, you represent each course with

```
Course = namedtuple('Course', 'dept num title units instr cap roster waitlist')
```

where the department, number, and title are strings, the number of units is an int, the instructor is a string, the maximum capacity of the course is an int, and both the roster and the waiting list are lists of Students.

```
ics31 = Course('ICS', '31', 'Intro Programming', 4, 'Kay, David', 350, [pp1, aa2], [])
infx269 = Course('Infx', '269', 'Computer Law', 4, 'Kay, David', 3, [pp1, pp2, aa2], [aa1])
ics32 = Course('ICS', '32', 'Software Libraries', 4, 'Thornton, Alex', 500, [pp2, aa1], [])
econ20a = Course('Economics', '20A', 'Intro Econ', 4, 'Chalfant, Jim', 8,
     [pp1, pp2, aa1, aa2], [])

Classes = [ics31, infx269, ics32, econ20a]
```

Use the above definitions in this problem. [Note that this is not the same organization as we used in class and the lab.]

(a) (5 points) Below are 10 Python expressions. Indicate the data type of each expression by checking the appropriate box.

(a.1) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`ics31` **== Course**

(a.2) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`infx269.waitlist` **== list # of Student**

(a.3) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`len(ics31.waitlist) == 0` **== bool**

(a.4) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`Classes[0].cap` **== int**

(a.5) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`infx269.num` **== str**

(a.6) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`Classes[1]` **== Course**

(a.7) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`Classes[1].roster` **== list # of Student**

(a.8) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`Classes[1].roster[2].major` **== str**

(a.9) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`len(pp1.name)` **== int**

(a.10) ❑int ❑float ❑bool ❑str ❑function ❑Student ❑Course ❑list of Student ❑list of Course
`Classes[0].roster[1]` **== Student**

(b) (5 points) Give the *value* of each of these expressions, based on the definitions above. Remember zero-based indexing.

```
ics31.instr == 'David Kay'
```
== False

```
len(Classes[3].dept)
```
== 9

```
Classes[0].roster[1].name
```
== 'Aardvark,Aaron' # (credit with or without quotes)

```
Classes[2].waitlist
```
== [ ] # or "Empty list""

```
Classes[1].roster[2].ID[-1]
```
== '6'

Problem 3 (5 points) Topic: Sorting lists

We have two functions that produce enrollment statistics for a course:

```
def percent_of_capacity(c: Course) -> float:
    ''' Return a measure of how full the class is, as a percentage of capacity
    '''
    return len(c.roster)/c.cap * 100
assert percent_of_capacity(infx269) == 100
assert percent_of_capacity(econ20a) == 50
assert percent_of_capacity(Course('d','n','t',4,'i', 10, [], [])) == 0


def relative_waitlist_size (c: Course) -> float:
    ''' Return the size of the Course's waitlist as a percentage of the capacity of
         the course.  (E.g., a course with capacity 10 and a 10-person waitlist size
         has a relative waitlist size of 100%)
    '''
    return len(c.waitlist)/c.cap * 100
assert relative_waitlist_size(Course('d','n','t',4,'i', 10,
                                     [aa1,aa1,aa1,aa1,aa1,aa1,aa1,aa1,aa1,aa1],
                                     [aa1,aa1,aa1,aa1,aa1,aa1,aa1,aa1])) == 80
assert relative_waitlist_size(econ20a) == 0
assert relative_waitlist_size(infx269) == 1/3 * 100
```

Which of the following code segments will priint the Courses in the list Classes, in order by percent of capacity, highest values first? Circle *one or more* of A, B, C, D, or E; one or more may be correct. Consider each code segment in isolation, not in the context of having executed any of the other segments previously.

A. **THIS**

```
for c in sorted(Classes, key=percent_of_capacity, reverse=True):
    print(c)
```

B. **[Problem here is that sort() is a freestanding method, not an expression)**

```
for c in Classes.sort(key=percent_of_capacity, reverse=True):
    print(c)
```

C. **THIS**

```
Classes.sort(key=percent_of_capacity, reverse=True)
for c in Classes:
    print(c)
```

[continued]

D. **THIS**

```
result = []
for c in sorted(Classes, key=percent_of_capacity, reverse=True):
    result += [c]
for c in result:
    print(c)
```

E. **[Problem here is that we print before we sort, so we don't see things in order]**

```
for c in Classes:
    print(c)
Classes.sort(key=percent_of_capacity, reverse=True)
```

Problem 4 (7 points) **Topic: String formatting**

From the previous problem, we have a list of Course objects named Classes.

(a) (4 points) Suppose we use this code to produce a table of enrollment figures:

```
print('Course      Cap  Enr % Full % Wait')
print('------      ---  --- ------ ------')
for c in Classes:
    print("{:4s} {:5s} {:4d} {:4d} {:5.1f}% {:5.1f}%".format(c.dept,
         c.num, c.cap, len(c.roster), percent_of_capacity(c),
         relative_waitlist_size(c)))
```

Which of the tables below is the actual output of this code? Circle *only one* of A, B, C, D, or E.

A. **# This. 4 pts. for this; that's it. The principle: A field value larger than specified format takes what it needs**

```
Course      Cap  Enr % Full % Wait
------      ---  --- ------ ------
ICS  31     350    2   0.6%   0.0%
Infx 269      3    3 100.0%  33.3%
ICS  32     500    2   0.4%   0.0%
Economics 20A      8    4  50.0%   0.0%
```

B.

```
Course      Cap  Enr % Full % Wait
------      ---  --- ------ ------
ICS  31     350    2     1%     0%
Infx 269      3    3   100%    33%
ICS  32     500    2     0%     0%
Econ 20A      8    4    50%     0%
```

C.

```
Course      Cap  Enr % Full % Wait
------      ---  --- ------ ------
ICS  31     350    2   0.6%   0.0%
Infx 269      3    3 100.0%  33.3%
ICS  32     500    2   0.4%   0.0%
Econ 20A      8    4  50.0%   0.0%
```

D.

```
Course           Cap  Enr % Full % Wait
------           ---  --- ------ ------
ICS  31          350    2   0.6%   0.0%
Infx 269           3    3 100.0%  33.3%
ICS  32          500    2   0.4%   0.0%
```

```
Economics 20A      8    4  50.0%   0.0%
```

[question continues on next page]

E.

```
Course           Cap  Enr % Full % Wait
------           ---  --- ------ ------
ICS       31     350    2   0.6%   0.0%
Infx      26       3    3 100.0%  33.3%
ICS       3      500    2   0.4%   0.0%
Economics 20A      8    4  50.0%   0.0%
```

(b) (3 points) Fill in each blank with the letter that corresponds to the code that produced it.

#A.

```
for c in Classes:
    print("{:4s} {:5s} {:4d} {:4d} {:5.2f}% {:5.2f}%".format(c.dept, c.num, c.cap,
            len(c.roster), percent_of_capacity(c), relative_waitlist_size(c)))
```

#B.

```
for c in Classes:
    print("{:4s} {:5s} {:4d} {:4d} {:5.0f}% {:5.0f}%".format(c.dept, c.num, c.cap,
            len(c.roster), percent_of_capacity(c), relative_waitlist_size(c)))
```

#C.

```
for c in Classes:
    print("{:4s} {:5s} {:4d} {:4d} {:6.1f}% {:6.1f}%".format(c.dept, c.num, c.cap,
            len(c.roster), percent_of_capacity(c), relative_waitlist_size(c)))
```

#D.

```
for c in Classes:
    print("{:4s} {:5s} {:4d} {:4d} {:6.2f}% {:6.2f}%".format(c.dept, c.num, c.cap,
            len(c.roster), percent_of_capacity(c), relative_waitlist_size(c)))
```

_____
```
ICS  31     350    2     1%     0%
Infx 269      3    3   100%    33%
```
 # From B. SCORING: −1 per mistake

_____
```
ICS  31     350    2    0.6%    0.0%
Infx 269      3    3  100.0%   33.3%
```
 # From C [Floor is zero]

_____
```
ICS  31     350    2  0.57%  0.00%
Infx 269      3    3 100.00% 33.33%
```
 # From A

_____
```
ICS  31     350    2   0.57%   0.00%
Infx 269      3    3 100.00%  33.33%
```
 # From D

Problem 5 (5 points) **Topic: Control structure behavior**

The code below returns the position, or index, where a Student with a given ID occurs in a list of Students.

```
def find_student_by_ID(L: 'list of Student', i: str) -> int:
    ''' Return index of specified student in list, or -1 if not found
    '''
    index = -1
    for n in range(len(L)):
        if L[n].ID == i:
            index = n
            break
    return index
```

Which of following are correct ways of rewriting this function? Circle *one or more* of A, B, C, or D; more than one may be correct.

A. **THIS ONE**

```
def find_student_by_ID(L: 'list of Student', i: str) -> int:
    ''' Return index of specified student in list, or -1 if not found
    '''
    index = -1
    for n in range(len(L)):
        if L[n].ID == i:
            index = n
    return index
```

B. **THIS ONE**

```
def find_student_by_ID(L: 'list of Student', i: str) -> int:
    ''' Return index of specified student in list, or -1 if not found
    '''
    for n in range(len(L)):
        if L[n].ID == i:
            return n
    return -1
```

C.

```
def find_student_by_ID(L: 'list of Student', i: str) -> int:
    ''' Return index of specified student in list, or -1 if not found
    '''
    index = -1
    for n in range(len(L)):
        if L[n].ID == i:
            index = n
        return index    # this return is done the first time through the loop, every time
    return index
```

D.

```
def find_student_by_ID(L: 'list of Student', i: str) -> int:
    ''' Return index of specified student in list, or -1 if not found
    '''
    index = -1
    for n in range(len(L)):
        if L[n].ID == i:
            index = n
        else:
            return n    # Here again, if we don't match L[0] on the first pass through, we lose.
    return index
```

Problem 6 (5 points) **Topic: String processing**

Here is the skeleton of a function to help in printing business letters:

```
def salutation(c: Course) -> str:
    '''  Return a string in the form "Dear Prof. X", where X is the last name (family
         name) of the course's instructor
    '''
    — Alternative body code goes here —
assert salutation(econ20a) == "Dear Prof. Chalfant"
assert salutation(infx269) == "Dear Prof. Kay"
```

Each of the following is a candidate for the body of this function. Circle *one or more* of the alternatives A, B, C, or D to indicate which candidate bodies are correct.

A. **THIS ONE**

```
return "Dear Prof. " + c.instr[:c.instr.find(",")]
```

B. **THIS ONE**

```
name_last_first = c.instr
separator = name_last_first.find(",")
lastname = c.instr[0:separator]
return "Dear Prof. " + lastname
```

C. **THIS ONE**

```
lastname = ''
position = 0
while c.instr[position] != ",":
    lastname += c.instr[position]
    position += 1
return "Dear Prof. " + lastname
```

D. **THIS ONE**

```
location = 0
for letter in c.instr:
    if letter != ",":
        location += 1
    else:
        break
return "Dear Prof. " + c.instr[:location]
```

Problem 7 (19 points) **Topic: Processing lists of namedtuples containing lists**

Now the Registrar wants you to implement some functions that manage course enrollments.

(a) (5 points) Complete the definition of the function below, consistent with its header, docstring comment, and assertions, by filling each blank with exactly one identifier, operator, or constant.

```
def is_enrolled(c: Course, sid: str) -> bool:
    ''' Does the specified student ID belong to a student enrolled in the course?
    '''
    for s in c._____________:                    # roster
        if s._____________ == _____________:     # ID    sid
            return _____________                 # True
    return _____________                         # False
assert is_enrolled(infx269, '22223333')
assert not is_enrolled(ics31, '33334444')
```

(b) (8 points) There are blanks in this code for comments. For each blank, choose the letter from the list below that indicates the most appropriate comment. Use each letter exactly once.

```
def drop(c: Course, sid: str) -> Course:
    '''  Return the Course, but with the student with the specified ID having been
         removed from the roster and the first person on the waiting list, if any
         having been added.
    '''
    location = find_student_by_ID(c.roster, sid)    # ____Determine (A)
    if location == -1:                              # ____Check (E)
        return c                                    # ____If (G)
    new_roster  = c.roster[:location] + c.roster[location+1:] # ____Remove (F)
    if len(c.waitlist) > 0:
        new_roster += [c.waitlist[0]]               # ____Add (C)
        new_waitlist = c.waitlist[1:]               # ____Take (H)
        return c._replace(roster = new_roster, waitlist = new_waitlist)  # ____Up/Ad/D
    else:
        return c._replace(roster = new_roster)      # ____Update/Removed(B)

assert drop(econ20a, '11112222') == Course('Economics', '20A', 'Intro Econ', 4,
                  'Chalfant, Jim', 8, [pp2, aa1, aa2], [])
assert drop(infx269, '22223333') == Course('Infx', '269', 'Computer Law', 4,
                  'Kay, David', 3, [pp1, aa2, aa1], [])
```

A. Determine student's position on list

B. Update course's roster with student removed

C. Add the first person on the waiting list to the roster

D. Update course's roster with student added from the waiting list

E. Check whether the specified student is now enrolled

F. Remove the student from the roster of enrolled students

G. If an error occurs, return the course unchanged

H. Take the first student off of the waiting list

(c) (6 points) Complete the definition of the function below, consistent with its header, docstring comment, and assertions, by filling each blank with exactly one identifier, operator, or constant.

```
def withdraw(L: 'list of Course', sid: str) -> 'list of Course':
    ''' Remove the specified student from all courses he or she is enrolled in
    '''
    result = [ ]
    for c in _____________: # L
        _____________.append(_____________(_____________, _____________))
    return result          # Above:  result   drop   c   sid.      1 point each except 2 for drop.
```

When you're done, please:

- Gather up all your stuff.
- Take your stuff and your exam down to the front of the room.
- Turn in your exam; show your ID if asked.
- Exit by the doors at the front of the room. Don't go back or disturb students still taking the test.