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)

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

```python
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:

```python
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.

```python
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 ____________  ____________:
        if s.____________ == ____________:
            result.append(____________)
    return ____________
```

```python
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.

```python
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.
    
```
Problem 2  (10 points)

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

\[
\text{Course} = \text{namedtuple('Course', 'dept num title units instr cap roster waitlist')}
\]

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.

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

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.

\[
\begin{align*}
(a.1) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{ics31} &= \text{Course}
\end{align*}
\]

\[
\begin{align*}
(a.2) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{infx269.waitlist} &= \text{list} \quad \# \text{of} \quad \text{Student}
\end{align*}
\]

\[
\begin{align*}
(a.3) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{len(ics31.waitlist)} &= \text{bool}
\end{align*}
\]

\[
\begin{align*}
(a.4) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{Classes[0].cap} &= \text{int}
\end{align*}
\]

\[
\begin{align*}
(a.5) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{infx269.num} &= \text{str}
\end{align*}
\]

\[
\begin{align*}
(a.6) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{Classes[1]} &= \text{list} \quad \# \text{of} \quad \text{Course}
\end{align*}
\]

\[
\begin{align*}
(a.7) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{Classes[1].roster} &= \text{list} \quad \# \text{of} \quad \text{Student}
\end{align*}
\]

\[
\begin{align*}
(a.8) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{Classes[1].roster[2].major} &= \text{str}
\end{align*}
\]

\[
\begin{align*}
(a.9) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{len(pp1.name)} &= \text{int}
\end{align*}
\]

\[
\begin{align*}
(a.10) & \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \\
\text{Classes[0].roster[1]} &= \text{list} \quad \# \text{of} \quad \text{Course}
\end{align*}
\]
(b) (5 points) Give the value of each of these expressions, based on the definitions above. Remember zero-based indexing.

ics31.instr == 'David Kay'

len(Classes[3].dept)

Classes[0].roster[1].name

Classes[2].waitlist

Classes[1].roster[2].ID[-1]

Problem 3 (5 points)

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

```python
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,aa1])) == 80
assert relative_waitlist_size(econ20a) == 0
assert relative_waitlist_size(infx269) == 1/3 * 100
```

Which of the following code segments will print 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.
```python
for c in sorted(Classes, key=percent_of_capacity, reverse=True):
    print(c)
```

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

C.
```python
Classes.sort(key=percent_of_capacity, reverse=True)
for c in Classes:
    print(c)
```
[continued]
D.

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

E.

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

**Problem 4 (7 points)**

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:

```python
print('Course      Cap  Enr % Full % Wait')
print('------      ---  --- ------ ------')
for c in Classes:
    print("{:4s} {:5d} {: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.

<table>
<thead>
<tr>
<th>Course</th>
<th>Cap</th>
<th>Enr % Full % Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS 31</td>
<td>350</td>
<td>2 0.6% 0.0%</td>
</tr>
<tr>
<td>Infx 269</td>
<td>3</td>
<td>3 100.0% 33.3%</td>
</tr>
<tr>
<td>ICS 32</td>
<td>500</td>
<td>2 0.4% 0.0%</td>
</tr>
<tr>
<td>Economics 20A</td>
<td>8</td>
<td>4 50.0% 0.0%</td>
</tr>
</tbody>
</table>

B.

<table>
<thead>
<tr>
<th>Course</th>
<th>Cap</th>
<th>Enr % Full % Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS 31</td>
<td>350</td>
<td>2 1% 0%</td>
</tr>
<tr>
<td>Infx 269</td>
<td>3</td>
<td>3 100% 33%</td>
</tr>
<tr>
<td>ICS 32</td>
<td>500</td>
<td>2 0% 0%</td>
</tr>
<tr>
<td>Econ 20A</td>
<td>8</td>
<td>4 50% 0%</td>
</tr>
</tbody>
</table>

C.

<table>
<thead>
<tr>
<th>Course</th>
<th>Cap</th>
<th>Enr % Full % Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS 31</td>
<td>350</td>
<td>2 0.6% 0.0%</td>
</tr>
<tr>
<td>Infx 269</td>
<td>3</td>
<td>3 100.0% 33.3%</td>
</tr>
<tr>
<td>ICS 32</td>
<td>500</td>
<td>2 0.4% 0.0%</td>
</tr>
<tr>
<td>Econ 20A</td>
<td>8</td>
<td>4 50.0% 0.0%</td>
</tr>
</tbody>
</table>

D.

<table>
<thead>
<tr>
<th>Course</th>
<th>Cap</th>
<th>Enr % Full % Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS 31</td>
<td>350</td>
<td>2 0.6% 0.0%</td>
</tr>
<tr>
<td>Infx 269</td>
<td>3</td>
<td>3 100.0% 33.3%</td>
</tr>
<tr>
<td>ICS 32</td>
<td>500</td>
<td>2 0.4% 0.0%</td>
</tr>
<tr>
<td>Economics 20A</td>
<td>8</td>
<td>4 50.0% 0.0%</td>
</tr>
</tbody>
</table>
### E.

<table>
<thead>
<tr>
<th>Course</th>
<th>Cap</th>
<th>Enr</th>
<th>% Full</th>
<th>% Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS</td>
<td>350</td>
<td>2</td>
<td>0.6%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Infx</td>
<td>3</td>
<td>3</td>
<td>100.0%</td>
<td>33.3%</td>
</tr>
<tr>
<td>ICS</td>
<td>500</td>
<td>2</td>
<td>0.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Economics 20A</td>
<td>8</td>
<td>4</td>
<td>50.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

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

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

#C.
```python
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.
```python
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% 0%
Infx 269 3 3 100% 33%

_____ ICS 31 350 2 0.6% 0.0% 0.0%
Infx 269 3 3 100.0% 33.3%

_____ ICS 31 350 2 0.57% 0.00% 0.00%
Infx 269 3 3 100.00% 33.33%

_____ ICS 31 350 2 0.57% 0.00% 0.00%
Infx 269 3 3 100.00% 33.33%
Problem 5 (5 points)

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

```python
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 the 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.
```python
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.
```python
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.
```python
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
```

D.
```python
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
    return index
```
Problem 6 (5 points)

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

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

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

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

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

Problem 7 (19 points)

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.

```python
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.__________:
        if s.__________ == __________:  
            return __________
    return __________
```

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.

ICS 31 • FALL 2015 • DAVID G. KAY • UC IRVINE
SECOND MIDTERM • PAGE8
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)  # __
    if location == -1:  # __
        return c  # __
    new_roster = c.roster[:location] + c.roster[location+1:]  # __
    if len(c.waitlist) > 0:  # __
        new_roster += [c.waitlist[0]]  # __
        new_waitlist = c.waitlist[1:]  # __
        return c._replace(roster = new_roster, waitlist = new_waitlist)  # __
    else:
        return c._replace(roster = new_roster)  # __

assert drop(econ20a, '11112222') == Course('Economics', '20A', 'Intro Econ', 4, 'Chalfant, Jim', 8, [pp2, aal, aal2], [])
assert drop(infx269, '22223333') == Course('Infx', '269', 'Computer Law', 4, 'Kay, David', 3, [pp1, aal2, aal1], [])

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 ____________:
        ____________.append(___________(__________, ____________))
    return result