ICS 31 • UC Irvine
Fall 2016 • David G. Kay

Your Name _____
Your Student ID (8 digits) _____
Your UCInet ID _____@uci.edu

Your Lab Section (circle one):

1. 8:00a Syed Safir
2. 9:30 Syed Safir
3. 11:00 Nathaniel Baer
4. 12:30 Nathaniel Baer
5. 2:00 Yadhu Prakash
6. 3:30 Yadhu Prakash
7. 5:00 Anurag Mishra
8. 6:30 Anurag Mishra
9. 8:00p Jason Desrosiers
10. 8:00a Swarun Krishnamoorthy
11. 9:30 Swarun Krishnamoorthy
12. 11:00 Harun Anver
13. 12:30 Harun Anver
14. 8:00a Karthik Prasad
15. 9:30 Karthik Prasad
16. 11:00 Jason Desrosiers

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 with or receive from anyone besides the instructor or TAs any information or materials during the exam. 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 (4 points)	
Problem 2 (11 points)	
Problem 3 (12 points)	
Problem 4 (3 points)	
Problem 5 (3 points)	
Problem 6 (8 points)	
Problem 7 (19 points)	
Total (60 points)	

Problem 1 (4 points) **Topic: Python expressions and data types**

Use the following definitions in this problem:

```
s = 'Four score and seven years ago, our fathers brought forth upon this ...'
L = [314, 159, 265, 358, 979, 323, 846, 264, 338, 327]
```

Below are eight segments of code, each with a part underlined. Indicate the data type of each underlined part by checking the appropriate box. **# SCORING: 1/2 point each**

(a) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
for x in range(len(s)):      # func
    if s[x] == ' ':
        print(s[x])
```

(b) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
for x in range(len(s)):    # int
    if s[x] == ' ':
        print(s[x])
```

(c) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
for x in s:              # str
    if x == ' ':
        print(x)
```

(d) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
for x in s:
    if x == ' ':
        print(x)          # bool
```

(e) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
result = 0
for n in L:
    result += n
assert(result > 0)       # bool
```

(f) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
print(L[3:5])        # list
```

(g) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
print(s[1])      # int
```

(h) ❑ int ❑ float ❑ bool ❑ str ❑ function ❑ list

```
print(s[1:4])      # str
```

Problem 2 (11 points) **Topic: Types of combined data structures**

Use the following definitions in this problem:

```
Course = namedtuple('Course', 'dept num title instr units')
# Each field is a string except the number of units
# An example showing the form of the data:
ics31 = Course('ICS', '31', 'Intro to Programming', 'Kay', 4.0)
ics32 = Course('ICS', '32', 'Programming with Libraries', 'Thornton', 4.0)
wr39a = Course('Writing', '39A', 'Intro Composition', 'Alexander', 4.0)
wr39b = Course('Writing', '39B', 'Intermediate Composition', 'Gross', 4.0)
bio97 = Course('Biology', '97', 'Genetics', 'Smith', 4.0)
mgt1  = Course('Management', '1', 'Intro to Management', 'Jones', 2.0)

Student = namedtuple('Student', 'ID name level major studylist')
# All are strings except studylist, which is a list of Courses.
# An example showing the form of the data:
sW = Student('11223344', 'Anteater, Peter', 'FR', 'PSB', [ics31, wr39a, bio97, mgt1])
sX = Student('21223344', 'Anteater, Andrea', 'SO', 'CS', [ics31, wr39b, bio97, mgt1])
sY = Student('31223344', 'Programmer, Pat', 'FR', 'COG SCI', [ics32, wr39a, bio97])
sZ = Student('41223344', 'Programmer, Patsy', 'SR', 'PSB', [ics32, mgt1])

StudentBody = [sW, sX, sY, sZ]
```

Below are 12 Python expressions. Indicate the data type of each expression by checking the appropriate box. **# SCORING: 1/2 point each. The value isn't required here.**

(a) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`bio97` **# Course**

(b) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[0].studylist` **# list of Course**

(c) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[2].name` **# str, [value "Programmer, Pat"]**

(d) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`sX` **# Student**

(e) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[1].studylist[0]` **# Course**

(f) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody` **# list of Student**

(g) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[2]` **# Student**

(h) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[3].studylist[0].title` **# str, [value "Programming with Libraries"]**

(i) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`sX.level` **# str, value SO**

(j) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`mgt1.units` **# float, [value 2.0]**

(k) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[1:3]` **# list of Student**

(l) ❑int ❑float ❑bool ❑str ❑function ❑Course ❑Student ❑list of Course ❑list of Student

`StudentBody[2].studylist[1].num` **# str, [value 39a]**

(m) (5 points) Give the *value* of each of these expressions, based on the definitions above. Remember zero-based indexing. **# SCORING: 1 point each (for value; type not required)**

`mgt1.units` **# float, value 2.0**

`StudentBody[3].studylist[0].title` **# str, value "Programming with Libraries"**

`sX.level` **# str, value SO**

`StudentBody[2].studylist[1].num` **# str, value 39a**

`StudentBody[2].name` **# str, value Programmer, Pat**

Problem 3 (12 points) **Topic: Loop behavior**

For this problem, use these definitions:

```
L = ['Bean', 'Lettuce', 'Artichokes', 'Celery']
M = [100, 20, 7, 3000, 1]
```

Match each of the following code segments ((a) through (d)) with the results (A through I) they produce when run in Python. You may use some results (A through I) more than once.

(a) Circle one: A B C D E F G H I **---> D**

```
for v in L:
    print(v, len(v))
print('Done', len(L))
```

(b) Circle one: A B C D E F G H I **---> A**

```
n = 0
for v in range(len(M)):
    print(M[v], v)
    n = n + M[v]
print('Done', n)
```

(c) Circle one: A B C D E F G H I **---> E**

```
n = 0
for v in M:
    n += v
    print(v, n)
print('Done', n)
```

(d) Circle one: A B C D E F G H I **---> H**

```
for v in L[0]:
    print(v, L[0])
print('Done', len(L[0]))
```

SCORING: 3 points each

A.

```
100 0
20 1
7 2
3000 3
1 4
Done 3128
```

B.

```
K 0
i 1
n 2
g 3
Done 4
```

C.

```
TypeError: list indices
must be integers, not str
```

D.

```
Bean 4
Lettuce 7
Artichokes 10
Celery 6
Done 4
```

E.

```
100 100
20 120
7 127
3000 3127
1 3128
Done 3128
```

F.

```
Bean 4
Lettuce 4
Artichokes 4
Celery 4
Done 4
```

G.

```
100 100
120 20
127 7
3127 3000
3128 1
Done 3128
```

H.

```
B Bean
e Bean
a Bean
n Bean
Done 4
```

I.

```
B B
e Be
a Bea
n Bean
Done 0
```

Problem 4 (3 points) **Topic: String formatting**

Here are some statistics on movies nominated for Academy Awards:

```
The Martian          $176.0 12
Room                 $127.1  7
Brooklyn             $303.2  5
Mad Max: Fury Road   $157.3  5
The Big Short        $447.4  0
```

The second column is the movie's "box office" (the amount of money it has taken in so far, in millions); the third column is the number of Academy Award nominations. Suppose that you represent this information in a namedtuple like this for each movie:

```
Movie = namedtuple('Movie', 'title income nominations')
```

If you have a list of these Movie objects and you want to print their information in the format of the table shown above, you could use a statement like this:

```
for m in MovieList:
    print(format_string.format(m.title, m.income, m.nominations))
```

Which one of the following values of `format_string` would format the movies correctly?

A. `"{:20} ${:5.2f} {}"`

B. `"{:20} ${:5.1f} {:2}"` **<--- THIS ONE. 3 pts for correct answer, 0 otherwise**

C. `"{} ${:5.2f} {:2}"`

D. `"{} ${:5.1f} {}"`

E. `"{:20} ${:5.1f} {:8}"`

Problem 5 (3 points) **Topic: String processsing**

This function is missing its body:

```
def remove_extra_whitespace(s: str) -> str:
    ''' Replace multiple whitespace characters with one blank '''
    — Insert body here —

x = """    Four    score    and
    seven         years ago          """

assert remove_extra_whitespace(x) == 'Four score and seven years ago'
```

Which one of the following is a correct body for the function?

A. `return s.split().join(" ")`

B. `return s.replace(" ", "")`

C. `return " ".join(s.split())` **<--- THIS ONE. 3 pts for correct answer, 0 otherwise**

D. `return s.replace(" \t\n", " ")`

E. `return s.translate(str.maketrans(" \t\n", "   ")`

Problem 6 (8 points) **Topic: Processing lists of namedtuples**

For this problem, use these definitions (which are the same as earlier on this exam):

```
Course = namedtuple('Course', 'dept num title instr units')
# Each field is a string except the number of units
ics31 = Course('ICS', '31', 'Intro to Programming', 'Kay', 4.0)
ics32 = Course('ICS', '32', 'Programming with Libraries', 'Thornton', 4.0)
wr39a = Course('Writing', '39A', 'Intro Composition', 'Alexander', 4.0)
bio97 = Course('Biology', '97', 'Genetics', 'Smith', 4.0)
```

(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 Course_equals(c1: Course, c2: Course) -> bool:
    ''' Return True if the department and number of c1 match the department and
        number of c2 (and False otherwise)
    '''
    return (c1._____ _____ c2._____  and

            _____._____ == _____._____)

assert(Course_equals(ics31, ics31))
assert(not Course_equals(ics31, ics32))
assert(Course_equals(ics31, Course('ICS', '31', '', '', 0)))
```

return c1.dept == c2.dept and c1.num == c2.num

SCORING: 1/2 point for each dept, 1/2 point for each num, 1 point for ==, 1 pt. for c1, 1 pt. for c2

[continued on next page]

(b) (3 points) Choose all of the following code segments (A through E) that correctly complete the definition of the function below, consistent with its header, docstring comment, and assertions. One or more code segments may be correct.

```
def Course_on_studylist(c: Course, CL: 'list of Course') -> bool:
    ''' Return True if the course c equals any course on the list CL (where equality
        means matching department name and course number) and False otherwise.
    '''
    — Insert body of function here (A, B, C, D, or E) —
assert Course_on_studylist(ics31, [ics32, ics31, bio97])
assert not Course_on_studylist(ics31, [ ])
assert not Course_on_studylist(wr39a, [ics32, ics31, bio97])
```

A.
```
result = False          ## THIS ONE
for a_course in CL:
    if Course_equals(c, a_course):
        result = True
return result
```

B.
```
for a_course in CL:
    if Course_equals(c, a_course):   ## THIS ONE
        return True
return False
```

C.
```
for a_course in CL:                  ## NO
    if Course_equals(c, a_course):
        return True
    return False
```

D.
```
for i in range(len(CL)):             ## THIS ONE
    if Course_equals(c, CL[i]):
        return True
return False
```

E.
```
for i in range(len(CL)):             ## NO
    if Course_equals(CL[i], a_course):
        return True
return False
```

SCORING: 3 points max, -1 for each incorrectly circled or incorrectly un-circled (min. 0)

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

For this problem, use the definitions below (which are the same as earlier on this exam). If a function defined earlier in this exam is appropriate in an answer to this question, you should use it to receive full credit [regardless of whether you answered the earlier question correctly yourself].

```
Course = namedtuple('Course', 'dept num title instr units')
# Each field is a string except the number of units
ics31 = Course('ICS', '31', 'Intro to Programming', 'Kay', 4.0)
ics32 = Course('ICS', '32', 'Programming with Libraries', 'Thornton', 4.0)
wr39a = Course('Writing', '39A', 'Intro Composition', 'Alexander', 4.0)
wr39b = Course('Writing', '39B', 'Intermediate Composition', 'Gross', 4.0)
bio97 = Course('Biology', '97', 'Genetics', 'Smith', 4.0)
mgt1  = Course('Management', '1', 'Intro to Management', 'Jones', 2.0)

Student = namedtuple('Student', 'ID name level major studylist')
# All are strings except studylist, which is a list of Courses.
sW = Student('11223344', 'Anteater, Peter', 'FR', 'PSB', [ics31, wr39a, bio97, mgt1])
sX = Student('21223344', 'Anteater, Andrea', 'SO', 'CS', [ics31, wr39b, bio97, mgt1])
sY = Student('31223344', 'Programmer, Pat', 'FR', 'COG SCI', [ics32, wr39a, bio97])
sZ = Student('41223344', 'Programmer, Patsy', 'SR', 'PSB', [ics32, mgt1])

StudentBody = [sW, sX, sY, sZ]
```

(a) (3 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 Courses_enrolled(S: Student) -> int:
    ''' Return the number of Courses on this Student's study list
    '''
    return __________(__________.__________)
#     return len(S.studylist)          SCORING: 1 point per blank
assert(Courses_enrolled(sW) == 4)
assert(Courses_enrolled(sZ) == 2)
assert(Courses_enrolled(Student('007', 'Bond, James', 'GR', 'MI6', [ ])) == 0)
```

(b) (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 Student_is_enrolled(S: Student, department: str, coursenum: str) -> bool:
    ''' Return True if the course (department and course number) is on the student's
        studylist (and False otherwise)
    '''                                  # SCORING: 1 point per blank


    return __________(Course(__________, __________, '', '', 0),

                      __________.__________)
# return Course_on_studylist(Course(department, coursenum, '', '', 0), S.studylist)
assert(Student_is_enrolled(sW, 'ICS', '31'))
assert(Student_is_enrolled(sX, mgt1.dept, mgt1.num))
assert(not Student_is_enrolled(sY, 'ICS', '31'))
```

(c) (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 Student_units(S: Student) -> float:
    ''' Return the total number of units this student is enrolled in
    '''
    total = __________                          0.  SCORING:  1 per correct blank

    for c in S.__________:                      for c in S.studylist

        total += __________.__________          c.units

    return total

assert(Student_units(sW) == 14)
assert(Student_units(Student('007', 'Bond, James', 'GR', 'MI6', [ ])) == 0)
assert(Student_units(sZ) == 6)
```

(d) (7 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 average_units(SB: 'list of Student') -> float:
    ''' Return the average number of enrolled units per student in the student body
    '''
    total = 0
    for s in __________:                                for s in SB:
        total += __________(__________)                     total += Student_units(s)
    if len(SB) == 0:
        return 0
    else:
        return __________ __________ __________ (__________)
         return total / len(SB)                         SCORING:  1 point/correct blank
assert(average_units([ ]) == 0)
assert(average_units([sW, sX]) == (Student_units(sW) + Student_units(sX))/2)
assert(average_units(StudentBody) == (14+14+12+6)/4)
```

Problem 8 (0 points)

When you're done with the exam, follow these steps (so you don't disturb your classmates and so your exam gets turned in properly):

- Write your UCInet ID in the blanks at the top of the odd-numbered pages. Also check for your name on the front page.
- 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 to your seat or disturb students who are still working.