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

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

Your Lab Section (circle one):

1. 8:00 Sanket Khanwalkar
2. 10:00 Neeraj Kumar
3. 12:00 Sanket Khanwalkar
4. 2:00 Vignesh Raghunathan
5. 4:00 Vignesh Raghunathan
6. 6:00 Neeraj Kumar
7. 10:00 Andrea D'Souza
8. 2:00 Andrea D'Souza

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	(10 points)
Problem 2	(3 points)
Problem 3	(12 points)
Problem 4	(4 points)
Problem 5	(19 points)
Problem 6	(12 points)
Total	(60 points)

Problem 1 (10 points) **Topic: Identifying types with lists and namedtuples**

The ZotCare Clinic asked you to computerize their business. You represent each of their doctors with:

```
Doctor = namedtuple('Doctor', 'name specialty price visits')
```

where the name and specialty are strings, the price is a float (the cost of an office visit), and visits is an int (the number of patient visits to this doctor's office in the past month). You will keep track of each patient at the clinic with

```
Patient = namedtuple('Patient', 'name phone deductible docs')
```

where the name and phone are strings, the deductible is a float (the amount the patient has to pay before insurance covers the rest), and docs is a list of Doctors that the patient has seen in the last month.

Use the following definitions in this problem:

```
DrAA = Doctor('Anteater, Andrew', 'Pediatrics', 125.00, 300)
DrBB = Doctor('Bear, Betsy', 'Cardiology', 225.00, 150)
DrCC = Doctor('Cheetah, Charles', 'Geriatrics', 99.50, 200)
DrDD = Doctor('Dingo, Diana', 'Orthopedics', 235.00, 220)
DrEE = Doctor('Echidna, Edith', 'Pediatrics', 145.00, 250)

pV = Patient('Vicuna, Vicki', '444-3333', 1000.00, [DrAA])
pW = Patient('Wallaby, Walter', '333-4444', 250.00, [DrBB, DrCC, DrEE])
pY = Patient('Yak, Yetta', '444-4444', 500.00, [DrBB, DrCC])
pZ = Patient('Zebra, Zoltan', '333-3344', 300.00, [DrAA, DrCC, DrDD, DrEE])

PatientBase = [pV, pW, pY, pZ]
```

(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 ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`pY` **# Patient, SCORING 1/2 point each**

(a.2) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`pZ.deductible` **# float,**

(a.3) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`pZ.docs` **# list of Doctor,**

(a.4) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`PatientBase[2]` **# Patient**

(a.5) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`pW.docs[0:2]` **# list of Doctor,**

(a.6) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`DrCC` **# Doctor**

(a.7) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`DrEE.specialty` **# str,**

(a.8) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`PatientBase[3].docs` **# list of Doctor**

(a.9) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`PatientBase[0].docs[0].price` **#float,**

(a.10) ❑int ❑float ❑bool ❑str ❑function ❑Doctor ❑Patient ❑list of Doctor ❑list of Patient
`pW.docs[2].name[0]` **# str**

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

SCORING: 1 point each

```
pZ.deductible                    # val 300 or 300.0 or 300.00 [Other numbers, check other versions]

PatientBase[1].docs[0].price     # val 225.00 (or 225 or 225.0)

PatientBase[2].name              # value "Yak,Yetta" (quotes not necessary)

DrBB.specialty                   # value 'Cardiology' (quotes not necessary)[Other values, other versions]

pW.docs[2].name[0]               # value 'E' (quotes not necessary)
```

Problem 2 (3 points) **Topic: String formatting**

These definitions appeared earlier on this exam:

```
Doctor = namedtuple('Doctor', 'name specialty price visits')
DrAA = Doctor('Anteater, Andrew', 'Pediatrics', 125.00, 300)
DrBB = Doctor('Bear, Betsy', 'Cardiology', 225.00, 150)
DrCC = Doctor('Cheetah, Charles', 'Geriatrics', 99.50, 200)
DrDD = Doctor('Dingo, Diana', 'Orthopedics', 235.00, 220)
DrEE = Doctor('Echidna, Edith', 'Pediatrics', 145.00, 250)
```

Suppose we have a list (called `DL`) of all the Doctors and we wish to produce a report on their revenues this month. We'd like the report to look like this:

```
Doctor                 Price  Visits     Revenue
------                 -----  ------     -------
Anteater, Andrew     $125.00     300  $ 37500.00
Bear, Betsy          $225.00     150  $ 33750.00
Cheetah, Charles     $ 99.50     200  $ 19900.00
Dingo, Diana         $235.00     220  $ 51700.00
Echidna, Edith       $145.00     250  $ 36250.00
```

You could print the table with code like this:

```
DL = [DrAA, DrBB, DrCC, DrDD, DrEE]

print('Doctor                 Price  Visits     Revenue')
print('------                 -----  ------     -------')
for d in DL:
    print(format_string.format(d.name, d.price, d.visits, d.price * d.visits))
```

Which one of the following values of `format_string` would format the lines correctly? Circe the one correct answer.

A. `"{} ${:6.2f}  {:6d}  ${:9.2f}"`

B. `"{:20s} ${:6.2f} {:4d} ${:9.2f}"`

C. `"{:20s} ${:6.2f}    {:4d}  ${:9.1f}"`

D. `"{:20s} ${:6.2f}  {:6d}  ${:9.2f}"` **THIS ONE; SCORING: 3 points for this; else zero.**

E. `"{:20s} ${:0.2f}  {:6d}  ${:0.2f}"`

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

For this problem, use these definitions:

```
S = [200, 800, 1000]
T = ['samosa', 'dumpling', 'pierogi', 'empanada']
```

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 **---> E**

```
i = 0
for n in range(len(S)):
    print(S[n], n)
    i = i + S[n]
print('End', i)
```

(b) Circle one: A B C D E F G H I **---> F, 1 pt for D.**

```
for f in T:
    print(f, len(f))
print('End', len(T))
```

(c) Circle one: A B C D E F G H I **---> A, 1 pt for C**

```
n = 0
for i in S:
    n += i
    print(i, n)
print('End', n)
```

(d) Circle one: A B C D E F G H I **---> H, 1 pt for G**

```
for c in T[0]:
    print(T[0], c)
print('End', len(T[0]))
```

SCORING: 3 points each

A.
```
200 200
800 1000
1000 2000
End 2000
```

B.
```
s 0
a 1
m 2
o 3
s 4
a 5
End 6
```

C.
```
200 200
1000 800
2000 1000
End 2000
```

D.
```
samosa 4
dumpling 4
pierogi 4
empanada 4
End 4
```

E.
```
200 0
800 1
1000 2
End 2000
```

F.
```
samosa 6
dumpling 8
pierogi 7
empanada 8
End 4
```

G.
```
0 200
1 800
2 1000
2000 End
```

H.
```
samosa s
samosa a
samosa m
samosa o
samosa s
samosa a
End 6
```

I.
```
samosa s
samosa sa
samosa sam
samosa samo
samosa samos
samosa samosa
End 6
```

Problem 4 (4 points) **Topic: Processing lists of namedtuples**

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

```
Doctor = namedtuple('Doctor', 'name specialty price visits')

Patient = namedtuple('Patient', 'name phone deductible docs')

DrAA = Doctor('Anteater, Andrew', 'Pediatrics', 125.00, 300)
DrBB = Doctor('Bear, Betsy', 'Cardiology', 225.00, 150)
DrCC = Doctor('Cheetah, Charles', 'Geriatrics', 99.50, 200)
DrDD = Doctor('Dingo, Diana', 'Orthopedics', 235.00, 220)
DrEE = Doctor('Echidna, Edith', 'Pediatrics', 145.00, 250)
```

Choose which one of the following code segments (A through D) correctly completes the definition of the function below, consistent with its header, docstring comment, and assertions. Only one code segment is correct.

```
def count_specialists(DL: 'list of Doctor', spec_to_count: str) -> int:
    ''' Return the number of Doctors on the list with the specified specialty.
    '''
```

— Insert one of the code segments A–D here —

```
assert count_specialists([DrAA,DrBB,DrCC,DrDD,DrEE], 'Pediatrics') == 2
assert count_specialists([DrAA,DrBB,DrCC,DrDD,DrEE], 'Orthopedics') == 1
assert count_specialists([DrAA,DrBB,DrCC,DrDD,DrEE], 'Psychiatry') == 0
```

A.

This one: 4 points.

```
    total = 0
    for d in DL:
        if d.specialty == spec_to_count:
            total = total + 1
    return total
```

B.

```
    for d in DL:
        if d.specialty == spec_to_count:
            total = total + 1
    return total
```

C.

```
    total = 0
    for d in DL:
        if d.specialty == spec_to_count:
            total = total + 1
            return total
```

D.

```
    total = 0
    for d in DL:
        if d.specialty == spec_to_count:
            total = total + 1
        return total
```

Problem 5 (19 points) **Topic: Processing namedtuples containing lists**

For full credit on this problem, use the definitions below (which are the same as earlier on this exam) and any other definitions on this exam that are appropriate:

```
Doctor = namedtuple('Doctor', 'name specialty price visits')

Patient = namedtuple('Patient', 'name phone deductible docs')

DrAA = Doctor('Anteater, Andrew', 'Pediatrics', 125.00, 300)
DrBB = Doctor('Bear, Betsy', 'Cardiology', 225.00, 150)
DrCC = Doctor('Cheetah, Charles', 'Geriatrics', 99.50, 200)
DrDD = Doctor('Dingo, Diana', 'Orthopedics', 235.00, 220)
DrEE = Doctor('Echidna, Edith', 'Pediatrics', 145.00, 250)

pV = Patient('Vicuna, Vicki', '444-3333', 1000.00, [DrAA])
pW = Patient('Wallaby, Walter', '333-4444', 250.00, [DrBB, DrCC, DrEE])
pY = Patient('Yak, Yetta', '444-4444', 500.00, [DrBB, DrCC])
pZ = Patient('Zebra, Zoltan', '333-3344', 300.00, [DrAA, DrCC, DrDD, DrEE])

PatientBase = [pV, pW, pY, pZ]
```

(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 specialists_seen(P: Patient, s: str) -> int:
    ''' The second argument is the name of a medical specialty.  Return the number of
        doctors with that specialty that have been seen by this patient.
    '''SCORING: 1 pt per correct blank
    return _____ (P._____, _____)
           count_specialists   docs          s   [Both parameter names are different in
assert specialists_seen(pZ, 'Pediatrics') == 2   different versions of the test. If you see the
assert specialists_seen(pZ, 'Geriatrics') == 1   wrong term, check that it's not from another
assert specialists_seen(pZ, 'Cardiology') == 0   version of the test (and flag it if it is).]
```

(b) (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 total_cost_of_visits(P: Patient) -> float:
    ''' Return the total cost of this patient's doctor visits (ignoring deductible)
    '''
    total = _____ 0                    [variable total differs in different versions: check]
    for d in _____ . _____: P.docs  [ Parameter P different in versions]
        total += _____ . _____ d.price  [variable d different in versions]
    return _____ total

assert total_cost_of_visits(pV) == 125.00
assert total_cost_of_visits(pZ) == 125 + 99.50 + 235 + 145
```

(c) (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 over_deductible(P: Patient) -> bool:
    ''' Return True if the patient has spent more on doctor visits than his or her
        deductible amount, and False otherwise.
    '''
    return _____(_____) > _____ .deductible
           total_cost_of_visits  P  [different in versions]      P
assert over_deductible(pZ)
assert not over_deductible(pV)
```

(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_cost(PL: 'list of Patient') -> float:
    ''' Return the average cost per patient of doctor visits
    '''
    total = _________________ 0
    for p in _________________: PL [Different parameter name in different versions: check.]
        total += _________________(_________________) total_cost_of_visits    p [also version diffs]
    return _________________ / _________________(_________________) total    len    PL

assert average_cost(PatientBase) == (125+ 225+99.5+145+ 225+99.5+ 125+99.5+235+145)/4
```

Problem 6 (12 points) Topic: String processing

The following excerpt from `help(str)` may be useful for this problem.

```
find(...)
    S.find(sub) -> int
    Return the lowest index in S where
    substring sub is found.
    Return -1 on failure.

replace(...)
    S.replace(old, new) -> str
    Return a copy of S with all
    occurrences of substring old replaced
    by new.

maketrans(...)
    str.maketrans(x, y) ->
         dict (static method)
    Return a translation table usable for
    translate(). The arguments must be
    strings of equal length, and in the
    resulting dictionary, each character
    in x will be mapped to the
    character at the same position in y.

translate(...)
    S.translate(table) -> str
    Return a copy of the string S, where
    all characters have been mapped
    through the given translation table.
    Unmapped characters are left
    untouched.

upper(...)
    S.upper() -> str
    Return a copy of S converted to
    uppercase.
```

Classified (secret) documents are occasionally released to the public with the names of specific people and places X'd out (to protect sources of information, for example). Thus, a message like "M sent James Bond to Berlin" might be transformed to "X sent Xxxxx Xxxx to Xxxxxx". This process is called "redaction"; we redact the original message to produce a redacted version (with certain words obscured).

[Problem continues on the next page]

(a) (3 points) First let's produce the replacement string for a single term we want to redact.

```
def redact_term (name: str) -> str:
    ''' Return the name with each letter replaced with X or x (according to
        its original upper or lower case) and other characters unchanged.
    '''
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    x_string = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
    ALPHABET = alphabet.upper()
    X_STRING = x_string.upper()

    table = ____________________________________________________

    return name.translate(table)

assert redact_term("") == ""
assert redact_term("Huey") == "Xxxx"
assert redact_term("duck duck Goose") == "xxxx xxxx Xxxxx"
assert redact_term("1600 Pennsylvania Avenue") == "1600 Xxxxxxxxxxx Xxxxxx"
```

Which of the five expressions below could go into the blank in `redact_term` to produce correct results consistent with the function header, docstring, and assertions? Circle *one or more* of A, B, C, D, and E; more than one may be correct.

A. `str.maketrans(alphabet+ALPHABET, x_string+X_STRING)` **THIS ONE**

B. `str.maketrans(ALPHABET+alphabet, x_string+X_STRING)`

C. `str.maketrans(alphabet+x_string, ALPHABET+X_STRING)`

D. `str.maketrans(ALPHABET+alphabet, X_STRING+x_string)` **THIS ONE**

E. `str.maketrans(ALPHABET+X_STRING, alphabet+x_string)`

(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 redact(message: str, terms: 'list of str') -> str:
    ''' In message, change each occurrence of a string in terms to a same-length
        string of Xs.
    '''
    for t in _________________: terms.      NEXT LINE:    message    t    redact_term
        message = _________________.replace(_________________, _________________(t))

    return _________________ message [other versions, other variables: check]


assert redact("Huey said to Louie, 'Get Dewey!'", ['Huey', 'Dewey', 'Louie']) == \
              "Xxxx said to Xxxxx, 'Get Xxxxx!'"

assert redact("Dewey lives at 1600 Pennsylvania Avenue",
                          ['Huey', 'Dewey', 'Louie', '1600 Pennsylvania Avenue']) == \
              "Xxxxx lives at 1600 Xxxxxxxxxxx Xxxxxx"

assert redact("Four score and seven years ago", ['Huey', 'Dewey', 'Louie']) == \
              "Four score and seven years ago"
```

(c) (2 points) Suppose we want to redact digits instead of leaving them alone, so redacting 1600 would produce XXXX. We can do this by redefining the four strings in `redact_term` so this assertion will be true:

```
assert redact_term("1600 Pennsylvania Avenue") == "XXXX Xxxxxxxxxxx Xxxxxx"
```

Below are four alternative sets of redefinitions; one of them is wrong. Circle *just one* of A, B, C, or D to indicate the *wrong* alternative.

A.
```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
x_string = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
ALPHABET = alphabet.upper() + '0123456789'
X_STRING = x_string.upper() + 'XXXXXXXXXX'
#
```

B.
```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
x_string = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
ALPHABET = '0123456789' + alphabet.upper()
X_STRING = x_string.upper() + 'XXXXXXXXXX'
# (X_STRING all Xs, order doesn't matter; awkward but correct)
```

C.
```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
x_string = 'x' * len(alphabet)
DIGITS = '0123456789'
ALPHABET = alphabet.upper() + DIGITS
X_STRING = 'X' * (len(ALPHABET) + len(DIGITS))
# Wrong: string lengths don't match; ALPHABET in last line ALREADY includes digits.  ***** ANSWER *****
```

D.
```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
x_string = 'x' * len(alphabet)
DIGITS = '0123456789'
ALPHABET = alphabet.upper() + DIGITS
X_STRING = 'X' * (len(alphabet + DIGITS))
#
```

(d) (2 points) It would be harder to figure out the actual names from our redacted messages if every term, no matter its original length or case, were simply transformed to XXXX. Fill in the blank with a Python expression that is consistent with the header, docstring, and assertions.

```
def redact_term (name: str) -> str:
    ''' Return 'XXXX', no matter how long or what case the parameter is.
    '''
    return _____________________________________________________  # "XXXX"
                                                                  (quotes req'd for full credit)

assert redact_term("") == "XXXX"
assert redact_term("Huey") == "XXXX"
assert redact_term("Goose duck duck") == "XXXX"
assert redact_term("1600 Pennsylvania Avenue") == "XXXX"
```

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.