ICS 31 • UC Irvine
Spring 2017 • David G. Kay

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

Your Lab Section (circle one):

1. 8:00 Ashwin Achar
2. 9:30 Ashwin Achar
3. 11:00 Nathaniel Baer
4. 12:30 Nathaniel Baer
5. 2:00 Yadhu Prakash
6. 3:30 Yadhu Prakash
7. 5:00 Shreyaan Kaushal
8. 6:30 Shreyaan Kaushal

Second Midterm

You have 75 minutes (until the end of the class period) to complete this exam. There are 61 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 (15 points)	
Problem 2 (18 points)	
Problem 3 (5 points)	
Problem 4 (20 points)	
Problem 5 (3 points)	
Total (61 points)	

Problem 1 (15 points)

U.S. telephone numbers can appear in many forms:

```
949-824-5072  (949)824-5072  9498245072  1-949-824-5072  1(949)824-5072  +1(949)824-5072
```

To process a collection of phone numbers, it's helpful if we store every number in the same form. Below is the outline of a function to perform this task; it shows three steps labeled (i), (ii), and (iii).

```
def phone_number_to_10_digits(original: str) -> str:
    """ Convert the parameter to a string of exactly 10 digits
    """
    # (i)    Change all punctuation to blanks

    # (ii)   Remove all blanks

    # (iii)  Remove extra 1 at beginning if necessary

    return result

assert phone_number_to_10_digits('+1(999)888-7777') == '9998887777'
assert phone_number_to_10_digits('1-999-888-7777') == '9998887777'
assert phone_number_to_10_digits('1(999)888-7777') == '9998887777'
assert phone_number_to_10_digits('19998887777') == '9998887777'
assert phone_number_to_10_digits('9998887777') == '9998887777'
assert phone_number_to_10_digits('(999)888-7777') == '9998887777'
assert phone_number_to_10_digits('999-888-7777') == '9998887777'
```

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.
```

(a) Which of the following code segments correctly implements step (i) above? Circle *one or more* of A, B, C, D, or E; more than one may be correct.

A.
```
result = original.replace("()-+", "    ")
```

B.
```
table = str.maketrans("()+-", "    ")
result = original.translate(table)
```

C.
```
result = translate("()+-", "    ")
```

D.
```
result = original.find("()+-").replace("    ")
```

E.
```
result = original.translate(str.maketrans("()+-", "    "))
```

(b) Which of the following code segments also correctly implements step (i) above? Circle *one or more* of A, B, C, or D; more than one may be correct.

A.
```
result = ''
for c in original:
    if c in "()-+":
        result += " "
    else:
        result += c
```

B.
```
result = ''
if c in original:
    for c in "()-+":
        result += " "
else:
    result += c
```

C.
```
result = ''
for c in original:
    if c not in "()-+":
        result += c
```

D.
```
result = original
for c in "()-+":
    result = result.replace(c, " ")
```

(c) Which of the following code segments correctly implements step (ii) above? Circle *one or more* of A, B, C, or D; more than one may be correct.

A.
```
result = result.strip()
```

B.
```
result.replace(" ", None)
```

C.
```
result = result.lstrip().rstrip()
```

D.
```
result = result.replace(" ", "")
```

(d) Which of the following code segments correctly implements **both steps (i) and (ii)** above? Your choices are the same code segments (A, B, C, and D) as in part (b) above, but the correct answer(s) may be different. Circle *one or more* of A, B, C, or D; more than one may be correct.

A. B. C. D.

(e) Which of the following code segments correctly implements step (iii) above? Circle *one or more* of A, B, C, or D; more than one may be correct.

A.
```
if len(result) > 10 and result[0] == '1':
    result = result[1:]
```

B.
```
if len(result) > 10 and result[0] == '1':
    result = result - 1
```

C.
```
if len(result) > 10 and result[0] == '1':
    del result[0]
```

D.
```
if len(result) > 10 and result[0] == '1':
    result = result - '1'
```

Problem 2 (18 points)

A government agency has asked your help in building a system for analyzing telephone call records. You represent a single phone call as follows:

```
Call = namedtuple('Call', 'num_from num_to date time seconds')
```

where `num_from` and `num_to` are strings representing 10-digit US phone numbers (the number of the caller—the person who dialed the phone—is `num_from`; the number dialed—the number of the person who says "Hello?"—is `num_to`), date is a string representing the date, time represents the time the call was made, and second represents how long the call lasted, in seconds.

```
c1 = Call('7142220000', '9494440000', '01/23/2014', '12:32', 240)
c2 = Call('7142220000', '8189990000', '01/23/2014', '12:37', 200)
c3 = Call('3132221111', '2121112222', '01/24/2014', '08:25', 1600)
all_calls = [c1, c2, c3]
```

(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. (Remember that `[Call]` as a type annotation is the same as `'list of Call'`.)

```
def calls_from(call_list: [Call], phone_number: str) -> [Call]:
    """ Return a list of all calls made from the specified number
    """
    result = [ ]

    for c in _____________:

        if c._____________ == _____________:

            _____________.append(_____________)
    return result

assert calls_from(all_calls, '7142220000') == [c1, c2]
assert calls_from(all_calls, '3132221111') == [c3]
assert calls_from([ ], '3132221111') == [ ]
assert calls_from(all_calls, '9999999999') == [ ]
```

(b) (2 points) Next, define a similar function named `calls_to()` that returns calls made to a specified number, as described below. The body of `calls_to()` could be identical to the body of `calls_from()` above, except for one identifier in one place. Fill in the changed identifier below and leave the other places blank.

```
def calls_to(call_list: [Call], phone_number: str) -> [Call]:
    """ Return a list of all calls made to the specified number
    """
    result = [ ]

    for c in _____________:

        if c._____________ == _____________:

            _____________.append(_____________)
    return result

assert calls_to(all_calls, '7142220000') == [ ]
assert calls_to(all_calls, '2121112222') == [c3]
assert calls_to([ ], '3132221111') == [ ]
```

(c) (4 points) We plan to use this software to identify people who call the numbers of known terrorists (or receive calls from terrorists' numbers). Next we'll write a version of `calls_from()` that selects calls from any phone number on a list of numbers (rather than from just a single number). 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 calls_from_any(call_list: [Call], suspected_numbers: [str]) -> [Call]:
    """ Return a list of all calls made from a number on the suspected_numbers list.
    """
    result = [ ]
    for c in _____________:
        if c._____________  _____________  _____________:
            _____________.append(_____________)
    return result

assert calls_from_any(all_calls, ['7142220000', '3132221111', '7778881111']) == [c1, c2, c3]
```

(d) (2 points) Next, define a function named `calls_to_any()` that returns a list of all calls made *to* a number on the suspected-number list. The body of `calls_to_any()` could be identical to the body of `calls_from_any()`, except for one identifier in one place. Fill in the changed identifier below and leave the other places blank.

```
def calls_to_any(call_list: [Call], suspected_numbers: [str]) -> [Call]:
    """ Return a list of all calls made to a number on the suspected_numbers list.
    """
    result = [ ]
    for c in _____________:
        if c._____________  _____________  _____________:
            _____________.append(_____________)
    return result

assert calls_to_any(all_calls, ['7142220000', '8189990000', '2121112222']) == [c2, c3]
```

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

```
def total_call_time(call_list: [Call]) -> int:
    """ Return the total duration of all calls on list (in seconds)
    """
    result = _____________
    for c in _____________:
       _____________ += _____________ . _____________
    return _____________

assert total_call_time(all_calls) == 2040
assert total_call_time([c1]) == 240
assert total_call_time([ ]) == 0
```

Problem 3 (5 points)

(a) (3 points) The function below prints all the calls from a given number in a formatted table:

```
From       To         Date       Time  Seconds  Minutes

7142220000 9494440000 01/23/2014 12:32     240    4.000
7142220000 8189990000 01/23/2014 12:37     200    3.333
```

```
def print_calls_from(whole_call_list: [Call], phone_number: str) -> None:
    """ Print all the calls from the specified number in table form.
    """
    selected_calls = calls_from(whole_call_list, phone_number)
    print('From       To         Date       Time  Seconds  Minutes')
    print()

    FORMAT_STRING =  — Choose the correct value from A–E below —

    for c in selected_calls:
        print(FORMAT_STRING.format(c.num_from, c.num_to, c.date, c.time,
              c.seconds, c.seconds/60))
```

Choose which one of these five values could go in the code above to produce the correct formatting. Circle one of A, B, C, D, or E; only one is correct.

A. `"{:15s} {:15s} {:15s} {:5s} {:7d} {:8.3f}"`

B. `"{:10s} {:10s} {:15s} {:5s} {:1d} {:2.3f}"`

C. `"{:10s} {:10s} {:10s} {:5s} {:1d} {:8.3f}"`

D. `"{:10s} {:10s} {:10s} {:5s} {:7d} {:8.3f}"`

E. `"{:10s} {:10s} {:10s} {:5s} {:7d} {:2.3f}"`

(b) (2 points) Which of the following is the correct output from this print statement:

```
print("Call time is {:2d} seconds.".format(185))
```

Circle one of A, B, C, D, or E; only one is correct.

A. `Call time is 18 seconds.`

B. `Call time is 85 seconds.`

C. `Call time is185 seconds.`

D. `Call time is 185 seconds.`

E. `Call time is  185 seconds.`

Problem 4 (20 points)

The previous problems dealt with a list containing every call. It will be convenient to organize those calls by customer (i.e., by the person who "owns" each phone number), with a record (namedtuple) for each person. Each record starts out as follows:

```
Customer = namedtuple('Customer', 'name address phone outgoing incoming')
p1 = Customer('Jones, John', '12 Elm St.', '9494440000', [ ], [ ])
p2 = Customer('Smith, Sally', '24 Oak St.', '7142220000', [ ], [ ])
p3 = Customer('Roberts, Dan', '36 Ash St.', '2137770000', [ ], [ ])
p4 = Customer('Brown, Mary', '48 Fir St.', '8189990000', [ ], [ ])
```

where the name, address, and phone are strings. The last two fields are lists: a list of all the (outgoing) Calls made from the customer's number and a list of all the (incoming) Calls made to the customer's number.

(a) (5 points) First we write a function that completes the outgoing and incoming call lists for a single customer. 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. As always, for full credit you should use previously defined functions wherever appropriate.

```
def fill_customer_record(RWC: Customer, all_calls: [Call]) -> Customer:
    """ "RWC" stands for "record without calls":  It's quicker to write on the exam.
        Return a customer record with that customer's outgoing and incoming calls
        filled in.
    """
    outgoing_call_list = _____________ (_____________, RWC.phone)
    incoming_call_list = _____________ (_____________, RWC.phone)
    return Customer(RWC._____________, RWC._____________, _____________._____________,
        outgoing_call_list, incoming_call_list)

assert fill_customer_record(p2, all_calls) == Customer(p2.name, p2.address, p2.phone, [c1, c2], [ ])
```

(b) (6 points) Suppose we have (i) the functions and namedtuples described above in this exam, (ii) the variable `all_calls` that contains a list of Calls containing every call made in the last year, and (iii) the variable `terrorist_phones` that contains `['7778889999', '6667778888', '5556667777']`, representing three phone numbers of known terrorists.

Which of the following is an accurate statement? Circle *one or more* of A, B, C, D, E, or F; more than one may be correct.

A. Each Call in `all_calls` occurs twice in the list of Customers constructed from `all_calls`: once on the caller's outgoing list and once on the recipient's incoming list.

B. On a given Customer's outgoing call list, every Call's `num_from` field is the same.

C. On a given Customer's incoming call list, every Call's `num_from` field is the same.

D. We can use `calls_to_any(all_calls, terrorist_phones)` to get a list of all the calls made to the terrorists' phone numbers.

E. The expression `total_call_time(all_calls, '7778889999')` gives us the number of seconds the suspected terrorist has spent on the phone.

F. The expression `total_call_time(calls_to(all_calls, '8189990000'))` gives us the number of seconds Mary Brown has spent on the phone with people who called her.

(c) (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 terrorist_talk_time(c: Customer, bad_phones: [str]) -> int:
    """ Return the number of seconds this customer spent talking to terrorists
        on the specified list of phone numbers.
    """

    seconds_outgoing = _____________(_____________(c._____________, bad_phones))

    seconds_incoming = _____________(_____________(c._____________, bad_phones))

    return _____________ + _____________

assert terrorist_talk_time(p2, ['8189990000']) == 200
```

(d) (4 points) Suppose we complete our customer list as shown below.

```
p1 = fill_customer_record(p1, all_calls)
p2 = fill_customer_record(p2, all_calls)
p3 = fill_customer_record(p3, all_calls)
p4 = fill_customer_record(p4, all_calls)
customer_list = [p1, p2, p3, p4]
```

Indicate the data type of each of the following expressions by selecting one item from the provided list.

A. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list`

B. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1]`

C. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1].name`

D. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1].outgoing`

E. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1].outgoing[1]`

F. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1].outgoing[1].num_to`

G. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`customer_list[1].outgoing[1].seconds`

H. ❑int ❑float ❑bool ❑str ❑list of str ❑Call ❑Customer ❑list of Call ❑list of Customer
`len(customer_list[1].outgoing)`

Problem 5 (3 points)

Each of the three code segments below could be rewritten to have the same meaning in less code. Rewrite each segment, primarily by removing unnecessary code as discussed on the lab assignments, while preserving the meaning. Just make your changes on the code below.

(a)

```
def is_negative(num: int) -> bool:
    """ Return True if the parameter is a negative number and False otherwise
    """
    if num < 0:
        return True
    else:
        return False
```

(b)

```
if is_negative(-23) == True:
    print('Negative')
else:
    print('Positive (or zero)')
```

(c)

```
assert is_negative(-18) == True
```

Problem 6 (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.