Second Midterm

You have 75 minutes (until the end of the class period) to complete this exam. There are 55 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 (4 points)  Topic: Python expressions and data types

Use the following definitions in this problem:

\[ s = 'We the people of the United States, in order to form a more perfect union' \]

Below are eight segments of code, each with a part underlined. Indicate the data type of each underlined part by checking the appropriate box.

(a)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{print(s[1])}  \quad \# \text{ int} \quad \text{SCORING: 1/2 point each}
\]

(b)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{print(s[1:4])}  \quad \# \text{ str}
\]

(c)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{for } x \text{ in range(len(s))}: \quad \# \text{ func}
\quad \text{if } s[x] == ' ': \\
\quad \quad \text{print(s[x])}
\]

(d)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{for } x \text{ in range(len(s))}: \quad \# \text{ int}
\quad \text{if } s[x] == ' ': \\
\quad \quad \text{print(s[x])}
\]

(e)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{for } x \text{ in s}: \quad \# \text{ str}
\quad \text{if } x == ' ': \\
\quad \quad \text{print(x)}
\]

(f)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{for } x \text{ in s}: \quad \# \text{ bool}
\quad \text{if } x == ' ': \\
\quad \quad \text{print}(x)
\]

(g)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{result} = 0 \\
\text{for } n \text{ in L:} \\
\quad \text{result} += n \\
\text{assert(result > 0)} \quad \# \text{ bool}
\]

(h)  
\[
\begin{array}{c}
\text{int}  \\
\text{float}  \\
\text{bool}  \\
\text{str}  \\
\text{function}  \\
\text{list}
\end{array}
\]
\[
\text{print(L[3:5])} \quad \# \text{ list}
\]
Problem 2 (11 points) **Topic: Types of combined data structures**

Use the following definitions in this problem:

```python
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, Paul', '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.

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

StudentBody  
# list of Student  SCORING: 1/2 point each

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

StudentBody[2]  
# Student

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

bio97  
# Course

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

StudentBody[0].studylist  
# list of Course

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

StudentBody[2].name  
# str, value Programmer, Paul

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

sX  
# Student

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

StudentBody[1].studylist[0]  
# Course
(h) | int | float | bool | str | function | Course | Student | list of Course | list of Student
sX.level # str, value SO

(i) | int | float | bool | str | function | Course | Student | list of Course | list of Student
StudentBody[3].studylist[0].title # str, value "Programming with Libraries"

(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.

StudentBody[2].name # str, value Programmer, Paul  SCORING: 1 point each (for value)

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

For this problem, use these definitions:

\[ L = ['King', 'Lincoln', 'Washington', 'Chavez'] \]
\[ M = [100, 20, 7, 3000, 1] \]

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  \(\rightarrow\) D

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

(b) Circle one:  A  B  C  D  E  F  G  H  I  \(\rightarrow\) A

```python
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  \(\rightarrow\) E

```python
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  \(\rightarrow\) H

```python
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.

```
King 4
Lincoln 7
Washington 10
Chavez 6
Done 4
```

E.

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

F.

```
King 4
Lincoln 4
Washington 4
Chavez 4
Done 4
```

G.

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

H.

```
K King
i King
n King
g King
Done 4
```

I.

```
K K
i Ki
n Ki
g King
Done 0
```
Problem 4  (3 points)  **Topic: String formatting**

Here are some statistics on movies nominated for Academy Awards:

<table>
<thead>
<tr>
<th>Movie</th>
<th>Income</th>
<th>Nominations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lincoln</td>
<td>$176.0</td>
<td>12</td>
</tr>
<tr>
<td>Argo</td>
<td>$127.1</td>
<td>7</td>
</tr>
<tr>
<td>Skyfall</td>
<td>$303.2</td>
<td>5</td>
</tr>
<tr>
<td>Django Unchained</td>
<td>$157.3</td>
<td>5</td>
</tr>
<tr>
<td>Dark Knight Rises</td>
<td>$447.4</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

```python
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 (6 points) **Topic: Processing lists of namedtuples**

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

```python
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) (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.

```python
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)))
```

**SCORING:** 1/2 point for both depts, 1/2 point for both nums, 1 point for ==, 1/2 for c1, 1/2 for c2

(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.

```python
def Course_on_studylist(c: Course, SL: 'list of Course') -> bool:
    ''' Return True if the course c equals any course on the list SL (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 SL:
       if Course_equals(c, a_course):
           result = True
   return result

B. for a_course in SL:
   if Course_equals(c, a_course):  ### THIS ONE
       return True
   return False
C. for a_course in SL:  ## NO
    if Course_equals(c, a_course):
        return True
    return False

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

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

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

Problem 6  (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, Paul', 'SR', 'PSB', [ics32, wr39b, 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 __________(__________.__________)                   SCORING: 1 point per blank

#   return len(S.studylist)
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.

```python
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.

```python
def Student_units(S: Student) -> float:
    ''' Return the total number of units this student is enrolled in'''
    total = __________
    for c in S.__________:
        total += __________.__________
    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.

```python
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 __________:
        total += __________(__________)  
        total += Student_units(s)
    if len(SB) == 0:
        return 0
    else:
        return __________ __________ __________ (__________)  
    return total / len(SB)  
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)
```