This 90-minute exam has 6 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```python
def foo():
    if something:
        do something
        do more things
        do something last
```

You should not use loops or recursion on this exam. Beyond that, you may use any Python feature that you have learned in class (if-statements, try-except, lists), unless directed otherwise.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

The Important First Question:

1. [2 points] Write your last name, first name, and netid, at the top of each page.
Reference Sheet

Throughout this exam you will be asked questions about strings and lists. You are expected to understand how slicing works. In addition, the following functions and methods may be useful.

### String Functions and Methods

<table>
<thead>
<tr>
<th>Expression or Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(s)</code></td>
<td>Returns: number of characters in s; it can be 0.</td>
</tr>
<tr>
<td><code>a in s</code></td>
<td>Returns: True if the substring a is in s; False otherwise.</td>
</tr>
<tr>
<td><code>s.count(s1)</code></td>
<td>Returns: the number of times s1 occurs in s</td>
</tr>
<tr>
<td><code>s.find(s1)</code></td>
<td>Returns: index of the first character of the FIRST occurrence of s1 in s (-1 if s1 does not occur in s).</td>
</tr>
<tr>
<td><code>s.find(s1,n)</code></td>
<td>Returns: index of the first character of the first occurrence of s1 in s STARTING at position n. (-1 if s1 does not occur in s from this position).</td>
</tr>
<tr>
<td><code>s.isalpha()</code></td>
<td>Returns: True if s is not empty and its elements are all letters; it returns False otherwise.</td>
</tr>
<tr>
<td><code>s.isdigit()</code></td>
<td>Returns: True if s is not empty and its elements are all numbers; it returns False otherwise.</td>
</tr>
<tr>
<td><code>s.isalnum()</code></td>
<td>Returns: True if s is not empty and its elements are all letters or numbers; it returns False otherwise.</td>
</tr>
<tr>
<td><code>s.islower()</code></td>
<td>Returns: True if s is has at least one letter and all letters are lower case; it returns False otherwise (e.g. ‘a123’ is True but ‘123’ is False).</td>
</tr>
<tr>
<td><code>s.isupper()</code></td>
<td>Returns: True if s is has at least one letter and all letters are upper case; it returns False otherwise (e.g. ‘A123’ is True but ‘123’ is False).</td>
</tr>
</tbody>
</table>

### List Functions and Methods

<table>
<thead>
<tr>
<th>Expression or Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(x)</code></td>
<td>Returns: number of elements in list x; it can be 0.</td>
</tr>
<tr>
<td><code>y in x</code></td>
<td>Returns: True if y is in list x; False otherwise.</td>
</tr>
<tr>
<td><code>x.count(y)</code></td>
<td>Returns: the number of times y occurs in x</td>
</tr>
<tr>
<td><code>x.index(y)</code></td>
<td>Returns: index of the FIRST occurrence of y in x (an error occurs if y does not occur in x).</td>
</tr>
<tr>
<td><code>x.index(y,n)</code></td>
<td>Returns: index of the first occurrence of y in x STARTING at position n (an error occurs if y does not occur in x).</td>
</tr>
<tr>
<td><code>x.append(y)</code></td>
<td>Adds y to the end of list x.</td>
</tr>
<tr>
<td><code>x.insert(i,y)</code></td>
<td>Inserts y at position i in list x, shifting later elements to the right.</td>
</tr>
<tr>
<td><code>x.remove(y)</code></td>
<td>Removes the first item from the list whose value is y (an error occurs if y does not occur in x).</td>
</tr>
</tbody>
</table>

The last three list methods are all procedures. They return the value `None`. 
2. [14 points total] **Short Answer Questions.**

   (a) [4 points] Name the four types of variables we have seen in class. Describe each one.

   (b) [3 points] What is a *parameter*? What is an *argument*? How are they related?

   (c) [4 points] What is the difference between a *function definition* and a *function call*? Give an example of each.
(d) [3 points] Consider the code below. What is printed out when the code is executed?

```python
x = 2
try:
    print('Part A')
    assert x < 0, 'Failure'
    print('Part B')
except:
    print('Part C')
print('Part D')
```

3. [20 points] **Call Frames.**

Consider the following (unspecified) function definitions.

```python
1  def trimit(b):
2      a = b[0]
3      return b[a:-a]
4  
5  def kneadit(a):
6      a[0] = a[-1]
7      b = trimit(a)
8      return b
```

Assume that `b = [4,2,1]` is a global variable referencing a list in the heap, as shown below. On the next two pages, diagram the evolution of the call

```python
a = kneadit(b)
```

Diagram the state of the entire call stack for the function `kneadit` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of eight diagrams, not including the (pre-call) diagram shown.

You should draw also the state of global space and the heap at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. You are also allowed to write “unchanged” if no changes were made to either global space or the heap.
<table>
<thead>
<tr>
<th>Call Stack</th>
<th>Global Space</th>
<th>The Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call Stack</td>
<td>Global Space</td>
<td>The Heap</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. [20 points] **String Slicing.**

Implement the function specified below. You may need to use several of the functions and methods on the reference page. **Pay close attention to the precondition, as it will help you** (e.g. only numbers less than 1,000,000 are possible with that string length).

```python
def valid_format(s):
    """Returns True if s is a valid numerical string; it returns False otherwise.

    A valid numerical string is one with only digits and commas, and commas only appear before every three digits. In addition, a valid string only starts with a 0 if it has exactly one character.

    Example: valid_format('12') is True
              valid_format('apple') is False
              valid_format('1,000') is True
              valid_format('1000') is False
              valid_format('10,00') is False
              valid_format('0') is True
              valid_format('012') is False

    Precondition: s is a nonempty string with no more than 7 characters""
```
5. [22 points total] **Testing and Debugging.**

(a) [9 points] The function `anglitime` takes a string representing a unit of time (in hours and minutes) and expands it into words with format 'hours, minutes'. One minute or one hour is singular, while all other amounts (including zero) are plural. So `anglitime('23:01')` returns 'twenty three hours, one minute', while `anglitime('1:45')` returns 'one hour, forty five minutes'. The call `anglitime('00:00')` returns 'zero hours, zero minutes'. There are at least three bugs in the code below. These bugs are potentially spread across multiple functions. To help find the bugs, we have added several print statements throughout the code, and show the results on the next page. Using this information as a guide, identify and fix the three bugs on the next page. You should explain your fixes.

```python
1 def anglitime(time):
2     """Returns full english word for time
3     See above for explanation of the results."
4     print('Colon at ' + repr(pos))  # WATCH
5     hours = int(time[:pos])
6     if digit2 == 0:
7     return tens(dig1)
8     hrword = wordify(hours)+suff
9     pos = time.find(':')
10    print('Hrs are ' + repr(hours))  # WATCH
11    minis = int(time[pos+1:])
12    print('Min are ' + repr(minis))  # WATCH
13    susf = ' hours'
14    if hours == 1:
15         print('Singular hour')
16         susf = ' hour'
17         hrword = wordify(hours)+susf
18         print('Hrs are ' + repr(hrword))  # WATCH
19    susf = ' minutes'
20    if hours == 1:
21         print('Singular minute')
22         susf = ' minute'
23         mmword = wordify(minis)+susf
24         print('Min are ' + repr(mmword))  # WATCH
25     return hrword+susf, '+mmword

26 def tens(n):
27     """Returns word for the (one digit) number n
28     Precond: n an int, 0 <= n <= 9"
29     names = ['zero', 'one', 'two', 'three',
30              'four', 'five', 'six',
31              'seven', 'eight', 'nine']
32     print('ones n is ' + repr(n))  # WATCH
33     return names[n]

34 def teens(n):
35     """Returns word for the ten number n
36     Precond: n an int, 10 <= n <= 19"
37     names = ['eleven', 'twelve', 'thirteen',
38              'fourteen', 'fifteen', 'sixteen',
39              'seventeen', 'eighteen', 'nineteen']
40     print('teens n is ' + repr(n))  # WATCH
41     return names[n-10]

42 def ones(n):
43     """Returns word for the (one digit) number n
44     Precond: n an int, 0 <= n <= 9"
45     names = ['zero', 'one', 'two', 'three',
46              'four', 'five', 'six',
47              'seven', 'eight', 'nine']
48     print('ones n is ' + repr(n))  # WATCH
49     return names[n]

43 def wordify(n):
44     """Returns english word for a number
45     Precond: n an int, 0 <= n <= 100"
46     dig1 = n // 10
47     dig2 = n % 10
48     if dig2 == 0:
49         return tens(dig1)
50         return tens(dig1)+' ' + ones(dig2)
51     if n >= 10:
52         return teens(n)
53     else:
54         return ones(n)
```

Tests:

```python
>>> anglitime('9:01')
Colon at 1
Hrs are 9
Min are 1
ones n is 9
Hrs are 'nine hours'
one n is 1
Min are 'one minutes'
'nine hours, one minutes'
```

Second Bug:

```python
>>> anglitime('05:13')
Colon at 2
Hrs are 5
Min are 13
ones n is 5
Hrs are 'five hours'
teens n is 13
Min are 'fourteen minutes'
'five hours, fourteen minutes'
```

Third Bug:

```python
>>> anglitime('24:00')
Colon at 2
Hrs are 24
Min are 0
dig1 is 2.4
dig2 is 4
tens n is 2.4
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "debug.py", line 19, in anglitime  
    hrword = wordify(hours)+suff  
  File "debug.py", line 80, in wordify  
    return tens(dig1)+' '+ones(dig2)  
  File "debug.py", line 40, in tens  
    return names[n-2]  
TypeError: list indices must be integers
```
(b) [8 points] On the previous page you saw three different tests for `anglitime`. Below, write six more test cases for this function. By a test case, we just mean an input and an expected output; you do not need to write an `assert_equals` statement. Each test case needs to be different, which in this case means it executes a different flow through the code. In addition, your tests must be different from the three test cases on the previous page. For each test case, explain why it is different.

(c) [5 points] The function specified below is similar to `wordify`, except that it has a different precondition. Using assert statements, enforce the precondition of this function. Error messages are not required.

```python
def wordify_minutes(mins):
    """Returns full english word for minutes mins

    Precond: mins is a string 'mm' representing an int in 0..59"
```

Page 10
6. [22 points total] **Objects and Functions.**

As you are probably aware, angles can be measured in either degrees or radians. There are $180^\circ$ in $\pi$, making conversion between the two easy. However, there is more than one way to specify degrees. We can specify degrees as decimals, like $75.3^\circ$. Or we can break up that same value into degrees and *minutes* as follows: $75^\circ 18'$ (the ' is for minutes).

There are 60 minutes to a degree, just as with minutes and hours. For heightened accuracy, we can further divide each minute into seconds. However, for simplicity, we will stop at minutes but all minutes to be decimals as follows: $75^\circ 18.21'$. To implement this, we create an `Angle` class with the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>degrees</td>
<td>the angle degrees</td>
<td>int value between 0 and 359 (inclusive)</td>
</tr>
<tr>
<td>minutes</td>
<td>the minutes of the degree</td>
<td>float value between 0 and 60 (excluding 60.0)</td>
</tr>
</tbody>
</table>

(a) [10 points] Implement the function below according to the specification.

**Hint:** You might want to convert the values to decimal degrees and back. In addition, remember that degrees “wrap around” so that $-15^\circ$ is really $345^\circ$, and $543^\circ$ is really $183^\circ$.

```python
def subtract(angle1, angle2):
    """MODIFIES angle1 to be the result of subtracting angle2

    This function is a procedure and does not return a value.

    Example: If angle1 is 123 d 34.5 m and angle2 is 75 d 54.3 m
    then subtract(angle1, angle2) changes angle1 to 47 d 40.2 m.

    Preconditions: angle1 and angle2 are Angle objects"
```

Page 11
(b) [12 points] The bisection of two angles is the angle in the middle of the arc created by moving clockwise from the first angle. This is shown in the picture below.

You can compute the bisection by averaging (adding and dividing by 2) the two angles, assuming that the first angle is larger. The tricky part is when the angles “wrap around” so that the first angle is actually less (in terms of degrees) that than the second angle. This is shown above on the right. To address this case, you have to do something to make the first angle larger. The hints from the previous problem can help you here. Using this guidance, implement the function below according to the specification.

```
def bisect(angle1, angle2):
    """Returns the angle bisecting sector from angle1 to angle2 (clockwise).

    Example: If angle1 is 123 d 34.5 m and angle2 is 75 d 54.3 m
    then bisect(angle1, angle2) returns 99 d 44.4 m, while (on the
    other hand) bisect(angle2, angle1) returns 279 d 44.4 m.

    Preconditions: angle1 and angle2 are Angle objects"
"
```