Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 1 October 17th, 2019

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:

```
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**.

Question	Points	Score
1	2	
2	14	
3	20	
4	20	
5	22	
6	22	
Total:	100	

The Important First Question:

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

Last Name: _____ First: _____ Netid: _____

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

Expression or Method	Description
`len(s)`	**Returns**: number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`s.count(s1)`	**Returns**: the number of times `s1` occurs in `s`
`s.find(s1)`	**Returns**: index of the first character of the FIRST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).
`s.find(s1,n)`	**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).
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all numbers; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.
`s.islower()`	**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).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are uppper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).

List Functions and Methods

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

The last three list methods are all procedures. They return the value `None`.

Last Name: _____ First: _____ Netid: _____

2. [14 points total] **Short Answer Questions**.

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

- Global variables – variables assigned outside of function definition
- Local variables – variables assigned in a function definition
- Parameters – variables in a function header
- Attributes – variables in an object folder

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

A *parameter* is a variable in the parentheses at the start of a function definition.
An *argument* is an expression in the parentheses of a function call.
A function call evaluates the arguments and plugs the result into the parameters before executing the function body.

(c) [4 points] What is the difference between a *function definition* and a *function call*? Give an example of each.

A *function definiton* is code that describes what a function should do when it is executed (but does not execute that code). It starts with the keyword `def` and a function header. Lines 1-3 of `trimit` on the next page are a function definition. A *function call* is an expression that instructs Python to execute the function. Line 7 of `kneadit` has an example of a function call: `b = trimit(a)`.

(d) [3 points] Consider the code below. What is printed out when the code is executed?

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

While the assert causes the program to crash, the error message is never displayed since the code recovers in the except. Hence the output is

```
Part A
Part C
Part D
```

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

Consider the following (unspecified) function definitions.

```
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

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

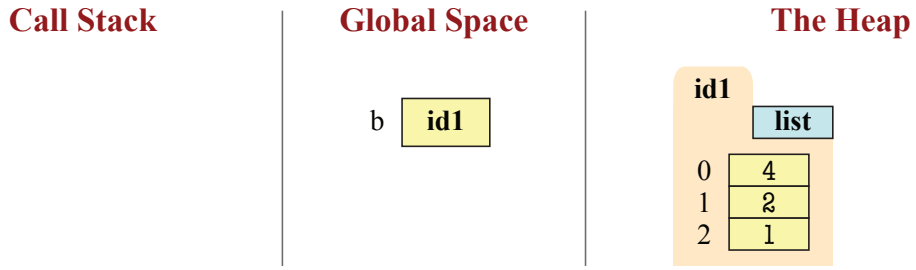
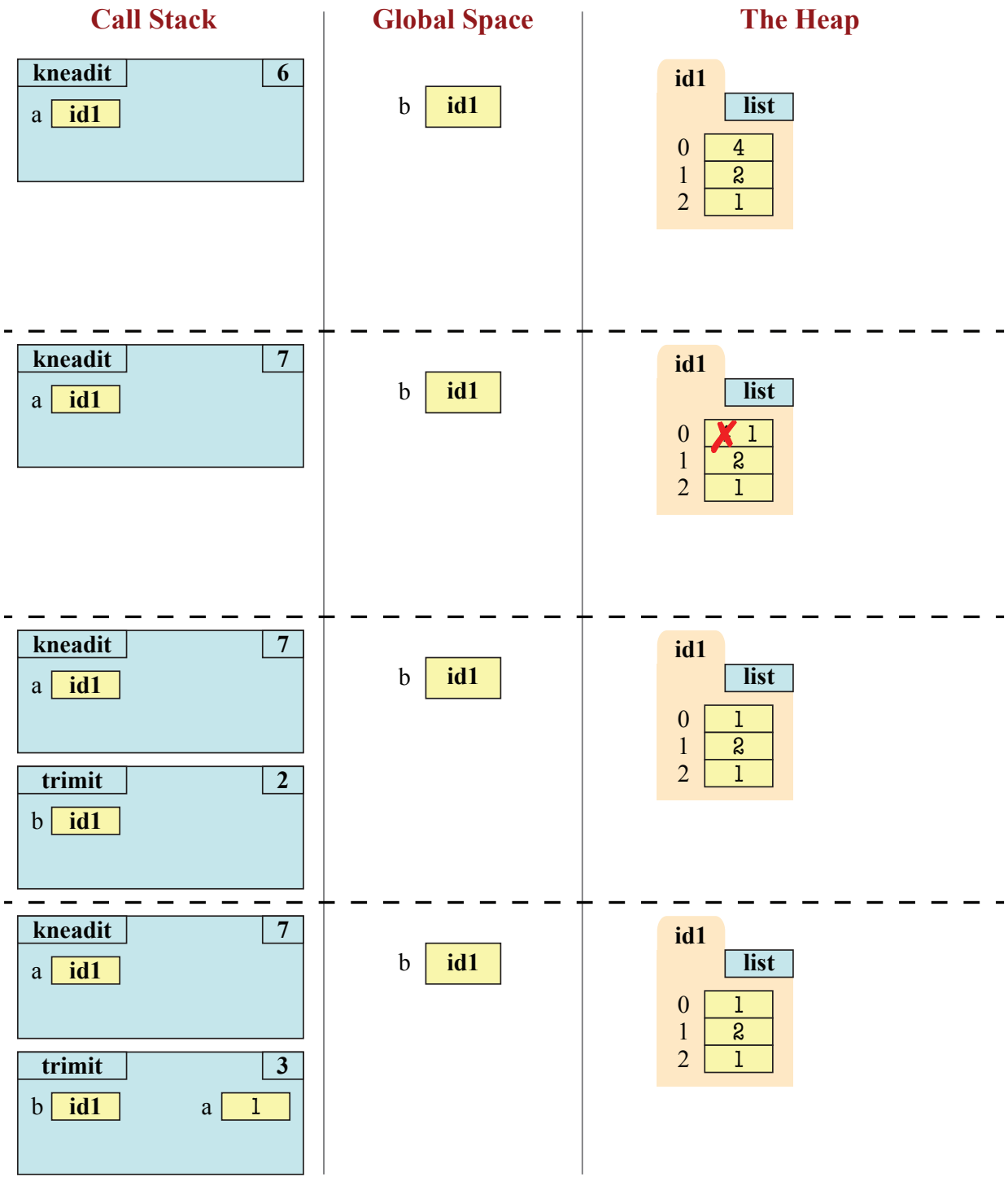


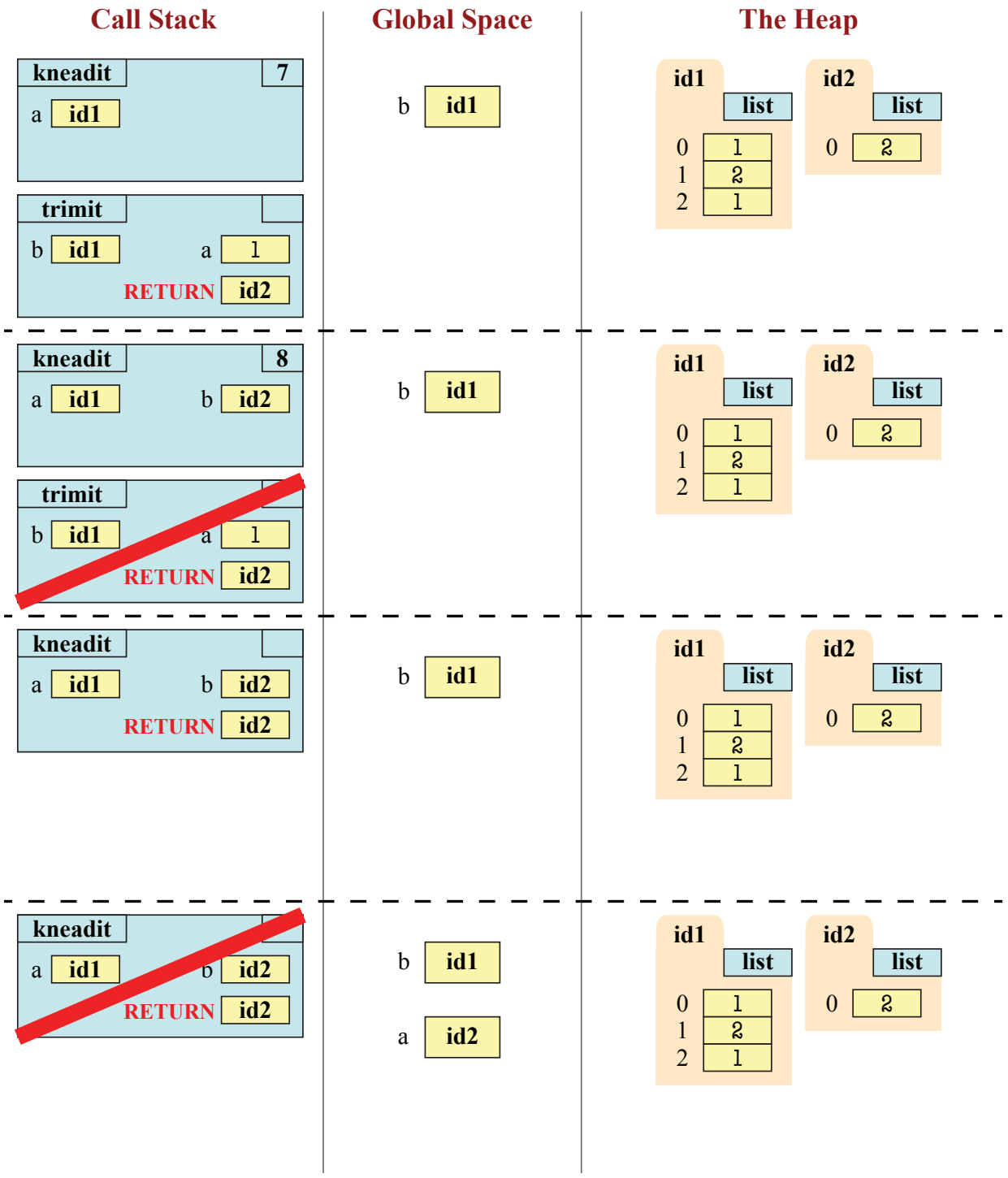
Diagram the state of the *entire call stack* for the function `kneadit` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **eight** diagrams, not including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

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

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).

```
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"""
    # First deal with the leading 0
    if s[0] == '0':
        return s == '0'

    # Check if a comma is possible
    if len(s) <= 3:
        result = s.isdigit()
    elif sp[-4] != ',':
        # The comma must be before the last three numbers
        result = False
    else:
        # Verify the other six characters are all numbers
        result = s[:-4].isdigit() and s[-3:].isdigit()

    return result
```

Last Name: _____ First: _____ Netid: _____

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.

```
1  def anglitime(time):
2      """Returns full english word for time
3
4      See above for explanation of the results.
5
6      Precond: time a string 'hh:mm' or 'h:mm'
7      where h, m digits. mm are in 0..59."""
8      pos = time.find(':')
9      print('Colon at '+repr(pos))       # WATCH
10     hours = int(time[:pos])
11     print('Hrs are '+repr(hours))      # WATCH
12     minis = int(time[pos+1:])
13     print('Min are '+repr(minis))      # WATCH
14
15     suff = ' hours'
16     if hours == 1:
17         print('Singular hour')         # TRACE
18         suff = ' hour'
19     hrword = wordify(hours)+suff
20     print('Hrs are '+repr(hrword))     # WATCH
21     suff = ' minutes'
22     if hours == 1:
23         print('Singular minute')       # TRACE
24         suff = ' minute'
25     mnword = wordify(minis)+suff
26     print('Min are '+repr(mnword))     # WATCH
27     return hrword+', '+mnword
28
29
30 def tens(n):
31     """Returns word for the tens digit n
32
33     Ex: tens(3) returns 'thirty'
34
35     Precond: n an int, 2 <= n <= 9"""
36     names = ['twenty','thirty','forty',
37              'fifty','sixty','seventy',
38              'eighty','ninety']
39     print('tens n is '+repr(n))        # WATCH
40     return names[n-2]
41
42
43 def ones(n):
44     """Returns word for the (one digit) number n
45
46     Precond: n an int, 0 <= n <= 9"""
47     names = ['zero','one','two','three',
48              'four','five','six',
49              'seven','eight','nine']
50     print('ones n is '+repr(n))        # WATCH
51     return names[n]
52
53
54 def teens(n):
55     """Returns word for the teen number n
56
57     Ex: teens(10) returns 'ten'
58
59     Precond: n an int, 10 <= n <= 19"""
60     names = ['eleven','twelve','thirteen',
61              'fourteen','fifteen', 'sixteen',
62              'seventeen','eighteen','nineteen']
63     print('teens n is '+repr(n))       # WATCH
64     return names[n-10]
65
66
67 def wordify(n):
68     """Returns english word for a number
69
70     Ex: wordify(93) returns 'ninety three'
71
72     Precond: n an int, 0 <= n < 100"""
73     if n >= 20:
74         dig1 = n / 10
75         print('dig1 is '+repr(dig1))    # WATCH
76         dig2 = n % 10
77         print('dig2 is '+repr(dig2))    # WATCH
78         if digit2 == 0:
79             return tens(dig1)
80         return tens(dig1)+' '+ones(dig2)
81     elif n >= 10:
82         return teens(n)
83     else:
84         return ones(n)
```

Last Name: _____ First: _____ Netid: _____

Tests:

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

First Bug:

The trace statement for a singular minute is not printing even though it should. Looking at the conditional on line 22, we see that it is checking the wrong variable. We must change line 22 to

```
if minis == 1:
```

```
>>> 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'
```

Second Bug:

The function `teens` should be returning `'thirteen'`, not `'fourteen'`. At a first glance, the problem might seem to be off by one (e.g. subtract 11, not 10). However, in the specification, we clearly see that `'ten'` counts as a teen. So the bug is that `'ten'` is missing from the list of names. Line 60 should start

```
names = ['ten', 'eleven', 'twelve',
```

and so on.

```
>>> 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
```

Third Bug:

While the crash occurs in `tens`, it is clear from the last print statement that this is a precondition violation (the bug must be earlier). We want to pass an `int` to `tens`, not a float. There are many ways to fix this, but any fix must make sure that both lines 79 and 80 receive `int` values. The easiest solution is to use integer division – not float division – changing line 74 as follows:

```
dig1 = n // 10
```

There is (again) a fourth bug we missed. The variable `digit2` on line 78 should be `dig2`.

(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.

For this problem, we want each test to have a different path through the conditionals in `anglitime` and `wordify`. There are at least 25(!) different possible answers to this question. There are five different test cases for each of hour and minute (one digit plural, one digit singular, teens value, two digit with 0 in ones place, and proper two digit number), and you can mix and match each of these. You can also get more answers by adding or removing a leading 0 to the hours. Below are just a few examples.
In providing your answers, you need to be sure not to duplicate the tests from the previous page. The first test is 'plural one digit hours, singular one digit minutes'. The second is 'plural one digit hours, teen valued minutes' (even though there is a leading 0). The third is 'proper two digit hours, plural singular digit minutes'.

Input	Output	Reason
`'24:14'`	`'twenty four hours, fourteen minutes'`	Proper two digit hours, teen valued minutes
`'30:02'`	`'thirty hours, two minutes'`	Zeroed ones digit hours, plural one digit minutes
`'12:50'`	`'twelve hours, fifty minutes'`	Teen valued hours, zeroed ones digit mins
`'7:43'`	`'seven hours, forty three minutes'`	Plural one digit hours, proper two digit mins
`'1:15'`	`'one hour, fifteen minutes'`	Singular one digit hour, teen valued minutes
`'10:01'`	`'ten hours, one minute'`	Teen valued hours, singular one digit mins

(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.

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

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

    assert type(mins) == str
    assert len(mins) == 2
    assert mins.isdigit()
    assert 0 <= int(mins) and int(mins) <= 59
```

Last Name: _____ First: _____ Netid: _____

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

   As you are probably aware, angles can be measured in either degrees or radians. There are 180° in π, 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°. Or we can break up that same values in to degrees and *minutes* as follows: 75°18′ (the ′ is for minutes).

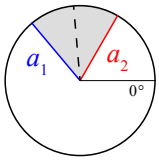
   There are 60 minutes to a degree, just as with minutes and hours. For heightened accuracy, we can further divide each minute into seconds. However, for simplicity, we will stop at minutes but all minutes to be decimals as follows: 75°18.21′. To implement this, we create an `Angle` class with the following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

Attribute	Meaning	Invariant
`degrees`	the angle degrees	`int` value between 0 and 359 (inclusive)
`minutes`	the minutes of the degree	`float` value between 0 and 60 (excluding 60.0)

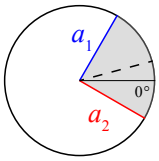
   (a) [10 points] Implement the function below according to the specification.
   Hint: You might want to convert the values to decimal degrees and back. In addition, remember that degrees "wrap around" so that −15° is really 345°, and 543° is really 183°.

```
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"""

    # Add degrees and minutes separately
    degrees = angle1.degrees-angle2.degrees
    minutes = angle1.minutes-angle2.minutes

    # Adjust minutes if out of bounds
    if minutes < 0:
        minutes = minutes+60
        degrees = degrees-1

    # Adjust degrees if out of bounds
    if degrees < 0:
        degrees = degrees+360

    # Modify the attributes of angle1
    angle1.degrees = degrees
    angle1.minutes = minutes
```

(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.



(a) Traditional Bisection



(b) Wrapped Bisection

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"""

    # Convert to decimal degrees
    degrees1 = angle1.degrees+angle1.minutes/60
    degrees2 = angle2.degrees+angle2.minutes/60

    # Make sure first angle is larger
    if (degrees1 < degrees2):
        degrees1 = degrees+360

    # Average and shift back in range
    degrees1 = (degrees1 + degrees2)/2
    if degrees1 > 360:
        degrees1 = degrees-360

    # Create the Angle object
    degrees = int(degrees1)
    minutes = (degrees1-degrees)*60
    return Angle(degrees,minutes)
```