

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.

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

List Functions and Methods

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

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.

(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?

```
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.

```
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)
```

Call Stack

Global Space

The Heap

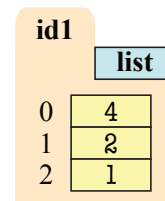
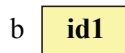


Diagram the state of the *entire call stack* for the function `kneadit` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **eight** diagrams, not including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

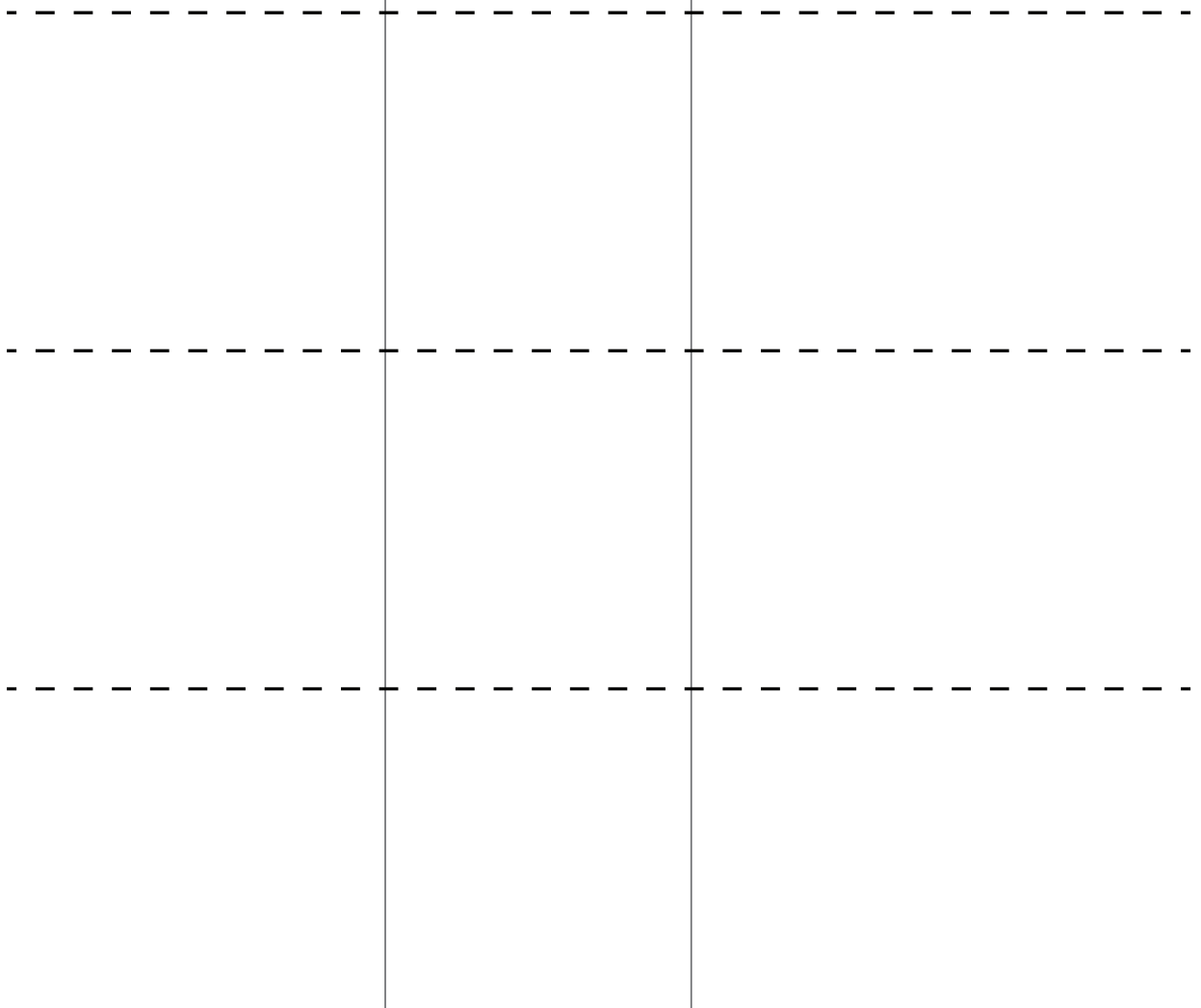
First: _____

Netid: _____

Call Stack

Global Space

The Heap



Last Name: _____

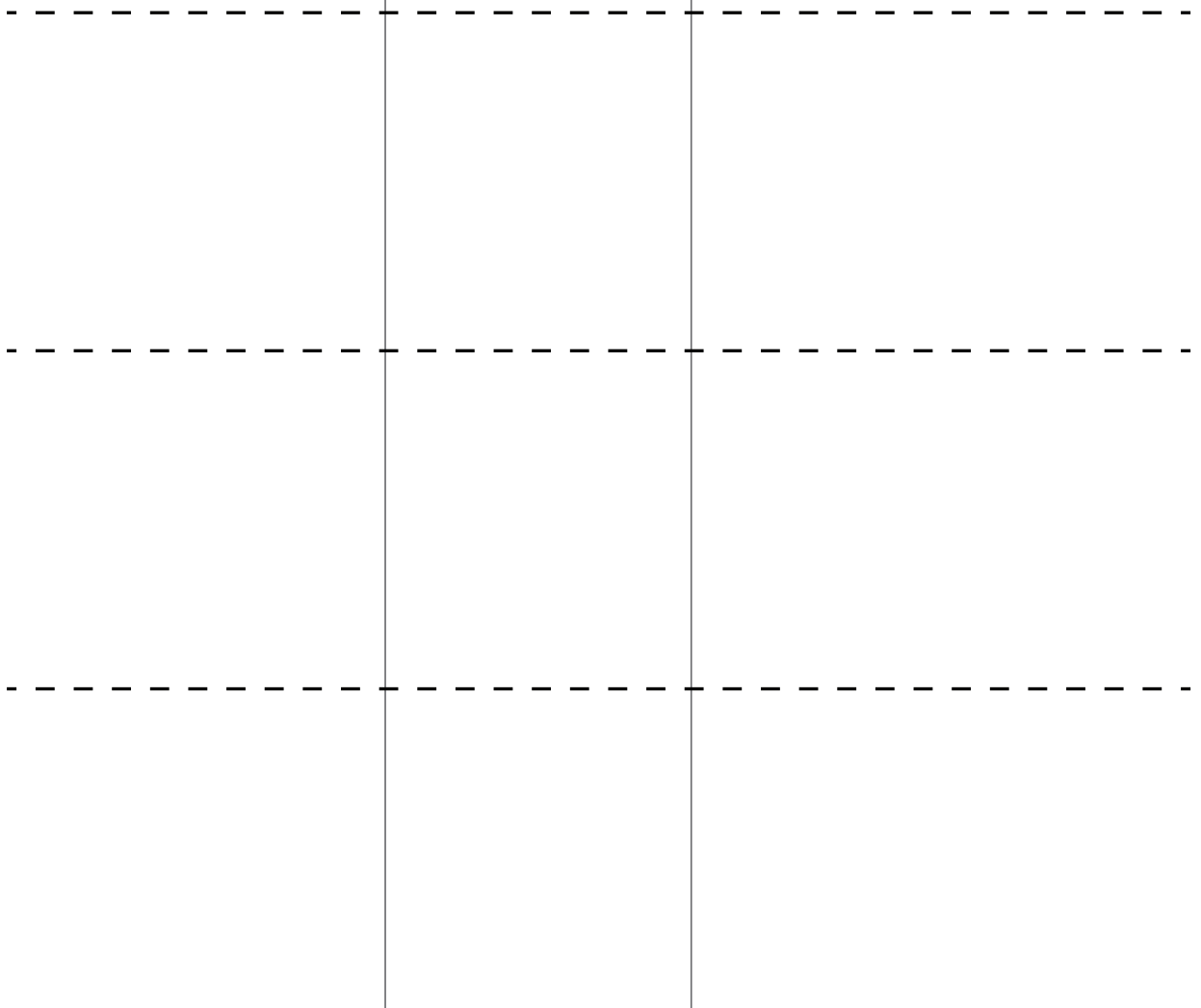
First: _____

Netid: _____

Call Stack

Global Space

The Heap



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"""
```

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:

```
>>> 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:

```
>>> 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:

Last Name: _____ First: _____ Netid: _____

- (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.

```
def wordify_minutes(mins):  
    """Returns full english word for minutes mins  
  
    Precond: mins is a string 'mm' representing an int in 0..59"""
```

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: $75^\circ 18'$ (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.

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

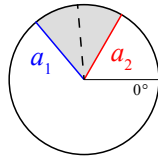
    This function is a procedure and does not return a value.

    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.

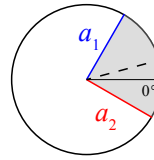
    Preconditions: angle1 and angle2 are Angle objects"""
```

Last Name: _____ First: _____ Netid: _____

- (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) 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"""
```