Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 1 October 16th, 2025

This 90-minute exam has 6 questions worth a total of 100 points. Read over 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; just make sure your name is on every page.

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	25	
5	19	
6	20	
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 upper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.lower()`	**Returns**: A copy of `s` with all letters lower case.
`s.upper()`	**Returns**: A copy of `s` with all letters upper case.

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) [3 points] What is a literal? Give an example.

(b) [4 points] What is a parameter? What is an argument? How are they related?

(c) [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')
```

(d) [4 points] Consider the code below. Is the code correct or will it produce an error? If it is correct, what value is put in the variable `y`? If it is not correct, explain the error.

```
import math

def foo(math):
    return math.cos(0)

y = foo(3)
```

Last Name: _____ First: _____ Netid: _____

3. [20 points] **String Slicing**.

   If you have ever downloaded DLC for a game, or redeemed a coupon online, you know that they are often defined as groups letters and numbers separated by dashes. The simplest variation has just one dash and groups its letters in numbers in blocks of four, like this: `K97J-FTRE`.

   Implement the function below, which takes an arbitrary string and determines whether it looks like a coupon code. You will need to use several of the functions and methods on the reference sheet. **Pay close attention to the specifications of these methods and functions. You may not use loops.** You do not need to assert preconditions..

```
def is_code(value):
    """Returns: True if value is a coupon code, otherwise False

    A code has the form XXXX-XXXX where XXXX is four characters
    that are each either an upper case letter or a number.

    Example: is_code('K97J-FTRE') is True
             is_code('K97J-9876') is True
             is_code('K97J') is False
             is_code('K97J-FTRE-9876') is False
             is_code('k97J-fTRe') is False
             is_code('K97J8-FTREK') is False

    Precondition: value is a string"""
```

Last Name: _____ First: _____ Netid: _____

4. [25 points total] **Testing and Debugging**.

   (a) [10 points] Consider the following function header and specification:

```
def shared_prefix(a, b):
    """Returns the longest common prefix of a and b.

    Example: shared_prefix('abcd','abgf') returns 'ab'

    Precondition: a and b are strings"""
```

   Do not implement this function. Instead, write down a list of at least **six test cases** that you would use to test out 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. For each test case *explain why it is significantly different from the others.*

   (b) [9 points] A Spoonerism is a word-play similar to the Pig Latin example from class. Given two words, we identify the *onset* of each word, which is the inital consonants up to the first vowel (what is left is called the *rime*). We then swap these two onsets. For example, the Spoonerism of `'grilled cheese'` is `'chilled greese'`. See the specification of the function `spoonerize` for examples about what to do if either the onset or rime is empty (which can happen if the word starts with a vowel or has no vowels).

   Note that Spoonerisms only apply to two words. If `spoonerize` is given a single word, it simply returns the word back. If it is more than two words, it only applies to the first two words. Again, this is clear in the examples for `spoonerize`.

   Examine the code for `spoonerize` and its helpers on the next page. There are ***at least* three bugs**. To help you find the bugs, we have added several print statements throughout the code, and show the results of these print statements on the page after the code. Using this information as a guide, identify and fix the three bugs on the answer page. **Your fixes may include more than one line of code**. You should explain your fixes.

Last Name: _____ First: _____ Netid: _____

```
def spoonerize(phrase):
    """Returns phrase with the onsets swapped

    This takes the first two words (if they
    exist) and splits them into onset and rime
    (either of which may be empty). It then
    swaps the onsets of those two words.

       Examples:
         'bunny rabbit' => 'runny babbit'
         'grilled cheese' => 'chilled greese'
         'apple tart' => 'tapple art'
         'brrr cold' => 'c brrrold'
         'cat bag dog' => 'bat cag dog'
         'apple' => 'apple'

    Precond: phrase is a nonempty string with
    only lowercase letters and spaces. It
    cannot start or end with a space."""
    pos1 = phrase.find(' ')

    w1 = phrase[:pos1]
    print('w1 is '+repr(w1))          # WATCH
    w2 = phrase[pos1+1:]
    print('w2 is '+repr(w2))          # WATCH

    if ' ' in w2:
        print('More than two')        # TRACE
        pos2 = w2.find(' ')
        w2 = w2[:pos2]
        print('w2 is '+repr(w2))      # WATCH
        w3 = w2[pos2+1:]
        print('w3 is '+repr(w3))      # WATCH
    else:
        w3 = ''

    # Process the first two words
    s1 = split_onset_rime(w1)
    print('w1 split as '+repr(s1))    # WATCH
    s2 = split_onset_rime(w2)
    print('w2 split as '+repr(s2))    # WATCH

    # Swap onsets
    new1 = s2[0] + s1[1]
    print('new1 is '+repr(new1))      # WATCH
    new2 = s1[0] + s2[1]
    print('new2 is '+repr(new2))      # WATCH

    if w3 != '':
        print('Swapping first two')   # TRACE
        return new1+' '+new2+' '+w3

    return new1 + ' ' + new2


def split_onset_rime(w):
    """Returns [onset, rime] from w

    The rime is the part after the onset. It
    is possible for either to be empty

    Precond: w a nonempty string of only letters"""
    print('split_onset: '+repr(w))       # TRACE/WATCH
    j = first_vowel(w)
    if j == -1:
        print('No vowels')                # TRACE
        return [w, '']
    return [w[:j], w[j:]]


def first_vowel(w):
    """Returns position of the first vowel 'aeiou'.

    Precond: w a nonempty string of only letters"""
    print('first_vowel: '+repr(w))       # TRACE/WATCH
    minpos = len(w)

    pos = w.find('a')
    if pos != -1 and pos < minpos:
        print('a found at '+repr(pos))   # WATCH
        minpos = pos

    pos = w.find('e')
    if pos != -1 and pos < minpos:
        print('e found at '+repr(pos))   # WATCH
        minpos = pos

    pos = w.find('i')
    if pos != -1 and pos < minpos:
        print('i found at '+repr(pos))   # WATCH
        minpas = pos

    pos = w.find('o')
    if pos != -1 and pos < minpos:
        print('o found at '+repr(pos))   # WATCH
        minpos = pos

    pos = w.find('u')
    if pos != -1 and pos < minpos:
        print('u found at '+repr(pos))   # WATCH
        minpos = pos

    if minpos == len(w):
        print('minpos is -1')            # TRACE
        return -1

    print('minpos is '+repr(minpos))     # WATCH
    return minpos
```

First Bug:

```
>>> spoonerize('grilled cheese')
w1 is 'grilled'
w2 is 'cheese'
split_onset: 'grilled'
first_vowel: 'grilled'
e found at 5
i found at 2
minpos is 5
w1 split as ['grill', 'ed']
split_onset: 'cheese'
first_vowel: 'cheese'
e found at 2
minpos is 2
w2 split as ['ch', 'eese']
new1 is 'ched'
new2 is 'grilleese'
'ched grilleese' # Expected: 'chilled greese'
```

Second Bug:

```
>>> spoonerize('apple')
w1 is 'appl'
w2 is 'apple'
split_onset: 'appl'
first_vowel: 'appl'
a found at 0
minpos is 0
w1 split as ['', 'appl']
split_onset: 'apple'
first_vowel: 'apple'
a found at 0
minpos is 0
w2 split as ['', 'apple']
new1 is 'appl'
new2 is 'apple'
'appl apple' # Expected: 'apple'
```

Third Bug:

```
>>> spoonerize('cat bag dog')
w1 is 'cat'
w2 is 'bag dog'
More than two
w2 is 'bag'
w3 is ''
split_onset: 'cat'
first_vowel: 'cat'
a found at 1
minpos is 1
w1 split as ['c', 'at']
split_onset: 'bag'
first_vowel: 'bag'
a found at 1
minpos is 1
w2 split as ['b', 'ag']
new1 is 'bat'
new2 is 'cag'
'bat cag' # Expected: 'bat cag dog'
```

(c) [6 points] **Do not implement the function specified below.** Instead, use assert statements to enforce the precondition. You may only use assert statements or assignment statements (e.g. to create a variable to make an assert statement easier to write), but no other control structures. Your assert statements do not need custom error messages.

```
def add_minute(s):
    """Returns the value (as a string) of adding one minute to 'HH:MM'

    Precondition: s is a string in the form 'HH:MM' where H and M are digits"""
```

5. [19 points] **Call Frames.**

Consider the following function definitions.

```
1  def equalize(a):
2      """Modifies a so the ends meet
3      Pre: nonempty list of ints"""
4      if a[0] > a[-1]:
5          a[-1] = a[0]
6      elif a[-1] > a[0]:
7          a[0] = a[-1]
8
```

```
9  def bumble(b):
10     """Returns a modified slice of b
11     Pre: nonempty list of ints"""
12     c = b[b[0]:]
13     equalize(c)
14     return c
15
```

Assume `a = [1, 2, 4, 3]` is a global variable referencing a list in the heap, as shown on the next page. Use the next two pages to diagram the evolution of the function call

```
a = bumble(a)
```

Diagram the state of the *entire call stack* for the function `bumble` 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 **nine** diagrams, not including the initial diagram that we have provided.

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. To help conserve time, you are allowed (and **encouraged**) to write "unchanged" if no changes were made to either a call frame, global space, or the heap.

Last Name: _____ First: _____ Netid: _____

Call Stack	Global Space	The Heap
	a **id1**	**id1** list 0 1 1 2 2 4 3 3

①

②

③

④

Last Name: _____ First: _____ Netid: _____

Call Stack	Global Space	The Heap
⑤		
⑥		
⑦		
⑧		
⑨		

Last Name: _____ First: _____ Netid: _____

6. [20 points total] **Objects and Functions**.

   In Assignment 3, you worked with the classes `CMYK` and `HSV`. Objects of the class `CMYK` have attributes `cyan`, `magenta`, `yellow`, and `black`, which are all floats in the range 0 to 100. Objects of the class `HSV` have attributes `hue`, `saturation`, and `value`. All of these attributes are floats. The attributes `saturation` and `value` must be in the range 0 to 1, while `hue` is in the range 0 to 360 (excluding 360).

   One the main things that we like to do with color objects is to combine two different colors. The proper way to combine colors depends on the color model. This is the motivation for the two problems below.

   (a) [8 points] To blend together two `CMYK` objects, we take each attribute color attribute A and combine the values for that attribute with the formula

   $$A = 1 - (1 - A_1)(1 - A_2)$$

   So to the get the `cyan` of the blending color, we would plug in the `cyan` of the first color into A_1 and the `cyan` of the second color into A_2. We would do the same for `magenta`, `yellow`, and `black`. However, as in the assignment, **this formula assumes that the color values are in the range 0 to 1**.

   With this in mind, implement the function below. You do not need to assert preconditions..

```
def blend_cmyk(cmyk1,cmyk2):
    """Returns a CMYK object that is a blend of the two

    Preconditions: cmyk1, cmyk2 are CMYK objects"""
```

(b) [12 points] We combine `HSV` objects by *interpolating* them. An interpolation is a weighted average using a value t in the range 0 to 1. For each attribute A we compute

$$A = A_1 + \Delta_A * t, \quad \text{where} \quad \Delta_A = (A_2 - A_1)$$

This formula does not require we convert any of the `HSV` attributes before using it. However, $\Delta_{\texttt{hue}}$ must be the shortest angle between the two hues, meaning **it must be a value between -180 degrees and 180 degrees**. If it falls outside of that range, you should adjust it to an equivalent angle before plugging it into the formula on the left.

With this in mind, implement the function below. You do not need to assert preconditions..

```
def interpolate_hsv(hsv1, hsv2, t):
    """MODIFIES hsv1 to be the t-value interpolation of hsv1 and hsv2

    Preconditions:  hsv1 and hsv2 are HSV objects. t is a float 0..1."""
```