The final exam is administered as a self-scheduled exam during final-exam week in the Hall of Presidents in James C. Colgate Student Union (where Donovan's Pub is). Instructions and times that exams may be taken are posted at: https://colgateuniversitycomputerscience.github.io/cosc101/exams/final/self-schedule.pdf. Note that calculators, notes, and computers are not permitted for the exam. You should allow two and a half hours to take the exam. The exam will be similar in format to the previous exams but slightly longer.

Final exam topics
The final exam is cumulative and encompasses all topics covered by earlier exams.

- New topics since the last exam
  - Dictionaries
  - Program design
  - Recursion
  - Classes and objects

- Topics from Exam 2:
  - Functions: defining and calling; parameter passing; return values; variable scope
  - Strings
  - Lists
  - Nested for loops
  - while loops
  - File Input/Output
  - Exceptions

- Topics from Exam 1:
  - Types (int, str, float, bool)
  - Variables and assignment
  - Operators (+, -, *, /, //, %, **, ==, !=, <, >, <=, >=)
  - Input and output (input, print)
  - Conditional statements (if/elif/else)
  - Repetition (non-nested for loops)
  - Accumulator pattern

1) Dictionaries, lists, and aliasing
What is the output of the following programs?

```python
a) def lookup(lst, dct):
    r = ''
    for i in lst:
        r += dct[i] + '.
    return r[:-1]

c = 'May'
d = {1: 'a', 2: 'b', 8: c[-1], 4: c[1]}
print(lookup([2, 4], d))
print(lookup([1, 8], d))
print(lookup([1, 2, 3, 4], d))
```
b) def insert(lst1, lst2, dct):
    for i in range(len(lst1)):
        if lst1[i] not in dct:
            dct[lst1[i]] = [lst2[i]]
        else:
            dct[lst1[i]] += [lst2[i]]
    return dct

d = {}
d2 = insert(['a', 'b'], [1, 2], d)
print(d)
print(d2)
d3 = insert(['b', 'c', 'b'], [3, 4, 5, 6], d2)
print(d3)
d4 = insert(['d', 'e'], [7], d3)
print(d4)

c) def remove(lst, dct):
    i = 0
    while i < len(lst):
        if lst[i] in dct:
            dct[lst[i]].remove(i)
        del lst[i]
        i += 1

lst1 = ['a', 'b', 'c', 'd']
dct1 = {'a': [0, 1, 2], 'b': [0, 1, 2], 'c': [0, 1, 2]}
remove(lst1, dct1)
print(lst1)
print(dct1)

d) def flip(dct1):
    dct2 = {}
    for i in dct1:
        dct2[dct1[i]] = i
    return dct2

dctA = {'a': 'b', 'c': 'd', 'e': 'b'}
dctB = flip(dctA)
print(dctA)
print(dctB)
dctC = flip(dctB)
print(dctC)
2) Classes and objects

Assume you are given the following code:

class Polygon:
    def __init__(self, num):
        self.sides = num

    def get_angle(self):
        return 180 - 360 / self.sides

    def get_name(self):
        if self.sides == 3:
            return "triangle"
        elif self.sides == 4:
            return "square"
        elif self.sides in prefixes:
            return prefixes[self.sides] + "gon"
        else:
            return str(self.sides) + "-gon"

    def has_fewer_sides(self, p):
        return self.sides < p.sides

    def __str__(self):
        return "A " + self.get_name() + " with " + str(self.get_angle()) + " degree angles"

def make(lst1):
    lst2 = []
    for i in lst1:
        lst2.append(Polygon(i))
    return lst2

a) What is the output of the following code?
a = Polygon(5)
b = make([3, 6, 9])
for c in b:
    print(c)
    print(c.has_fewer_sides(a))

b) Write a method called has_equal_angles for the Polygon class that takes another Polygon object as a parameter and returns True if the other polygon has angles of the same size and False otherwise.
c) Write a method called increase_sides for the Polygon class that takes an integer as parameter and increases (or decreases in the case of a negative number) the number of sides of the polygon by that amount. The resulting number of sides should never be less than 3 or greater than 100.

d) Outside of the Polygon class, write a function called most_sides that takes a list of Polygon objects and returns the polygon that has the largest number of sides. If there is a tie, the function should return the first polygon in the list that has the largest number of sides. If the list is empty, the function should return None.

3) Classes, lists, and modules
Write a class called Deck that represents a deck of cards.

- The state associated with the deck is a draw pile and a discard pile.
- The constructor should initialize the draw pile to contain all cards in a standard 52 card deck. Use a string to represent a card: e.g., "AH" is ace of hearts, "2H" is two of hearts, ... "10H" is ten of hearts, "JH" is Jack of hearts, ... "AD" is ace of diamonds, "AS" is ace of spades, "AC" is ace of clubs, ...
- A method called shuffle should rearrange the draw pile in a random order. Hint: use the choice function from the random module to randomly choose an item from a list.
- A method called draw that should remove the first card from the draw pile, add it to the discard pile, and return the drawn card.
- A method called was_drawn that takes a card (e.g., "AH") as a parameter and returns True if the card is in the discard pile, or False otherwise.
- A method called reset that moves all cards from the discard pile to the draw pile.
4) Recursion

What is the output of the following programs?

a) def recA(lst, n):
   if len(lst) == 1:
       return lst[0] == n
   else:
       m = len(lst)//2
       a = recA(lst[:m], n)
       b = recA(lst[m:], n)
       return a or b

   print(recA([1, 2, 3, 4], 2))
   print(recA([1, 2, 3, 4], 5))
   print(recA([1, 2, 3, 2, 1], 1))

b) def recB(n):
   if n == 1:
       return n
   else:
       return n + recB(n//2)

   print(recB(4))
   print(recB(7))

c) def recC(s):
   if len(s) == 0:
       return "" # Empty string
   elif len(s) == 1:
       return s
   else:
       return s[-1] + recC(s[1:-1]) + s[0]

   print(recC("abc"))
   print(recC("defg"))