COSC 101, Final Exam
December 2021

Name: ____

Please write your name above. Do not start the exam until instructed to do so.

You have 120 minutes to complete this exam. There are 8 questions and a total of 70 points available for this exam. Don't spend too much time on any one question.

Since indentation is important in Python, please be sure that your use of indentation is obvious for any code you write. If you want partial credit, show as much of your work and thought process as possible.

If you run out of space for answering a question, you can continue your answer on one of the blank pages at the end of the exam. If you do so, be sure to indicate this in two places: (1) below the question, indicate which blank page contains your answer, and (2) on the blank page, indicate which question you are answering.

The last page of the exam contains documentation for string, list, and dictionary methods.

Question	Points	Score
1	10	
2	5	
3	10	
4	5	
5	4	
6	15	
7	9	
8	12	
Total:	70	

1. (10 points) Assume that the following statements have already been executed:

```
dozen = 12
nums = [1, "two", 3.0]
title = "Still I Rise"
poems = {"Angelou": [title, "Caged Bird"], "Frost": "The Road Not Taken"}
```

Evaluate each of the following expressions and write the resulting value, or identify (and explain in plain english) the error in the code that would prevent it from running.

(a) `nums[0] + nums[1]`

(b) `nums[:2] + [nums[-1]]`

(c) `title[6:] * 2`

(d) `dozen - 4 / 2 + 3`

(e) `poems["Frost"][1]`

(f) `poems["The Road Not Taken"]`

(g) `poems["Angelou"][1]`

(h) `"Caged Bird" in poems or title in poems`

(i) `"rise" in title.lower() and (nums[2] == "two" or dozen == 12)`

(j) `str(nums[0]) + nums[1] + str(int(nums[2]))`

2. (5 points) What would the following program print?

```
n=6

while n > -5:
  if n == 0 or n == 6:
    print("first")
    n = 3

  elif n in [2, 3, -2, -3, 6]:
    print("second")
    if n == 2:
      print("third")
      n = -1
    else:
      print("fourth")
    n -= 1

  else:
      print("fifth")
      n -= 1
```

3. (10 points) Implement a function that takes a list with many zero values and returns a *sparse representation* of the list. The *sparse representation* of a list is a dictionary where the keys are the **indices** of the **non-zero** elements in the original list and the values are those non-zero elements.

   The dictionary must also include a **special key "d"** with the **length of the original list** as its value.

   This sparse representation uses less memory than the original list but provides enough information to re-create the original list from the dictionary if needed.

   For example, the list `[0,0,0,1,0,0,2,0,0,3]` would be converted to sparse representation `{3:1, 6:2, 9:3, "d":10}`

   The following code shows the doctest of the required function.

```
def dense2sparse(lst):
    """Create a sparse representation of the list
    >>> dense2sparse([-1, 0, 3, 0])
    {0: -1, 2: 3, 'd': 4}
    >>> dense2sparse([])
    {'d': 0}
    >>> dense2sparse([3.0, 0, 0, -0.5, 0])
    {0: 3.0, 3: -0.5, 'd': 5}
    >>> dense2sparse([0, 0, 0])
    {'d': 3}
    """
```

(This page left blank for your code for problem 3)

4. (5 points) What is the output of the following program?

```
def function(data):
    keys = []
    for k in data:
        keys.append(k)
    for k in keys:
        v = data[k]
        data[v] = k
    return keys

a = {"one": 1, "two": 2, "three": 3}
b = function(a)
print(a)
print(b)
```

5. (4 points) Assume that you have a dictionary with lists of information about trees, including their scientific name and maximum height (in feet):

```
trees = {'Norway Spruce': ['Picea abies', 124],
         'River Birch': ['Betula nigra', 120],
         'Northern Red Oak': ['Querces rubra', 165]}
```

(a) Circle **each** of the following lines of code that will **cause an error** if you use it to extract specific information about the trees

- `trees[0][1]`
- `'Spruce' in trees`
- `trees['Northern Red Oak'].upper()`
- `trees['Norway Spruce'][1] < trees['Northern Red Oak'][1]`

(b) Which of the following functions will return a list of tree heights when called as `helper(trees)`? Circle your answer.

```
def helper(x):
    h = []
    while len(x) > 0:
        h.append(x.pop()[1])
    return h
```

```
def helper(x):
    h = {}
    for t in x:
        h[x[t]] = x[t][-1]
    return h
```

```
def helper(x):
    h = []
    for t in x:
        h.append(x[t][-1])
    return h
```

6. (15 points) Imagine that you have downloaded your recent text messages into a file named `texts.txt`. This file has one text message per line prefaced by the date the text message was sent (in MM/DD/YY format). For example, the following might be the contents of `texts.txt`:

```
11/15/21: What's up???
11/15/21: That 101 exam had way more questions that I expected
11/28/21: Cool, l8tr
12/01/21: I like COSC
12/01/21: Definitely a great class, real fun too
12/01/21: 10/10 would take again
```

   You are curious how many text messages you sent with each of the following features:

   - At least one question mark
   - Fewer than 5 words (not including the date)
   - Were sent in December

   Write a function `texts_to_csv()` that reads your `texts.txt` file, counts the number of texts with each of the above features, and writes a new CSV file named `results.csv` with the results. For the example `texts.txt` file above, `results.csv` should contain the following:

```
Questions, Short Texts, December
1, 4, 3
```

   If the `texts.txt` file cannot be found, the program should print the following error message but not crash: `Sorry, no texts found!`

   Hint: Remember the `.split()` string method!

(This page left blank for your code for problem 5)

7. (9 points) For this problem, select one line of code from each of the pairs of lines of code below to solve the following problem. **Do not reorder the lines.**:

   Write a function called `sample` that takes two parameters (a list called `lst` and an integer called `n`) and then returns a new list of `n` random elements from `lst`. Each element can be selected only once (imagine pulling `n` items out of a bag and putting them into another bag). For example calling:

   ```
   sample([1, 2, 3, 4, 5], 3)
   ```

   might return `[3, 1, 5]` and calling

   ```
   sample(['The', 'ships', 'hung', 'in', 'the', 'sky'], 4)
   ```

   might return `['ships', 'sky', 'in', 'The']`

   Some reminders for the `random` module:

   - `random.random()` returns a floating point number between 0.0 and 1.0 (but not including 1.0 itself)
   - `random.randrange(a, b)` returns a random integer between a and b (but not including b itself)
   - `random.randint(a, b)` returns a random integer between a and b (including b itself)

```
A1  import randint
A2  import random

B1  def sample(lst):
B2  def sample(lst, n):

C1      lst2 = []
C2      lst2 = lst.copy()

D1      samp = lst.copy()
D2      samp = []

E1      for x in range(n):
E2      for x in range(n-1):

F1          randIdx = random.randint(0, len(lst2)-1)
F2          randIdx = randint(0, len(lst2)-1)

G1          samp.append(lst2[randIdx])
G2          samp = samp.append(lst2[randIdx])

H1          lst2.pop(lst2[randIdx])
H2          lst2.remove(lst2[randIdx])

I1      return samp
I2      return lst
```

Select only 9 lines of code from above, and only one line from each pair. **Do not reorder the lines.** You may fill in line identifiers (e.g., A1, B2, C1, etc.) below, or write out the code.

8. (12 points) For this problem, select one line of code from each of the pairs of lines of code below to solve the following problem. **Do not reorder the lines.**:

   Write a function called `count_occurrences` that takes a string to find and a string to search and counts the number of non-overlapping occurrences of the first string within the second. For example:

```
>>> count_occurrences("an", "hand me the banana")
3
count_occurrences("an", "sand in my hand")
2
>>> count_occurrences("ana", "banana")
1
```

```
A1  def count_occurrences(find, search):
A2  def count_occurrences():

B1      count = 0
B2      count = ""

C1      index = len(search)
C2      index = 0

D1      while index < len(search):
D2      while index < len(find):

E1          start = 0
E2          start = index

F1          end = len(search)
F2          end = index + len(find)

G1          if find in search[start:end]:
G2          if find == search[start:end]:

H1              count += index
H2              count += 1

I1              index += len(find)
I2              index += 1

J1          else:
J2          elif find not in search:

K1              index = count
K2              index += 1

L1      return count
L2      return index
```

Select only 12 lines of code from above, and only one line from each pair. **Do not reorder the lines.** You may fill in line identifiers (e.g., A1, B2, C1, etc.) below, or write out the code.

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)

String methods

- `upper()` — Returns a string in all uppercase
- `lower()` — Returns a string in all lowercase
- `capitalize()` — Returns a string with first character capitalized, the rest lower
- `strip()` — Returns a string with the leading and trailing whitespace removed
- `lstrip()` — Returns a string with the leading whitespace removed
- `rstrip()` — Returns a string with the trailing whitespace removed
- `count(item)` — Returns the number of occurrences of item
- `replace(old, new)` — Replaces all occurrences of old substring with new
- `center(width)` — Returns a string centered in a field of width spaces
- `ljust(width)` — Returns a string left justified in a field of width spaces
- `rjust(width)` — Returns a string right justified in a field of width spaces
- `find(item)` — Returns the leftmost index where the substring item is found, or -1 if not found
- `rfind(item)` — Returns the rightmost index where the substring item is found, or -1 if not found
- `index(item)` — Like find except causes a runtime error if item is not found
- `rindex(item)` — Like rfind except causes a runtime error if item is not found
- `split(separator)` — Return a list of the words in the string, using separator as the delimiter string
- `join(lst)` — Return a string which is the concatenation of the strings in lst
- `isalpha()` — Return True if all characters in the string are alphabetic and there is at least one character
- `isdigit()` — Return True if all characters in the string are decimal characters and there is at least one character
- `islower()` — Return True if all cased characters in the string are lowercase and there is at least one cased character
- `isspace()` — Return True if there are only whitespace characters in the string and there is at least one character
- `isupper()` — Return True if all cased characters in the string are uppercase and there is at least one cased character

List methods

- `append(item)` — Adds a new item to the end of a list
- `insert(position, item)` — Inserts a new item at the position given
- `extend(lst)` — Extend the list by appending all the items from lst
- `pop()` — Removes and returns the last item
- `pop(position)` — Removes and returns the item at position
- `sort()` — Modifies a list to be sorted
- `reverse()` — Modifies a list to be in reverse order
- `index(item)` — Returns the position of first occurrence of item
- `count(item)` — Returns the number of occurrences of item
- `remove(item)` — Removes the first occurrence of item
- `copy()` — Return a clone of the list
- `clear()` — Remove all items from the list

Dictionary methods

- `values()` — Gets the values of the dictionary for iteration
- `keys()` — Gets the keys of the dictionary for iteration
- `items()` — Gets (key, value) tuples of the dictionary for iteration