CSE 163 **Name:** _____
Winter 2020
Exam 1
02/14/2020 **Student Number:** _____
Time Limit: 50 Minutes

Do not open the exam before the exam begins and close the booklet when time is called. Starting early or working after time is called will lead to a -10 deduction. You may write your name and student number on the front of the exam before we start.

This exam contains 13 pages (including this cover page) and 5 questions. Some questions have sub-parts.

You are allowed to have one sheet of paper (both sides) with you as your cheatsheet. All other materials besides writing utensils should be put away before the exam starts. This includes all electronic devices like phones, calculators, and smartwatches.

This exam is not, in general, graded on style and you do not need to include comments or imports for your code. Specific questions may specify restrictions about the style of your code that you must follow to receive full credit.

You may not abbreviate any code, such as "ditto" marks or "..." marks. You may write code to the side and indicate where it should be inserted. These markings must be unambiguous and any ambiguity when grading may result in a deduction if your code is not readable. All code and answers should remain within the provided boxes if possible.

You are allowed to ask for scratch paper after the exam starts to use as additional space when writing answers, but you must indicate on the original page for the problem that part of the answer is on scratch paper. Scratch paper must be stapled to the **END** of your exam after you finish the test. Failure to do so may result in your work on scratch paper not being accepted while grading.

Initials: _____

Initial above to indicate you have read and agreed to the rules above. Failure to initial may result in your exam not being accepted for credit.

Question	Points	Score
1	10	
2	25	
3	25	
4	20	
5	10	
Total:	90	

---

1. (10 points) Write a function called `print_double_alternating` that takes the name of a file as a parameter and that prints out the file line-by-line, but only printing every other word on each line. Which word the line starts with should alternate on each line. For example, the first line should print the words in the first, third, fifth, etc. positions. The second line should print the words in the second, fourth, sixth, etc. positions.

   For example, if we had a file called `poem.txt` and called `print_double_alternating('poem.txt')`, then we would see the output on the right for the input file on the left.

   `poem.txt`

   According to all known laws
   of aviation,

   there is no way a bee
   should be able to fly.

   `print_double_alternating('poem.txt')`

   According all laws
   aviation,

   is way bee
   should able fly.

   We will use the same definition of "word" that we have used for every programming problem (a sequence of characters separated by whitespace). You should not make an assumption about the number of lines or the number of words on each line. Notice the example can handle a line that is blank (which should produce a blank line of output since there are no words).

   Hints:

   - Be careful with newlines. You may assume every line in the original file ends with a single-newline and does not begin with any white space characters. The output should not contain any blank lines unless it's a corresponding blank line from the input.
   - You may want to use the optional parameter value `end` for `print` to avoid printing values on separate lines. Example: `print('hello', end='')` prints "hello" without a newline afterwards.

(Write your solution in the box on the next page)

Solution:

```
def print_double_alternating(file_name):
  with open(file_name) as f:
    lines = f.readlines()
    for i in range(len(lines)):
      words = lines[i].split()
      for j in range(len(words)):
        if j % 2 == i % 2:
          print(words[j], end='')
      print()

def print_double_alternating(file_name):
  with open(file_name) as f:
    lines = f.readlines()
    print_evens = True
    for i in range(len(lines)):
      words = lines[i].split()
      for j in range(len(words)):
        if print_evens:
          if j % 2 == 0:
            print(words[j], end='')
        else:
          if j % 2 == 1:
            print(words[j], end='')

      print_evens = not print_evens
      print()
```

2. (25 points total) For the following problems, we will be working with some election data shown below. The data isn't perfectly formatted. There might be multiple rows for a particular candidate in a particular state. There are two main parts to this problem, 2.a represents the data as a list of dictionaries and 2.b represents it as a pandas `DataFrame`.

candidate	party	state	num_votes
Bernoulli Sanders	D	WA	200
Donatella Trump	R	WA	48
Donatella Trump	R	FL	300
Bernoulli Sanders	D	CA	400
Bernoulli Sanders	D	WA	50
Bernoulli Sanders	D	FL	100

(a) (10 points) Write a function called `count_state_votes` that takes two parameters, a list of dictionaries representing the data above and a name of a party, and returns the number of votes that party received in each state.

For example, if the data described above is stored in a variable called `data`, then the call `count_state_votes(data, 'D')` would return:

`{'WA': 250, 'CA': 400, 'FL': 100}`.

You may assume there are no missing values, but you shouldn't assume anything else about the data. If there are no rows in the dataset for the given party, your function should return `None`. For full credit, your solution should run in $\mathcal{O}(n)$ time where n is the number of rows in the dataset.

Solution:

```
def count_state_counts(data, party):
  counts = {}
  for row in data:
    if row['party'] == party:
      state = row['state']
      if state in counts:
        counts[state] += row['num_votes']
      else:
        counts[state] = row['num_votes']

  if len(counts) == 0:
    return None
  else:
    return counts
```

(b) For the following parts, we will assume we have parsed the above dataset in a `DataFrame` named `df`.

i. (5 points) Consider the following block of code.

```
df.loc[(df['state'] == 'WA') | (df['party'] == 'R'), 'num_votes']
```

First, what is the type of the value this expression produces (**select one**)

- ◯ `DataFrame`
- √ **`Series`**
- ◯ `dict` (dictionary)
- ◯ `int`
- ◯ `None`
- ◯ This code causes an error

Second, write the value this expression evaluates to. If you write out a `DataFrame` or a `Series`, you do NOT need to write out the index but if you write a `DataFrame`, you must indicate the column names. If you answered "error" to the last question, explain why in at most two sentences.

Solution:

```
200
48
300
50
```

ii. (10 points) Write a function called `count_state_votes` behaves exactly the same as 2.a except it takes a pandas `DataFrame` as a parameter instead of the list of dictionaries. You should convert the return value to a `dict` to match 2.a.

Like in the homeworks, for full credit your solution must not use any loops or comprehensions.

(Write your solution in the box on the next page)

Solution:

```
def count_state_votes(data, party):
  data = data[data['party'] == party]
  if len(data) == 0:
    return None
  else:
    return dict(data.groupby('state')['num_votes'].sum())
```

3. (25 points total) For this problem, we will use a global health dataset from the World Bank. For each problem, we will assume the data has been parsed as a `DataFrame` in a variable called `df`.

year	country	continent	fertility_rate	life_expectancy	population
1967	USA	North America	2.6	71	200
2017	USA	North America	1.8	79	350
1967	China	Asia	6.3	54	500
2017	China	Asia	1.7	76	1300
1967	India	Asia	5.8	46	750
2017	India	Asia	2.2	69	1300

   (a) For this part, we will focus on data visualization using the `seaborn` library.

      i. (9 points) In the space below, write the code to draw a scatter plot that shows the relationship between fertility rate and life expectancy in the year 2017. The plot should color each point based on the country's continent and the size of the point should reflect the population of the country. Assume we already ran `import seaborn as sns` and `sns.set()`.

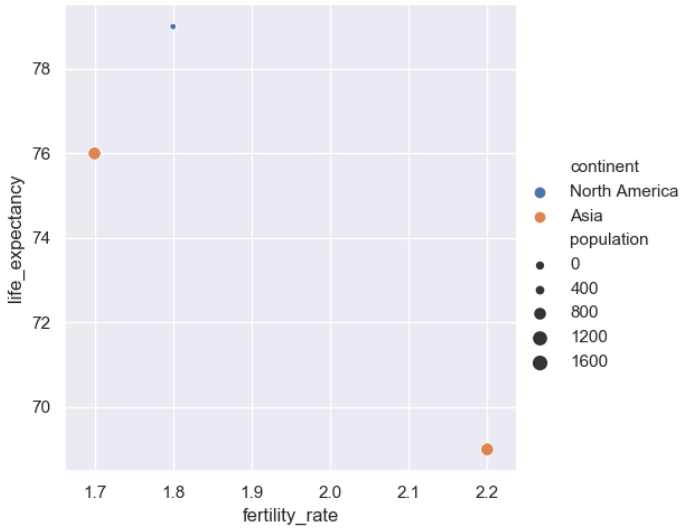
Solution:

```
sns.relplot(x='fertility_rate', y='life_expectancy',
            kind='scatter', hue='continent',
            size='population', data=df[df['year'] == 2017])
```

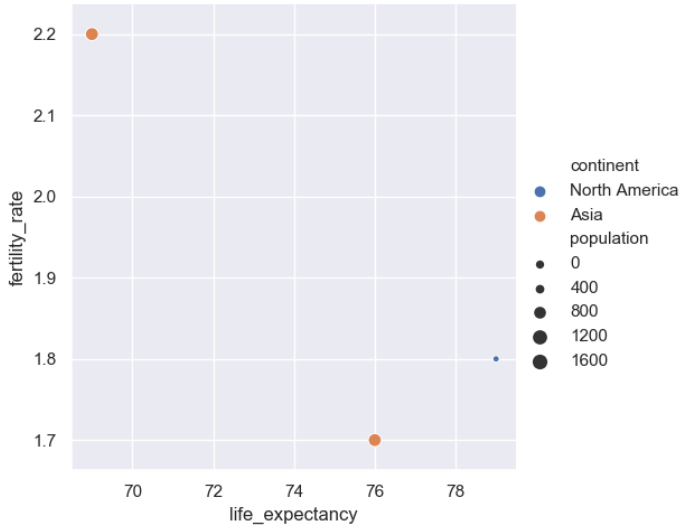
ii. (6 points) Sketch the scatter plot you created in the last problem. It does not need to be pixel perfect (we won't use a ruler to measure anything), but it should accurately convey the data. The graph you draw should have labeled axes and a legend explaining the colors. For drawing purposes, you should shade the points that represent "North America" and leave the others unshaded The sizes of the points do not need to be perfect, but your drawing should clearly demonstrate any population differences.

We encourage you to "clip" the axes so that they don't start at 0 to help differentiate the points. Make sure to label where they start/stop.

Solution: (yours would have "North America" shaded instead of colors)
Option 1



Option 2



(b) This problem involves training a machine learning model to predict the life expectancy in each country. We will make the following assumptions:

- We are working with a larger dataset that is representative of real-world data (more rows for more years and from all across the world).
- We have already run all imports and stored the data in a variable named `df`.

i. (4 points) We started writing the code to train and evaluate the model, but forgot the rest.

```
1 X = df[df.columns != 'life_expectancy']
2 y = df['life_expectancy']
3 X_train, X_test, y_train, y_test = \
4     train_test_split(X, y, test_size=0.2)
5 ------------------
6 ------------------
7 ------------------
8 ------------------
```

On the left, we show many possible lines of code that we could use. In each box on the right, write the corresponding letter for the line of code that should be placed there. Remember, to evaluate the model we want to have a good idea of how it will perform on future data.

(A) `model.fit(X, y)`

(B) `model.fit(X_train, y_train)`

(C) `model.fit(X_test, y_test)`

(D) `model = DecisionTreeClassifier()`

(E) `model = DecisionTreeRegressor()`

(F) `print(accuracy_score(y_pred, y))`

(G) `print(accuracy_score(y_pred, y_train))`

(H) `print(accuracy_score(y_pred, y_test))`

(I) `print(mean_square_error(y_pred, y))`

(J) `print(mean_square_error(y_pred, y_train))`

(K) `print(mean_square_error(y_pred, y_test))`

(L) `y_pred = model.predict(X)`

(M) `y_pred = model.predict(X_train)`

(N) `y_pred = model.predict(X_test)`

Line 5

Solution: E

Line 6

Solution: B

Line 7

Solution: N

Line 8

Solution: K

ii. (3 points) Assuming your choices above are correct, when running the code, it runs into the following error:
`ValueError: could not convert string to float: 'USA'`
In at most two sentences, explain why this error occurs and how to fix it. You do not need to write the code for the fix, just explain the idea.

Solution: The features `country` and `continent` are not numeric. We need to transform the data so that each string feature uses a one-hot encoding in order to train the model (using `pd.get_dummies`).

iii. (3 points) Your friend trains a decision tree model like yours and notices that the **training error** is very **low** while the **test error** is very **high**. They are wondering if they should increase or decrease the height of the tree to improve the model. In at most two sentences, explain what you would suggest to your friend and explain to them why you suggestion should work.

Hint: Think about extreme cases of increasing/decreasing the height of the tree.

Solution: They should decrease the height of the tree since the situation described sounds like overfitting. Decreasing the height of the tree makes the model less complex which makes it harder to overfit.

4. (a) (15 points) It's Valentine's Day and people are trying to make reservations for dinner on Open Table. We are going to help our friends at Open Table by writing a class called `OpenTable` manages reservations for restaurants on the website. An `OpenTable` object will store information about restaurants and the names of people who have reservations at each restaurant. Each restaurant will have a certain number of tables in it and a person can make a reservation at a restaurant which will always take one of its open tables. You will likely want multiple fields to track this information. People are uniquely determined by their name. **Write your response in the box on the next page.**

   The `OpenTable` class should have the following methods with the described arguments. All fields of the class should be private. You should not include any additional arguments for these methods:

Method	**Description**
`__init__(self)`	Creates an `OpenTable` and sets up any fields necessary.
`add_restaurant(self, res, cap)`	Adds a restaurant with the name `res` to this system with the given number of tables (`cap`). You may assume `res` is unique.
`get_open_tables(self, res)`	Returns the number of tables that are still open for the given restaurant `res`. If `res` is not a restaurant in this `OpenTable`, this function should return `None`.
`make_reservation(self, res, person)`	If possible, makes a reservation for the given `person` at the given `res`. There are three conditions where it is **not** possible to make a reservation: 1) If `res` is not in this `OpenTable`, 2) If there are no tables left for `res`, or 3) This `person` already has a reservation for `res`. If its possible to make the reservation, this information should be recorded. This method should return `True`/`False` if the reservation was made.

   (b) (5 points) **In the box below**, write a short program that constructs an `OpenTable` adds a restaurant named "Cafe Flora" with 3 tables, tries to make a reservation two times for Hunter, and then prints the number of tables remaining for Cafe Flora. You should not access the fields of the `OpenTable` object in this program, you should only call methods.

Solution:

```
open_table = OpenTable()
open_table.add_restaurant('Cafe Flora', 3)
open_table.make_reservation('Cafe Flora', 'Hunter')
open_table.make_reservation('Cafe Flora', 'Hunter')
print(open_table.get_open_tables('Cafe Flora'))
```

(Please implement the OpenTable class below)

Solution:

```
class OpenTable:
    def __init__(self):
        self._restaurants = {}
        self._reservations = {}

    def add_restaurant(self, res, cap):
        self._restarants[res] = res
        self._reservations[res] = set()

    def get_open_tables(self, res):
        if res in self._restaurants:
            return self._restaurants[res] \
                - len(self._reservations[res])
        else:
            return None

    def make_reservation(self, res, person):
        if res not in self._restaurants \
                or person in self._reservations[res] \
                or self.get_open_tables(res) == 0:
            return False
        else:
            self._reservations[res].add(person)
            return True
```

5. (10 points total) For the following problems, write the run-time of each function using the Big-O notation. For these problems, we will use n as the variable to describe the length of the input structure. Your answer should be the "smallest" Big-O runtime possible (i.e. you may not say $\mathcal{O}(n^{12})$ as an answer if $\mathcal{O}(n)$ is an answer that is closer to the actual run-time).

(a) (2½ points)

```
def fun1(nums):
  t = 0
  for i in range(3):
    for l in range(14):
      t += l

    for j in range(200):
      t += j
  return t
```

Solution: $\mathcal{O}(1)$

(b) (2½ points)

```
def fun2(nums):
  x = len(nums)
  t = 0
  for i in range(x * x / 2):
    t += i
  return t
```

Solution: $\mathcal{O}(n^2)$

(c) (2½ points)

```
def fun3(nums):
  return max(nums) * min(nums)
```

Solution: $\mathcal{O}(n)$

(d) (2½ points)

```
def fun4(nums):
  r = None
  for i in range(10):
    if i % 2 == 0:
      r = max(nums)
    else:
      r = None
  return r
```

Solution: $\mathcal{O}(n)$

- **Built-in Python functions**
  - `print(*strings)`
  - `range(end)`
  - `range(start, end[, step])`
  - `abs(v)`
  - `min(v1, v2) / max(v1, v2)`
  - `sum(v1, v2)`
  - `open(fname)`
  - Types: `int(v)`,`float(v)`,`str(v)`,`bool(v)`
- **String methods**
  - `upper()`, `lower()`
  - `find(s)`
  - `strip()`
  - `split()`
- **List methods**
  - Construct: `list()` or `[]`
  - `append(val)`
  - `extend(lst)`
  - `insert(idx, val)`
  - `remove(val)`
  - `pop(idx)`
  - `index(val)`
  - `reverse()`
  - `sort(key=None)`
- **Set methods**
  - Construct: `set()`
  - `add(val)`
  - `remove(val)`
- **Dictionary methods**
  - Construct: `dict()` or `{}`
  - `keys()`
  - `values()`
  - `items()`
- **File methods**
  - `readlines()`
  - `read()`
- **Pandas methods**
  - Parse: `pd.read_csv(file_name)`
  - `mean()`
  - `min() / max()`
  - `idxmin() / idxmax()`
  - `count()`
  - `unique()`
  - `groupby(col)`
  - `apply(fun)`
  - `isnull() / notnull()`
  - `dropna() / fillna(v)`
  - `sort_values(col)`
  - `sort_index()`
  - `nlargest(n, col)`
- **Pandas fields**
  - `index`
  - `loc[row, col]`
- **Seaborn methods**
  - `sns.catplot(x, y, data, kind[, hue])`
    kind: [“count”, “bar”, “violin”]
  - `sns.relplot(x, y, data, kind[, hue[, size]])`
    kind: [“scatter”, “line”]
  - `sns.regplot(x, y, data)`
- **sklearn methods**
  - `sklearn.metrics.accuracy_score(y_true, y_pred)`
  - `sklearn.metrics.mean_square_error(y_true, y_pred)`
  - `sklearn.model_selection.train_test_split(X, y, test_size)`
- **sklearn model classes**
  - `sklearn.tree.DecisionTreeClassifier()`
  - `sklearn.tree.DecisionTreeRegressor()`
- **sklearn model methods**
  - `fit(X, y)`
  - `predict(X)`
- **Special object methods**
  - `__init__`
  - `__repr__`
  - `__eq__`