## Data Management and Analysis, Sample Midterm Questions:

(these questions do not indicate the total length / difficulty level of the exam; instead, it shows the types of questions that you'll see… and the reference is not going to be the exact reference used in the exam)

1 Answer the questions about list comprehensions below:

a. Consider the following variable definitions:

```
vowels = 'aeiou'
s = "queueing"
```

Write one line of code to count the number of vowels in `s`. Both variables listed above must be used. You **must use a list comprehension** somewhere in your solution (you can use other built-in functions / methods in conjunction with your list comprehension to help):

______________________________________________________________

b. **Use a list comprehension** to find the largest label in the following Series, `s` (again, you can use any other built-in functions / methods in conjunction with your list comprehension to help):

```
from pandas import Series

s = Series(
    data=['foo', 'bar', 'baz', 'qux', 'quxx', 'corge'],
    index=['ant', 'bat', 'cat', 'good doge', 'eel', 'fly']
)
```

______________________________________________________________

2 Given the following `DataFrame`, write code to modify it based on the specifications below:

```
df = DataFrame(
    [['Ca', 'San Diego'],
     ['OR', 'Portland'],
     ['CA', 'Oakland'],
     ['ca', 'Pasadena'],
     ['wA', 'Olympia']],
    columns=['state', 'city']
)
```

a. Change `df` so that state abbreviations are normalized to uppercase.

______________________________________________________________

b. Modify `df` so that the order of columns is swapped (change to city first… followed by state)

______________________________________________________________

3 Write the output of code below in the space adjacent to the code (error or no output are possible; not all lines have to be filled):

```
numbers = [1, 2, 3]

def foo():
    numbers = [True, False]

print(numbers)
```

a.

```
def outer(x):
    print('stan')

    def inner():
        x()
        print('mable')
    return inner


def f():
    print('dipper')
```

b.

```
def add_o(v):
    # type(v) == str checks if v is a string
    s = f'.oO{v}Oo.' if type(v) == str else v
    return s

def fancify_it(plain):

    def fancy(*args):
        modified = [add_o(a) for a in args]
        plain(*modified)

    return fancy

@fancify_it
def wat(a, b):
    print(a)
    print(b)

wat('hi', 1)
```

c.

4 Imagine that a single pixel in a digital image can be represented by a tuple consisting of 3 integers, each from 0 - 255. The integers can represent red, green, and blue (based on position in the tuple). Imagine further that an image can be composed of a 2-dimensional array of pixels (with each pixel having length 3).

Using this idea of a digital image, create a class, Image, such that it can be initialized by passing in a grid of pixels (again, each pixel has integers for red, green and blue). The instance of this class can be indexed into to retrieve or write values. Furthermore, it can support a method called dim that returns the width and height as a tuple (width first, height next). Here's how the class works:

```
img = Image([
    [(255, 255, 255), (  0,   0,   0), (100, 200,   0)],
    [(  0, 100,   0), (  0,  77,  77), (123, 123, 123)],
])

print(img[1, 0])      # prints out (0, 0, 0)

img[2, 1] = (1, 1, 1) # (sets pixel to tuple)
print(img[2, 1])      # prints out (1, 1, 1)

print(img.dim())      # prints out (3, 2)
```

Create a class that would support the behavior above  (the bare minimum implementation, without error handling or concern regarding efficiency is adequate).

Hint: In the code, `img[2, 1]`, the what type is `2, 1`?

5

```
# Given the following data:
exam_data = {
    'name':      ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',
                  'Michael','Matthew', 'Laura', 'Kevin', 'Jonas'],
    'score':     [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
    'attempts':  [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'qualify':   ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no',
                  'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

a) Create a dataframe from the dictionary data above, and with index as the variable, `labels`.

b) Get the first 3 rows of the dataframe

c) Select the ‘name’ and ‘score’ columns from the dataframe

d) Select rows where number of attempts in the examination is greater than 2

e) Select rows where the score is missing i.e. is NaN

f) Select rows where number of attempts in the examination is < 3 and score > 15

g) Change the score in row ‘d’ to 11.5

h) Calculate the mean score amongst all the students

i) Add a new row with index ‘k’ to the dataframe with the following values for each column:
name: “Sam”, score: 15.5, attempts: 1, qualify: “yes”, label: “k”
Then, delete the new row that you just created!

j) Sort the DataFrame first by “name” in descending order

k) Modify the values in “qualify” column such that ‘yes’ becomes `True` and ‘no’ becomes `False`

6 Consider the data shown in the below table:

| street_number | street_name | num_bedrooms | owner_occupied |
|---|---|---|---|
| 104 | PUTNAM | 15 | Yes |
| 197 | LEXINGTON | 3 | No |
| NaN | LEXINGTON | n/a | No |
| 201 | BERKELEY | 2 | 12 |
| 203 | BERKELEY | 3 | Yes |
| 207 | BERKELEY | NaN | Yes |
| NaN | LEXINGTON | 2 | NaN |
| 213 | TREMONT | -5 | Yes |
| 217 | TREMONT | na | Yes |

- List 4 issues with the data above
- Describe your approach to "fixing" each issue
- Write out any assumptions you've made when coming up with these issues and their corresponding fixes
- When writing code, refer to the data above as df (think of this variable as a DataFrame containing the data above that already exists prior to writing you code)

7 Using the same date in 6, assume that the data is stored in a table called `property`

a. find the average number of rooms for each street in the data set; sort by average number of rooms in decreasing order of properties

b. only retrieve the street number and number of bedrooms, but restrict results to not owner occupied with more than two bedrooms

# Reference Material / Scrap Paper

**ASCII Chart**

| Char | Dec | Char | Dec | Char | Dec | Char | Dec |
|---|---|---|---|---|---|---|---|
| (nul) | 0 | (sp) | 32 | @ | 64 | ` | 96 |
| (soh) | 1 | ! | 33 | A | 65 | a | 97 |
| (stx) | 2 | " | 34 | B | 66 | b | 98 |
| (etx) | 3 | # | 35 | C | 67 | c | 99 |
| (eot) | 4 | $ | 36 | D | 68 | d | 100 |
| (enq) | 5 | % | 37 | E | 69 | e | 101 |
| (ack) | 6 | & | 38 | F | 70 | f | 102 |
| (bel) | 7 | ' | 39 | G | 71 | g | 103 |
| (bs) | 8 | ( | 40 | H | 72 | h | 104 |
| (ht) | 9 | ) | 41 | I | 73 | i | 105 |
| (nl) | 10 | * | 42 | J | 74 | j | 106 |
| (vt) | 11 | + | 43 | K | 75 | k | 107 |
| (np) | 12 | , | 44 | L | 76 | l | 108 |
| (cr) | 13 | - | 45 | M | 77 | m | 109 |
| (so) | 14 | . | 46 | N | 78 | n | 110 |
| (si) | 15 | / | 47 | O | 79 | o | 111 |
| (dle) | 16 | 0 | 48 | P | 80 | p | 112 |
| (dc1) | 17 | 1 | 49 | Q | 81 | q | 113 |
| (dc2) | 18 | 2 | 50 | R | 82 | r | 114 |
| (dc3) | 19 | 3 | 51 | S | 83 | s | 115 |
| (dc4) | 20 | 4 | 52 | T | 84 | t | 116 |
| (nak) | 21 | 5 | 53 | U | 85 | u | 117 |
| (syn) | 22 | 6 | 54 | V | 86 | v | 118 |
| (etb) | 23 | 7 | 55 | W | 87 | w | 119 |
| (can) | 24 | 8 | 56 | X | 88 | x | 120 |
| (em) | 25 | 9 | 57 | Y | 89 | y | 121 |
| (sub) | 26 | : | 58 | Z | 90 | z | 122 |
| (esc) | 27 | ; | 59 | [ | 91 | { | 123 |
| (fs) | 28 | < | 60 | \ | 92 | \| | 124 |
| (gs) | 29 | = | 61 | ] | 93 | } | 125 |
| (rs) | 30 | > | 62 | ^ | 94 | ~ | 126 |
| (us) | 31 | ? | 63 | _ | 95 | (del) | 127 |

| **Built-in** | **String Methods** | **Math Module** | **List Methods** | **File Object Methods** |
|---|---|---|---|---|
| abs<br>bool<br>bytes<br>chr<br>dict<br>enumerate<br>filter<br>float<br>format<br>input<br>int<br>len<br>list<br>map<br>max<br>min<br>open<br>ord<br>pow<br>print<br>range<br>round<br>set<br>sorted<br>str<br>sum<br>super<br>zip | capitalize<br>count<br>endswith<br>find<br>format<br>index<br>isalnum<br>isalpha<br>isdecimal<br>isdigit<br>islower<br>isnumeric<br>isprintable<br>isspace<br>istitle<br>isupper<br>join<br>lower<br>replace<br>split<br>startswith<br>strip<br>title<br>upper | acos<br>acosh<br>asin<br>asinh<br>atan<br>atan2<br>atanh<br>ceil<br>cos<br>cosh<br>degrees<br>floor<br>log<br>log10<br>log2<br>pi<br>pow<br>radians<br>sin<br>sinh<br>sqrt<br>tan<br>tanh | append<br>count<br>extend<br>index<br>insert<br>pop<br>remove<br>reverse<br>sort | read<br>readline<br>readlines<br>write<br><br>with<br>open(…)<br>as<br><br>for ln in f |
| | | | **Dictionary Methods** | **Random Module Functions** |
| | | | get<br>items<br>keys<br>pop<br>popitem<br>update<br>values | choice<br>randint<br>**sample**<br>shuffle |

Pandas DataFrame methods (we did not go over most of these, and you won't need to use the majority of these)

abs() Return a Series/DataFrame with absolute numeric value of each element.
add(other[, axis, level, fill_value]) Addition of dataframe and other, element-wise (binary operator add).
all([axis, bool_only, skipna, level]) Return whether all elements are True, potentially over an axis.
any([axis, bool_only, skipna, level]) Return whether any element is True over requested axis.
append(other[, ignore_index, …]) Append rows of other to the end of this frame, returning a new object.
apply(func[, axis, broadcast, raw, reduce, …]) Apply a function along an axis of the DataFrame.
applymap(func) Apply a function to a Dataframe elementwise.
astype(dtype[, copy, errors]) Cast a pandas object to a specified dtype dtype.
clip([lower, upper, axis, inplace]) Trim values at input threshold(s).
clip_lower(threshold[, axis, inplace]) Return copy of the input with values below a threshold truncated.
clip_upper(threshold[, axis, inplace]) Return copy of input with values above given value(s) truncated.
combine(other, func[, fill_value, overwrite]) Add two DataFrame objects and do not propagate NaN values, so if for a (column, time) one frame is missing a value, it will default to the other frame's value (which might be NaN as well)
combine_first(other) Combine two DataFrame objects and default to non-null values in frame calling the method.
copy([deep]) Make a copy of this object's indices and data.
count([axis, level, numeric_only]) Count non-NA cells for each column or row.
cummax([axis, skipna]) Return cumulative maximum over a DataFrame or Series axis.
cummin([axis, skipna]) Return cumulative minimum over a DataFrame or Series axis.
cumprod([axis, skipna]) Return cumulative product over a DataFrame or Series axis.
cumsum([axis, skipna]) Return cumulative sum over a DataFrame or Series axis.
describe([percentiles, include, exclude]) Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
drop([labels, axis, index, columns, level, …]) Drop specified labels from rows or columns.
drop_duplicates([subset, keep, inplace]) Return DataFrame with duplicate rows removed, optionally only considering certain columns
dropna([axis, how, thresh, subset, inplace]) Remove missing values.
duplicated([subset, keep]) Return boolean Series denoting duplicate rows, optionally only considering certain columns
eq(other[, axis, level]) Wrapper for flexible comparison methods eq
equals(other) Determines if two NDFrame objects contain the same elements.
ffill([axis, inplace, limit, downcast]) Synonym for DataFrame.fillna(method='ffill')
fillna([value, method, axis, inplace, …]) Fill NA/NaN values using the specified method
filter([items, like, regex, axis]) Subset rows or columns of dataframe according to labels in the specified index.
ge(other[, axis, level]) Wrapper for flexible comparison methods ge

get_dtype_counts()	Return counts of unique dtypes in this object.
get_ftype_counts()	(DEPRECATED) Return counts of unique ftypes in this object.
groupby([by, axis, level, as_index, sort, …])	Group series using mapper (dict or key function, apply given function to group, return result as series) or by a series of columns.
gt(other[, axis, level])	Wrapper for flexible comparison methods gt
head([n])	Return the first n rows.
interpolate([method, axis, limit, inplace, …])	Interpolate values according to different methods.
isin(values)	Return boolean DataFrame showing whether each element in the DataFrame is contained in values.
isna()	Detect missing values.
isnull()	Detect missing values.
items()	Iterator over (column name, Series) pairs.
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns with other DataFrame either on index or on a key column.
last_valid_index()	Return index for last non-NA/null value.
le(other[, axis, level])	Wrapper for flexible comparison methods le
lt(other[, axis, level])	Wrapper for flexible comparison methods lt
max([axis, skipna, level, numeric_only])	This method returns the maximum of the values in the object.
mean([axis, skipna, level, numeric_only])	Return the mean of the values for the requested axis
median([axis, skipna, level, numeric_only])	Return the median of the values for the requested axis
merge(right[, how, on, left_on, right_on, …])	Merge DataFrame objects by performing a database-style join operation by columns or indexes.
min([axis, skipna, level, numeric_only])	This method returns the minimum of the values in the object.
mod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator mod).
mode([axis, numeric_only])	Gets the mode(s) of each element along the axis selected.
mul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator mul).
multiply(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator mul).
ne(other[, axis, level])	Wrapper for flexible comparison methods ne
nlargest(n, columns[, keep])	Return the first n rows ordered by columns in descending order.
notna()	Detect existing (non-missing) values.
notnull()	Detect existing (non-missing) values.
nsmallest(n, columns[, keep])	Get the rows of a DataFrame sorted by the n smallest values of columns.
nunique([axis, dropna])Return Series with number of distinct observations over requested axis.
pop(item)	Return item and drop from frame.
pow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator pow).
prod([axis, skipna, level, numeric_only, …])	Return the product of the values for the requested axis
product([axis, skipna, level, numeric_only, …])	Return the product of the values for the requested axis
rank([axis, method, numeric_only, …])	Compute numerical data ranks (1 through n) along axis.
rdiv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator rtruediv).
rename([mapper, index, columns, axis, copy, …])	Alter axes labels.
rename_axis(mapper[, axis, copy, inplace])	Alter the name of the index or columns.
reorder_levels(order[, axis])	Rearrange index levels using input order.
replace([to_replace, value, inplace, limit, …])	Replace values given in to_replace with value.
reset_index([level, drop, inplace, …])	For DataFrame with multi-level index, return new DataFrame with labeling information in the columns under the index names, defaulting to 'level_0', 'level_1', etc.
set_axis(labels[, axis, inplace])	Assign desired index to given axis.
set_index(keys[, drop, append, inplace, …])	Set the DataFrame index (row labels) using one or more existing columns.
set_value(index, col, value[, takeable])	(DEPRECATED) Put single value at passed column and index
shift([periods, freq, axis])	Shift index by desired number of periods with an optional time freq
sort_index([axis, level, ascending, …])	Sort object by labels (along an axis)
sort_values(by[, axis, ascending, inplace, …])	Sort by the values along either axis
sortlevel([level, axis, ascending, inplace, …])	(DEPRECATED) Sort multilevel index by chosen axis and primary level.
stack([level, dropna])	Stack the prescribed level(s) from columns to index.
std([axis, skipna, level, ddof, numeric_only])	Return sample standard deviation over requested axis.
sub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator sub).
subtract(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator sub).
sum([axis, skipna, level, numeric_only, …])	Return the sum of the values for the requested axis
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately
tail([n])	Return the last n rows.
to_csv([path_or_buf, sep, na_rep, …])	Write DataFrame to a comma-separated values (csv) file
to_dict([orient, into])	Convert the DataFrame to a dictionary.
to_excel(excel_writer[, sheet_name, na_rep, …])	Write DataFrame to an excel sheet
to_html([buf, columns, col_space, header, …])	Render a DataFrame as an HTML table.
to_json([path_or_buf, orient, date_format, …])	Convert the object to a JSON string.
to_string([buf, columns, col_space, header, …])	Render a DataFrame to a console-friendly tabular output.
transform(func, *args, **kwargs)	Call function producing a like-indexed NDFrame and return a NDFrame with the transformed values
transpose(*args, **kwargs)	Transpose index and columns.
truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.
where(cond[, other, inplace, axis, level, …])	Return an object of same shape as self and whose corresponding entries are from self where cond is True and otherwise are from other.

# Solutions

1.
```
a. len([letter for letter in s if letter in vowels])
   sum([1 for letter in s if letter in vowels])
   sum([letter in vowels for letter in s]) # tricky solution!
```
The last one is a bit strange in that sum's behavior coerces `True` to `1` and `False` to `0` (not the expected solution, but added for completeness).
```
b.  max([len(label) for label in s.index])
```

2.
```
a. df['state'] = df['state'].str.upper()
   df['state'] = df['state'].apply(lambda state: state.upper())
b. df = df.reindex(columns=['city', 'state'])
   df = df[['city', 'state']]
```

3.
```
a. [1, 2, 3]
b. stan
   dipper
   mable
c. .oOhiOo.
```

4.
```
class Image:
    def __init__(self, pixels):
        self.pixels = pixels
    def __getitem__(self, t):
        x, y = t
        return self.pixels[y][x]
    def __setitem__(self, t, v):
        x, y = t
        self.pixels[y][x] = v
    def dim(self):
        return len(self.pixels[0]), len(self.pixels)
```

5.
```
a. df = pd.DataFrame(exam_data, index=labels)
b. df.head(3) # OR df.iloc[:3]
c. df[['name','score']]
d. df[df['attempts']>2]
e. df[df['score'].isnull()]
f. df[df['attempts'] < 3][df['score'] > 15]
   # OR df[(df['attempts'] < 3) & (df['score'] > 15)]  ...(did not
   # cover, but for completeness)
g. df.loc['d', 'score'] = 11.5
h. df['score'].mean()
i. df.loc['k'] = ['Sam', 15.5, 1, 'yes']
   df = df.drop('k')
j. df = df.sort_values(by=['name'], ascending=[False])
k. df['qualify'] = df['qualify'].map({'yes': True, 'no': False})
```

6. Varying solutions

7. a. SELECT street_name, AVG(num_bedrooms) FROM property GROUP BY street_name ORDER BY AVG(num_bedrooms) DESC;
   b. SELECT street_number, num_bedrooms FROM property WHERE num_bedrooms > 2 a owner_occupied <> 'Yes';