In [ ]:
```
"""
Add LabeledList and Table classes
"""


import numpy as np
import csv

def read_csv(fn):
    with open(fn, 'r') as f:
        csv_reader = csv.reader(f, delimiter=',')
        columns = next(csv_reader)
        data_lines = []
        for line in csv_reader:
            one_line = []
            for i in range(len(line)):
                if line[i].isnumeric():
                    try:
                        one_line.append(float(line[i]))
                    except:
                        raise ValueError(f"Value error: {line[i]}")
                else:
                    one_line.append(line[i])
            if one_line != []:
                data_lines.append(one_line)
        index = list(range(len(data_lines)))
    return Table(data_lines, index, columns)


class LabeledList:

    def __init__(self, data= None, index = None):
        self.values = data
        if index is None:
            self.index = list(range(0, len(data)))
        else:
            self.index = index
        self.tup_list = [(self.index[i], self.values[i]) for i in range(len(self.index))]
        self.max_iter = len(data)

    def __str__(self):
        tup_list = self.tup_list
        pair_list = [f'{tup[0]:>15}{tup[1]:>15}\n' for tup in tup_list]
        ret = ''.join(pair_list)
        return ret

    def __repr__(self):
        return str(self)

    def __getitem__(self, key_list):
        index = self.index
        values = self.values
        tup_list = self.tup_list
        if isinstance(key_list, LabeledList):
            key_list = key_list.values
        elif isinstance(key_list, list):
            if isinstance(key_list[0], bool):
                true_index = [i for i, x in enumerate(key_list) if x]
                key_list = [index[i] for i in true_index]
        elif isinstance(key_list, str):
            if self.index.count(key_list)==1:
                return values[index.index(key_list)]
            key_list = [key_list]
        elif isinstance(key_list, int):
            return values[key_list]
        else:
            raise KeyError

        pair_tuples = [tup for tup in tup_list if tup[0] in key_list]
        new_index , new_values = list(zip(*pair_tuples))[0], list(zip(*pair_tuples))[1]
        return LabeledList(new_values, new_index)

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        values = self.values
        if self.n < self.max_iter:
            result = values[self.n]
            self.n += 1
            return result

        else:
            raise StopIteration

    def __eq__(self, scalar):
        values = self.values
        labels = self.index
        equal = [False if value is None or value!=scalar else True for value in values]
        return LabeledList(equal, labels)



    def __ne__(self, scalar):
        values = self.values
        labels = self.index
        not_equal = [False if value is None or value==scalar else True for value in values]
        return LabeledList(not_equal, labels)


    def __gt__(self, scalar):
        values = self.values
        labels = self.index
        gt = [False if value is None or value<scalar else True for value in values]
        return LabeledList(gt, labels)


    def __lt__(self, scalar):
        values = self.values
        labels = self.index
        lt =[False if value is None  or value>scalar else True for value in values]
        return LabeledList(lt, labels)

    def map(self, f):
        res = list(map(f, self.values))
        return LabeledList(res, self.index)




class Table:

    def __init__(self, data, index= None, columns = None):
        self.values = data
        if index is None:
            self.index = list(range(0, len(data)))
        else:
            self.index = index
        if columns is None:
            self.columns = list(range(0, len(data[0])))
        else:
            self.columns = columns
        self.max_iter = len(data)


    def __str__(self):
        columns = self.columns
        index = self.index
        values = self.values
        join_list = lambda l: ''.join([f'{w:>15}' for w in l])+'\n'
        heading =  join_list(columns)
        row_list = [f"{index[i]}" + join_list(values[i]) for i in range(len(index))]
        combined = heading + ''.join(row_list)
        return combined

    def __repr__(self):
        return str(self)

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        values = self.values
        if self.n < self.max_iter:
            result = values[self.n]
            self.n += 1
            return result

        else:
            raise StopIteration
    def __getitem__(self, col_list):
        columns = self.columns
        index = self.index
        values = self.values
        get_a_column = lambda i: [value[i] for value in values]
        get_a_row = lambda i: values[i]

        def indices(lst, item):
            return [i for i, x in enumerate(lst) if x == item]

        def indices_of_duplicates(col_list):
            ret = []
            for col in col_list:
                ret.append(columns.index(col))
            return ret

        def get_col_index_and_list(col_list):

            if isinstance(col_list[0], bool):
                row_index = [i for i, x in enumerate(col_list) if x]
                col_index = list(range(0, len(columns)))
                select_row = True
                select_column = False
            else:
                if len(set(col_list)) == len(col_list):
                    col_index = [indices(columns, col) for col in col_list]
                    col_index = [item for sublist in col_index for item in sublist]

                else:
                    col_index = indices_of_duplicates(col_list)
                row_index = index
                select_row = False
                select_column = True

            if select_row:
                new_values = [get_a_row(i) for i in row_index]
            elif select_column:
                new_values = [get_a_column(i) for i in col_index]
                new_values = list(np.array(new_values).T)
            return col_index, row_index, new_values, select_row

        if isinstance(col_list, LabeledList):
            col_list = col_list.values
            col_index, row_index, new_values, select_row = get_col_index_and_list(col_list)
            new_col_list = [columns[i] for i in col_index]
            new_table = Table(new_values, row_index, new_col_list)
            return new_table

        elif isinstance(col_list, list):
            col_index,  row_index, new_values, select_row = get_col_index_and_list(col_list)
            new_col_list = [columns[i] for i in col_index]
            if select_row:
                # if filtering rows, change the index to those in the unfiltered df
                row_index = [self.index[i] for i in row_index]
                new_table = Table(new_values, row_index, new_col_list)
            else:
                # without filtering rows, index unchanged
                new_table = Table(new_values, row_index, new_col_list)
            return new_table


        elif isinstance(col_list, str):
            col_index = indices(columns, col_list)
            if len(col_index) == 1:
                ret_col = get_a_column(col_index[0])
                return LabeledList(ret_col, index)
            else:
                new_values = [get_a_column(i) for i in col_index]
                new_values = list(np.array(new_values).T)
                row_index = index
                new_col_list = [columns[i] for i in col_index]
                new_table = Table(new_values, row_index, new_col_list)
                return new_table
        else:
            raise KeyError()


    def head(self, n):
        new_values = self.values[:n]
        new_index = self.index[:n]
        return Table(new_values, new_index, self.columns)

    def tail(self, n):
        new_values = self.values[-n:]
        new_index = self.index[-n:]
        return Table(new_values, new_index, self.columns)

    def shape(self):
        return (len(self.index), len(self.columns))
```

Table of Contents

first lambda

In [ ]:
```
import csv

file_path = './occupations-truncated.csv'
with open(file_path, 'r') as f:
    csv_reader = csv.reader(f, delimiter=',')
    next(csv_reader)
    lines = [(line[1], int(line[4].replace('$', '').replace(',', ''))) for line in csv_reader if 'computer' in lin
    max_occupation = max(lines, key = lambda x: x[-1])
    min_occupation = min(lines, key = lambda x: x[-1])
print( min_occupation, max_occupation,)
```

```
('Data Entry Keyers', 35850) ('Computer and Information Systems Managers', 161730)
```

In [ ]:
```
import nelta as nt
import numpy as np
```

In [ ]:
```
file_path = './occupations-truncated.csv'
occupation = nt.read_csv(file_path)
occupation.shape()
```

Out[ ]:
```
(1034, 5)
```

In [ ]:
```
occupation.columns
```

Out[ ]:
```
['Occupation code',
 'Occupation title',
 'Level',
 'Employment',
 'Annual mean wage']
```

In [ ]:
```
occupation.head(4)
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
0        00-0000All Occupations          total    139099570.0        $56,310
1        11-0000Management Occupations          major      7947300.0       $126,480
2        11-1000 Top Executives          minor      2601070.0       $129,920
3        11-1011Chief Executives         detail       202360.0       $197,840
```

In [ ]:
```
last_four = occupation.tail(4)
```

In [ ]:
```
last_four['Employment']
```

Out[ ]:
```
           1030        12050.0
           1031       120850.0
           1032        12610.0
           1033        26300.0
```

In [ ]:
```
employment_iter = iter(last_four['Employment'])
for i in employment_iter:
    print(i)
```

```
12050.0
120850.0
12610.0
26300.0
```

In [ ]:
```
last_four
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
1030        53-7073Wellhead Pumpers         detail        12050.0        $60,500
1031        53-7081Refuse and Recyclable Material Collectors         detail        120850.0        $42,620
1032        53-7121Tank Car, Truck, and Ship Loaders         detail        12610.0        $50,670
1033        53-7199Material Moving Workers, All Other         detail        26300.0        $37,770
```

In [ ]:
```
last_four[['Occupation title', 'Employment']]
```

Out[ ]:
```
Occupation title     Employment
1030Wellhead Pumpers        12050.0
1031Refuse and Recyclable Material Collectors       120850.0
1032Tank Car, Truck, and Ship Loaders        12610.0
1033Material Moving Workers, All Other        26300.0
```

In [ ]:
```
level_list = occupation['Level'].values
detail_index = [True if level_list[i]=='detail' else False for i in range(len(level_list)) ]
details = occupation[detail_index]
```

In [ ]:
```
my_filter = details['Employment']< 500
```

In [ ]:
```
type(my_filter)
```

Out[ ]:
```
nelta.LabeledList
```

In [ ]:
```
details[my_filter.values]
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
508         35-2013Cooks, Private Household         detail          320.0        $46,810
916         51-7032Patternmakers, Wood         detail          190.0        $60,750
```

second lambda

In [ ]:
```
rare_occ = details[my_filter.values]
for row in rare_occ:
    row[1] = ''.join(list(map(lambda s: s.upper(), row[1])))
rare_occ
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
508         35-2013COOKS, PRIVATE HOUSEHOLD         detail          320.0        $46,810
916         51-7032PATTERNMAKERS, WOOD         detail          190.0        $60,750
```

In [ ]:
```
rare_occ
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
508         35-2013COOKS, PRIVATE HOUSEHOLD         detail          320.0        $46,810
916         51-7032PATTERNMAKERS, WOOD         detail          190.0        $60,750
```

In [ ]:
```
for row in occupation:
    row[4] = int(row[4].replace('$', '').replace(',', ''))
gt_200000_list = occupation['Annual mean wage']>200000
occupation[gt_200000_list.values]
```

Out[ ]:
```
Occupation codeOccupation title          Level     EmploymentAnnual mean wage
383         29-1022Oral and Maxillofacial Surgeons         detail         4120.0         234990
384         29-1023  Orthodontists         detail         5040.0         237990
385         29-1024Prosthodontists         detail         530.0         214870
407         29-1211Anesthesiologists         detail         28590.0         271440
408         29-1215Family Medicine Physicians         detail         98590.0         214370
409         29-1216General Internal Medicine Physicians         detail         50600.0         210960
410         29-1218Obstetricians and Gynecologists         detail         18900.0         239120
412         29-1223  Psychiatrists         detail         25540.0         217100
413         29-1228Physicians, All Other; and Ophthalmologist...         detail        375390.0         218850
414         29-1248Surgeons, Except Ophthalmologists         detail         37900.0         251650
```