**Федеральное государственное образовательное бюджетное учреждение  
высшего образования  
«Финансовый университет при Правительстве РФ»**

**Отчет по практикуму по программированию. Вариант 9.**

Выполнил студент

Зайцев Никита Валерьевич

Группы ПИ20-2

Проверил доцент

Бочаров Михаил Иванович

Москва 2020

**Условие задачи:**

**Базовая часть:**

На базе модулей: csv, pickle и прямой работы с файлами реализовать следующий базовый функционал:

* функций **load\_table**, **save\_table** по загрузке/сохранению табличных данных во внутреннее представление модуля/из внутреннего представления модуля:
  + файла формата csv (отдельный модуль с **load\_table**, **save\_table** в рамках общего пакета)
  + файла формата pickle (отдельный модуль с **load\_table**, **save\_table** в рамках общего пакета), модуль использует структуру данных для представления таблицу, удобную автору работы.
  + текстового файла (только функция **save\_table** сохраняющая в текстовом файле представление таблицы, аналогичное выводу на печать с помощью функции **print\_table()).**

Примечание: внутреннее представление может базироваться на словаре, где по разным ключам хранятся ключевые «атрибуты» таблицы, а значения таблицы хранятся в виде вложенных списков. Студент может выбрать другое внутреннее представление таблицы (согласовав его с преподавателем), в том числе, студенты знакомые с ООП на Python, могут реализовать собственный класс для таблицы.

При определении api модулей максимально полно использовать возможности сигнатур функций на Python (значения по умолчанию, запаковка/распаковка, в т.ч. именованных параметров, возвращение множественных значений), интенсивно выполнять проверки и возбуждать исключительные ситуации.

* модуля с базовыми операциями над таблицами:
  + **get\_rows\_by\_number(start, [stop], copy\_table=False)** – получение таблицы из одной строки или из строк из интервала по номеру строки. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с исходным набором данных (**copy\_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.
  + **get\_rows\_by\_index(val1, … , copy\_table=False)** – получение новой таблицы из одной строки или из строк со значениями в первом столбце, совпадающими с переданными аргументами **val1, … , valN**. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с исходным набором данных (**copy\_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.
  + **get\_column\_types(by\_number=True)** – получение словаря вида *столбец:тип\_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by\_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.
  + **set\_column\_types(types\_dict, by\_number=True)** – задание словаря вида *столбец:тип\_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by\_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.
  + **get\_values(column=0)** – получение списка значений (типизированных согласно типу столбца) таблицы из столбца либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца)
  + **get\_value(column=0) –** аналог **get\_values(column=0)** для представления таблицы с одной строкой, возвращает не список, а одно значение (типизированное согласно типу столбца).
  + **set\_values(values, column=0)** – задание списка значений **values** для столбца таблицы (типизированных согласно типу столбца) либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца).
  + **set\_value(column=0) –** аналог **set\_values(value, column=0)** для представления таблицы с одной строкой, устанавливает не список значений, а одно значение (типизированное согласно типу столбца).
  + **print\_table()** – вывод таблицы на печать.
* Для каждой функции должно быть реализована генерация не менее одного вида исключительных ситуаций.

**Дополнительное задание:**

4) Реализовать автоматическое определение типа столбцов по хранящимся в таблице значениям.

Оформить как отдельную функцию и встроить этот функционал как опцию работы функции load\_table.

Сложность 1 или 2

6) Добавить набор функций add, sub, mul, div, которые обеспечат выполнение арифмитических

операций для столбцов типа int, float, bool. Продумать сигнатуру функций и изменения в

другие функции, которые позволят удобно выполнять арифметические операции со столбцами и присваивать результаты выч.

Реализовать реагирование на некорректные значения с помощью генерации исключительных ситуаций.

Сложность 2

8) Реализовать функцию merge\_tables(table1, table2, by\_number=True): в результате слияния создается таблица с

набором столбцов, соответствующих объединенному набору столбцов исходных таблиц. Соответствие строк ищется либо по их номеру

(by\_number=True) либо по значению индекса (1й столбец). При выполнении слияния возможно множество конфликтных ситуаций.

Автор должен их описать и определить допустимый

способ реакции на них (в т.ч. через дополнительные параметры функции и инициацию исключительных ситуаций).

Сложность 2

3) Реализовать функцию concat(table1, table2) и split(row\_number) склеивающую две таблицы или

разбивающую одну таблицу на 2 по номеру строки.

Сложность 1

**Листинг:**

*import* csv  
*import* pickle  
*import* copy  
*import* os  
  
'''  
tableModule by Kroll   
Table provided as dictionary (columns) in list (rows)  
'''  
  
  
*class* Table:  
 *def \_\_init\_\_*(*self*, filename='', table=*None*, delimiter\_csv=';'):  
 *self*.table = []  
 *if* table:  
 *self*.table = copy.deepcopy(table)  
 *return  
 if not* os.path.exists(filename):  
 *raise FileNotFoundError  
 if* '.csv' *in* filename:  
 *with open*(filename) *as* f\_obj:  
 reader = csv.DictReader(f\_obj, delimiter=delimiter\_csv)  
 *for* line *in* reader:  
 *self*.table.append(line)  
 *elif* '.pickle' *in* filename:  
 *with open*(filename, 'rb') *as* f\_obj:  
 unpickler = pickle.Unpickler(f\_obj)  
 *self*.table = unpickler.load()  
 *else*:  
 *raise Exception*('Unknown file extension')  
 *for* row *in self*.table:  
 *for* k *in* row.keys():  
 *if* (*type*(row[k]) != *int*) *and* row[k].isdigit():  
 row[k] = *int*(row[k])  
  
 *print*('Imported table:\n', *self*.table)  
  
 *def \_\_str\_\_*(*self*):  
 *return str*(*self*.table)  
  
 *def* \_\_check\_rows(*self*, row):  
 *if type*(row) != *int*:  
 *raise TypeError*('Row variable is not int')  
 *if* row >= *len*(*self*.table) *or* row <= -1:  
 *raise Exception*('Not correct row selected')  
  
 *def* \_\_check\_column(*self*, column):  
 *if type*(column) != *int*:  
 *raise TypeError*('Column variable is not int')  
 *if* column >= *len*(*self*.table[0].keys()) *or* column <= -1:  
 *raise Exception*('Not correct column selected')  
  
 *def* \_\_check\_bool(*self*, bool\_variable, name):  
 *if* (bool\_variable <= -1 *or* bool\_variable >= 2) *and type*(bool\_variable) != *bool*:  
 *raise TypeError*(name + ' variable more than 2 or not bool')  
  
 *def* save\_table(*self*, filename, delimiter\_csv=';'):  
 *if* '.csv' *in* filename:  
 *with open*(filename, "w", newline='') *as* f\_obj:  
 fieldnames = *self*.table[0].keys()  
 writer = csv.DictWriter(f\_obj, delimiter=delimiter\_csv, fieldnames=fieldnames)  
 writer.writeheader()  
 *for* row *in self*.table:  
 writer.writerow(row)  
 *elif* '.pickle' *in* filename:  
 *with open*('data.pickle', 'wb') *as* f\_obj:  
 pickle.dump(*self*.table, f\_obj)  
 *else*:  
 *raise Exception*('Unknown file extension')  
 *print*('Saved table:\n', *self*.table)  
 *print*('Saved at', filename)  
  
 *def* add(*self*, row, col, value):  
 *try*:  
 *self*.table[row][col] += value  
 *except Exception as* e:  
 *print*(e)  
  
 *def* sub(*self*, row, col, value):  
 *try*:  
 *self*.table[row][col] -= value  
 *except Exception as* e:  
 *print*(e)  
  
 *def* mul(*self*, row, col, value):  
 *try*:  
 *self*.table[row][col] %= value  
 *except Exception as* e:  
 *print*(e)  
  
 *def* div(*self*, row, col, value):  
 *try*:  
 *self*.table[row][col] //= value  
 *except Exception as* e:  
 *print*(e)  
  
 *def* merge\_tables(*self*, table, by\_number=*True*):  
 keys = *list*(*self*.table[0].keys())  
 *for* key *in* table[0].keys():  
 *if* key *not in* keys:  
 keys.append(key)  
 *print*(keys)  
 *for* row1, row2 *in zip*(*self*.table, table):  
 *for* key1, key2 *in zip*(*list*(row1.keys()), *list*(row2.keys())):  
 *if* key1 != key2:  
 row1[key2] = row2[key2]  
 *print*(*self*.table)  
  
 *def* get\_rows\_by\_number(*self*, start, stop=*None*, copy\_table=*False*):  
 *if type*(start) != *int*:  
 *raise TypeError*('Start variable not int')  
 *self*.\_\_check\_bool(copy\_table, 'Copy\_table')  
  
 table = []  
 *if* stop:  
 *if type*(stop) != *int*:  
 *raise TypeError*('Stop variable is not int')  
 table = *self*.table[start:stop]  
 *else*:  
 table.append(*self*.table[start])  
 *if* copy\_table:  
 table = copy.deepcopy(table)  
 *return* Table(table=table)  
  
 *def* split(*self*, row\_number):  
 *self*.\_\_check\_rows(row\_number)  
  
 table = *self*.table[row\_number:]  
 *self*.table = copy.deepcopy(*self*.table[:row\_number])  
 *return* Table(table=table)  
  
 *def* concat(*self*, table):  
 *if* table[0].keys() != *self*.table[0].keys():  
 *raise KeyError*('No correct key in provided table!')  
 *self*.table.append(table)  
  
 *def* get\_rows\_by\_index(*self*, \*index, copy\_table=*False*):  
 *self*.\_\_check\_bool(copy\_table, 'Copy\_table')  
  
 table = []  
 *for* i *in* index:  
 *for* row *in self*.table:  
 key = *list*(row.keys())[0]  
 *print*(row[key], i)  
 *if str*(row[key]) == *str*(i):  
 table.append(row)  
 *if* copy\_table:  
 table = copy.deepcopy(table)  
 *return* Table(table=table)  
  
 *def* get\_column\_types(*self*, by\_number=*True*):  
 *self*.\_\_check\_bool(by\_number, 'By\_number')  
  
 table = []  
 *for* i *in range*(*len*(*self*.table)):  
 table.append({})  
 keys = *self*.table[i].keys()  
 *for* k *in* keys:  
 *if* by\_number:  
 table[i][*list*(keys).index(k)] = *type*(*self*.table[i][k])  
 *else*:  
 table[i][k] = *type*(*self*.table[i][k])  
 *return* Table(table=table)  
  
 *def* set\_column\_types(*self*, types\_dict, by\_number=*True*):  
 *if len*(types\_dict.keys()) != *len*(*self*.table[0].keys()):  
 *raise Exception*('Not equal length between dicts')  
 *self*.\_\_check\_bool(by\_number, 'By\_number')  
  
 *for* i *in range*(*len*(*self*.table)):  
 keys\_td = types\_dict.keys()  
 keys = *self*.table[i].keys()  
 *for* k *in* keys\_td:  
 *if* by\_number:  
 *self*.table[i][*list*(keys)[k]] = types\_dict[k](*self*.table[i][*list*(keys)[k]])  
 *else*:  
 *self*.table[i][k] = types\_dict[k](*self*.table[i][k])  
  
 *def* get\_values(*self*, column=0):  
 *self*.\_\_check\_column(column)  
  
 table = []  
 *for* i *in range*(*len*(*self*.table)):  
 table.append({})  
 key = *list*(*self*.table[i].keys())[column]  
 table[i][key] = *self*.table[i][key]  
 *return* Table(table=table)  
  
 *def* get\_value(*self*, column=0):  
 *self*.\_\_check\_column(column)  
  
 table = [{}]  
 key = *list*(*self*.table[0].keys())[column]  
 table[0][key] = *self*.table[0][key]  
 *return* Table(table=table)  
  
 *def* set\_values(*self*, \*values, column=0):  
 *self*.\_\_check\_column(column)  
  
 *for* i *in range*(*len*(values)):  
 key = *list*(*self*.table[i].keys())[column]  
 *self*.table[i][key] = values[i]  
  
 *def* set\_value(*self*, value, column=0):  
 *self*.\_\_check\_column(column)  
  
 key = *list*(*self*.table[0].keys())[column]  
 *self*.table[0][key] = value  
  
 *def* print\_table(*self*):  
 *print*(*self*.table)  
  
  
*def* test():  
 a = Table('/Users/nikitazaytsev/git/FA/1 Курс/ПрактПрог/Ex3/data.csv')  
 t = Table('/Users/nikitazaytsev/git/FA/1 Курс/ПрактПрог/Ex3/data1.csv')  
 a.merge\_tables(t.table)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 test()

Интернет-источники

1. <https://docs.python.org/3/>