**Текущее решение:**

Программа перебирает файлы в указанной папке, из каждого файла достает данные план и факт каждого сотрудника, подсчитывает успешность каждого проекта, затем высчитывает среднее арифметические успешности по всем проектам и выводит в порядке убывания успешности всех сотрудников, либо в консоль (при запуске программы в IDE).

**Перспективы расширения функционала:**

Можно реализовать анализ успешности всех руководителей проектов и анализ каждого проекта в отдельности. Для этого можно использовать данные из словаря, который уже создан и хранит все данные по сотрудникам. Если анализировать успешность проекта смотрим на количество сотрудников и количество затраченных дней каждым сотрудником. Если анализировать успешность руководителей, берем все проекты которыми управлял сотрудник и вычисляем успешность этих проектов.

**Сторонние библиотеки:**

openpyxl — библиотека для записи и чтения файлов excel

<https://openpyxl.readthedocs.io/en/stable/>

glob - предоставляет функцию для создания списков файлов по шаблону из найденных в каталоге

<https://docs.python.org/3/library/glob.html>

**Исходный код**

**main.pyw**

**import** openpyxl  
**import** glob  
**from** Employee **import** \*  
**from** Operation **import** \*  
  
**def** main():  
 *# Перебор каждого файла в папке*  
 *# В файле берем только первый лист*  
**for** file **in** glob.glob(**"\*.xlsx"**):  
 book = openpyxl.open(file, read\_only=**True**)  
 sheet = book.active  
  
 *# номер колоки с которой произойдет считывание (счетчик начинается с 0)*  
 *# задается в зависимости от расположения данных в ячейках (строках/столбцах) файла excel*  
start\_col = 4  
  
 *# перебираем колонки через одну, так как 1ая - план сотрудника, 2-ая - факт,*  
 *# выходит что каждый новый сотрудник начниается на нечетной колонке (1, 3, ...)*  
**for** col **in** range(start\_col, sheet.max\_column, 2):  
 name\_employee = Operation.remove\_pass(sheet[1][col].value)  
  
 *# Добавляем сотрудника в словарь если его там нет*  
**if** name\_employee **not in** Employee.dict\_employee:  
 Employee.dict\_employee[name\_employee] = Employee(name\_employee)  
 **else**:  
 **continue**  
  
*# Перебор со 2ой и до последней строки файла, первая строка с данными ФИО*  
**for** row **in** range(2, sheet.max\_row + 1):  
 *# Увеличиваем количество проектов на 1 для этого сотрудника*  
Employee.dict\_employee[name\_employee].count\_project += 1  
  
 *# Cокращение:*  
plan = sheet[row][col].value  
 fact = sheet[row][col + 1].value  
  
 *# Добавляем в словарь количество смен план*  
**if** type(plan) == int **and** plan > 0:  
 Employee.dict\_employee[name\_employee].plan += plan  
  
 *# Добавляем в словарь количество смен факт*  
**if** type(fact) == int **and** fact > 0:  
 Employee.dict\_employee[name\_employee].fact += fact  
  
 *# Вычисление и запись успешности сотруднику*  
Operation.success\_rate(Employee.dict\_employee[name\_employee], plan, fact)  
  
*# Вывод результата*  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()  
 Operation.get\_sorted\_list(Employee.dict\_employee)

**Employee.pyw**

*# Создание сотрудника*  
*#*  
*# name - имя сотруника, count\_project - количество проектов у сотрудника,*  
*# plan - план, fact - факт, success\_rate - коэффициент успешности*  
**class** Employee:  
 *# Хранит все экземпляры класса:*  
dict\_employee = {}  
  
 **def** \_\_init\_\_(self, name, count\_project=0, plan=0, fact=0, success\_rate=0):  
 self.\_\_name = name  
 self.\_\_count\_project = count\_project  
 self.\_\_plan = plan  
 self.\_\_fact = fact  
 self.\_\_success\_rate = success\_rate  
  
 **def** get\_name(self):  
 **return** self.\_\_name  
  
 **def** get\_count\_project(self):  
 **return** self.\_\_count\_project  
  
 **def** get\_plan(self):  
 **return** self.\_\_plan  
  
 **def** get\_fact(self):  
 **return** self.\_\_fact  
  
 **def** get\_success\_rate(self):  
 **return** self.\_\_success\_rate  
  
 **def** set\_name(self, value):  
 **if not** isinstance(value, str):  
 **raise** ValueError(**"Имя должно быть строкой"**)  
 self.\_\_name = value  
  
 **def** set\_count\_project(self, value):  
 **if not** isinstance(value, (int, float)):  
 **raise** ValueError(**"Количество проектов должно бысть числом"**)  
 self.\_\_count\_project = value  
  
 **def** set\_plan(self, value):  
 **if not** isinstance(value, (int, float)):  
 **raise** ValueError(**"Количество проектов должно бысть числом"**)  
 self.\_\_plan = value  
  
 **def** set\_fact(self, value):  
 **if not** isinstance(value, (int, float)):  
 **raise** ValueError(**"Количество проектов должно бысть числом"**)  
 self.\_\_fact = value  
  
 **def** set\_success\_rate(self, value):  
 **if not** isinstance(value, (int, float)):  
 **raise** ValueError(**"Количество проектов должно бысть числом"**)  
 self.\_\_success\_rate = value  
  
 name = property(fget=get\_name, fset=set\_name)  
 count\_project = property(fget=get\_count\_project, fset=set\_count\_project)  
 plan = property(fget=get\_plan, fset=set\_plan)  
 fact = property(fget=get\_fact, fset=set\_fact)  
 success\_rate = property(fget=get\_success\_rate, fset=set\_success\_rate)

**Operation.py**

**class** Operation: *# Удаление переносов строки и пробелов после ФИО*@staticmethod **def** remove\_pass(value): index\_new\_line = value.index(**'\n'**) index\_point = value.index(**'.'**) index\_end\_name = value.index(**'.'**, index\_point + 1) **if** index\_new\_line == -1: **if** index\_end\_name == -1: **return** value **else**: **return** value[:index\_end\_name + 1] **else**: **return** value[:index\_new\_line] *# Вычисляем успешность сотрудника по формуле:* *# успешность = (текущая успешность + план/факт) / 2 (2 - так как расчитываем средн. арифм. двух коэф.)* *#* *# value - Экземпляр класса Employee, plan - план, fact - факт*@staticmethod **def** success\_rate(value, plan, fact): **if** (type(plan) != int **or** plan == 0) **and** (type(fact) != int **or** fact == 0): *# Сотрудник не участвовал в проекте, убавляем количество проектов у него*value.count\_project -= 1 **elif** (type(plan) != int **or** plan == 0) **and** fact > 0 **or** plan == fact: *# Сотрудник приступил к работе внепланово или выполнили работу в срок (факт = план)*value.success\_rate = (value.success\_rate + 1) / 2 **elif** plan > 0 **and** type(fact) != int **or** fact == 0: *# Сотруднику дали работу, но он не приступил к ней, КПД сотрудника в этом проекте 0*value.success\_rate = (value.success\_rate + 0) / 2 **else**: *# Базовое вычисление успешности*value.success\_rate = (value.success\_rate + plan / fact) / 2 *# Вывод отсортированного результата успешности сотрудников* *#* *# dict\_employee - словарь с экземплярами класса*@staticmethod **def** get\_sorted\_list(dict\_employee): *# Создание отсортированного списка только с именем сотрудника и его успешностью*list\_employee = [] count = 0 **for** employee **in** dict\_employee: temp\_list = [0, 0] temp\_list[0] = format(dict\_employee[employee].success\_rate, **'.3f'**) temp\_list[1] = dict\_employee[employee].name list\_employee.append(temp\_list) count += 1 list\_employee.sort(reverse=**True**) *# Вывод в консоль*count\_employee = 1 print(**'№ Успешность Ф.И.О.'**) print(**'- - - - - - - - - - - - - - -'**) **for** success, employee **in** list\_employee: print(count\_employee, **'. '**, success, **' '**, employee) count\_employee += 1 *# Убрать документирование для создания текстового файла с результатами анализа*Operation.set\_text\_file(list\_employee) *# Создание текстового файла и запись в него полученных данных*@staticmethod **def** set\_text\_file(list\_employee): my\_file = open(**'Анализ успешности сотрудников.txt'**, **'w'**) count\_employee = 1 my\_file.write(**'№ Успешность Ф.И.О.\n'**) my\_file.write(**'- - - - - - - - - - - - - - -\n'**) **for** item **in** list\_employee: my\_file.write(str(count\_employee)) my\_file.write(**'. '**) my\_file.write(str(item[0])) my\_file.write(**' '**) my\_file.write(str(item[1])) my\_file.write(**'\n'**) count\_employee += 1 my\_file.close()