|  |  |
| --- | --- |
|  | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет** **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Отчёт по лабораторной работе №3 по курсу**

**«Разработка Интернет-Приложений»**

**Тема работы: "Функциональные возможности языка Python"**

Выполнил: Забурунов Леонид, РТ5-51Б

Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15 октября 2020 г.

ЗАЧТЕНО / НЕ ЗАЧТЕНО\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

# Цель лабораторной работы.

Изучение основных конструкций и возможностей ЯП Python при функциональном подходе.

# Задание на лабораторную работу.

Для лабораторной работы №3 необходимо разработать консольное приложение, включающее в себя несколько модулей, представляющих из себя объекты, работающие в парадигме функционального программирования. Требуется создать:

1. Обработчик коллекции (генератор), работающий со списком словарей и выдающий те словари, в которых содержатся задаваемые в виде параметров вызова ключи;
2. Генератор случайных целых чисел;
3. Итератор, исключающий из исходной коллекции (списка или генератора) дубликаты;
4. Процедуру сортировки списка чисел по их абсолютному значению с помощью расширений коллекций (с использованием лямбда-выражения и без него);
5. Декоратор функции, предварительно оповещающей об исполняющейся процедуре и возвращающий её значения в удобном для чтения формате;
6. Контекстный менеджер в виде таймера (в двух исполнениях: как класс и как функция с использованием библиотеки “contextlib”);
7. Обработчик данных из файла в формате JSON, используя при этом созданные в предыдущих частях лабораторной работы элементы.

# Ход выполнения лабораторной работы.

Первым делом создаём обработчик коллекции:

**def** field(dicts, \*args):  
 **if** len(args) > 1:  
 **for** dictionary **in** dicts:  
 **for** arg **in** args:  
 **if** arg **not in** dictionary.keys():  
 **break  
 else**:  
 **yield** dictionary  
 **elif** len(args) > 0:  
 **for** dictionary **in** dicts:  
 **for** dictKey **in** dictionary.keys():  
 **for** arg **in** args:  
 **if** dictKey == arg:  
 **yield** dictionary.get(dictKey)

Далее – генератор случайных чисел:

**def** genRandom(amountOfNums, minNum, maxNum):  
 **for** i **in** range(amountOfNums):  
 **yield** random.randint(minNum, maxNum)

Итератор, пропускающий дубликаты:

**class** UniqueIterator:  
 **def** \_\_init\_\_(self, \*\*kwargs):  
 self.currentIndex = 0  
 self.usedElements = set()  
 *# Насколько правильно использован kwargs?* self.ignoreCase = kwargs.get(**"ignoreCase"**)  
 data = kwargs.get(**"data"**)  
 **if** isinstance(data, GeneratorType):  
 *# Если на вход получен генератор, то необходимо создать коллекцию* self.data = list()  
 **for** generatorElement **in** data:  
 self.data.append(generatorElement)  
 **else**:  
 self.data = data  
 self.length = len(self.data)  
  
 **def** \_\_iter\_\_(self):  
 **return** self  
  
 **def** \_\_next\_\_(self):  
 **while True**:  
 **if** self.currentIndex >= self.length:  
 **raise** StopIteration  
 **else**:  
 currentElement = self.data[self.currentIndex]  
 self.currentIndex = self.currentIndex + 1  
 **if** isinstance(currentElement, str) **and** self.ignoreCase == **True**:  
 condition = currentElement.lower()  
 **else**:  
 condition = currentElement  
 **if** (condition) **not in** self.usedElements:  
 self.usedElements.add(condition)  
 **return** currentElement

Сортировка списка:

generator = genRandom(20, -10, 10)  
data = list()  
**for** num **in** generator:  
 data.append(num)  
print(data)  
*# Насколько верна задумка?  
# без лямбда-выражения*print(**"No lambda:"**)  
print(list(i[1] **for** i **in** reversed(sorted(zip([abs(num) **for** num **in** data], data)))))  
*# с лямбда-выражением*print(**"Lambda:"**)  
print(list(i[1] **for** i **in** reversed(sorted(zip(map(**lambda** x: abs(x), data), data)))))

Декоратор:

**def** printResult\_decorator(funcToPrint):  
 **def** decorating(\*args, \*\*kwargs):  
 funcResult = funcToPrint(\*args, \*\*kwargs)  
 print(**"Function {0} returns: "**.format(funcToPrint.\_\_name\_\_), end = **''**)  
 **if** isinstance(funcResult, list):  
 print()  
 **for** listElement **in** funcResult:  
 print(listElement)  
 **elif** isinstance(funcResult, dict):  
 print()  
 **for** dictElement **in** funcResult.items():  
 print(**"Key = {0} -> Value = {1}"**.format(dictElement[0], dictElement[1]))  
 **else**:  
 print(funcResult)  
 print(**"\_"** \* 20)  
 **return** funcResult  
 **return** decorating

Таймер:

**class** Timer\_CM\_Class:  
 **def** \_\_init\_\_(self):  
 print(**"Created (class instance)"**)  
 self.timer = **None  
  
  
 def** \_\_enter\_\_(self):  
 self.timer = time.time()  
  
  
 **def** \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 self.timer = time.time() - self.timer  
 print(**"Time: {0}"**.format(self.timer))  
 print(**"Left (class instance)"**)  
  
  
@contextmanager  
**def** Timer\_CM\_Lib():  
 print(**"Entered (func with contextlib)"**)  
 timer = time.time()  
 [i **for** i **in** genRandom(80000, -10, 10)]  
 **yield** time.time() - timer  
 print(**"Left (func with contextlib)"**)

Обработка данных:

**import** json  
**from** lab\_python\_fp.print\_result **import** printResult\_decorator  
**from** lab\_python\_fp.unique **import** UniqueIterator  
**from** lab\_python\_fp.cm\_timer **import** Timer\_CM\_Class  
**from** lab\_python\_fp.field **import** field  
**from** lab\_python\_fp.gen\_random **import** genRandom  
  
@printResult\_decorator  
**def** f1(dataFile):  
 *# Правомерно ли с точки зрения задания модифицировать исходный регистр?* **return** sorted(vocation.capitalize() **for** vocation **in** UniqueIterator(data = field(dataFile, **"job-name"**), ignoreCase = **True**))  
  
  
@printResult\_decorator  
**def** f2(sortedData):  
 **return** list(filter(**lambda** x : **"Программист" in** x, sortedData))  
  
  
@printResult\_decorator  
**def** f3(filteredData):  
 **return** list(map(**lambda** x : x + **" с опытом Python."**, filteredData))  
  
  
@printResult\_decorator  
**def** f4(modifiedData):  
 **return** list(str(info[0]) + **" Зарплата: "** + str(info[1]) + **"!" for** info **in** zip(modifiedData,genRandom(len(modifiedData), 100000, 200000)))  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 data = list()  
 **with** open(**"d:\\study\\5\_Семестр\\РИП\\Lab3\\data\_light.json"**, **'r'**, encoding = **'utf8'**) **as** data\_file:  
 data = json.load(data\_file)  
 **with** Timer\_CM\_Class():  
 f4(f3(f2(f1(data))))

# Результаты работы

Поскольку размер коллекции составляет тысячи записей, скриншоты с демонстрацией выведения списка профессий без дубликатов (состоящего так же из нескольких тысяч элементов) не прилагаются. Результаты последующей обработки:

