**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3-4

“ Функциональные возможности языка Python”

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверила: |
| студент группы ИУ5-34Б: |  | преподаватель каф. ИУ5 |
| Стукалов Иван Дмитриевич |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

2022

Описание задания:

1. Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.

Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

1. Необходимо реализовать генератор, который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.
2. Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

При реализации необходимо использовать конструкцию \*\*kwargs.

Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

1. В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.

Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

1. Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.

Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

1. Необходимо написать контекстные менеджеры, которые считают время работы блока кода и выводят его на экран.
2. В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.

В файле data\_light.json содержится фрагмент списка вакансий.

Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер cm\_timer\_1 выводит время работы цепочки функций.

Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.

Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию map.

Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

Текст программы:

// файл field.py

def field(dictionary, \*args):  
 assert len(args) > 0  
 new\_list = []  
 new\_dict = {}  
 for i in dictionary:  
 for j in args:  
 if i.get(j) != None:  
 new\_dict[j] = i.get(j)  
 new\_list.append(dict(new\_dict))  
 new\_dict = {}  
 return new\_list

//файл gen\_random.py

import random  
  
# генератор случайных чисел  
  
def gen\_random(count, min, max):  
 rand\_list = list()  
 for i in range (count):  
 rand\_list.append(random.randint(min, max))  
 return rand\_list

// файл unique.py

# Итератор для удаления дубликатов  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 # Нужно реализовать конструктор  
 # В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore\_case,  
 # в зависимости от значения которого будут считаться одинаковыми строки в разном регистре  
 # Например: ignore\_case = True, Aбв и АБВ - разные строки  
 # ignore\_case = False, Aбв и АБВ - одинаковые строки, одна из которых удалится  
 # По-умолчанию ignore\_case = False  
  
 self.ignore\_case = kwargs.get("ignore\_case")  
 self.list = items  
  
  
 def \_\_next\_\_(self):  
 # Нужно реализовать \_\_next\_\_  
 if not self.ignore\_case:  
 if self.current < len(self.list):  
 if self.current == 0:  
 self.current += 1  
 return self.list[self.current - 1]  
  
 is\_last = False  
 i = 0  
 while i < self.current:  
 if self.list[i] == self.list[self.current] and len(self.list) > self.current + 1:  
 self.current += 1  
 i = -1  
 elif self.list[i] == self.list[self.current] and len(self.list) == self.current + 1:  
 is\_last = True  
 i += 1  
  
 if not is\_last:  
 result = self.list[self.current]  
 self.current += 1  
 return result  
 else:  
 if self.current < len(self.list):  
 if self.current == 0:  
 self.current += 1  
 return self.list[self.current - 1]  
  
 is\_last = False  
 i = 0  
 while i < self.current:  
 if self.list[i].upper() == self.list[self.current].upper() and len(self.list) > self.current + 1:  
 self.current += 1  
 i = -1  
 elif self.list[i].upper() == self.list[self.current].upper() and len(self.list) == self.current + 1:  
 is\_last = True  
 i += 1  
  
 if not is\_last:  
 result = self.list[self.current]  
 self.current += 1  
 return result  
  
  
 def \_\_iter\_\_(self):  
 self.current = 0  
 return self

// файл sort.py

# сортировка  
  
def sortedList(list, parameter=lambda x: x):  
 for i in range(len(list)):  
 for j in range(len(list) - 1):  
 if parameter(list[j]) < parameter(list[j + 1]):  
 list[j], list[j + 1] = list[j + 1], list[j]  
  
 return list

// файл print\_result.py

# выводит на экран результат выполнения функции  
  
def print\_result(function):  
 def wrapper(\*args, \*\*kwargs):  
 print(function.\_\_name\_\_)  
 result = function(\*args, \*\*kwargs)  
  
 if isinstance(result, list):  
 for i in result:  
 print(i)  
 elif isinstance(result, dict):  
 for i in result:  
 print("{} = {}".format(i, result[i]))  
 else:  
 print(result)  
 return wrapper

//файл cm\_timer.py

# время работы кода  
  
import time  
from contextlib import contextmanager  
  
class TimerError(Exception): #5  
 *"""Пользовательское исключение, используемое для сообщения об ошибках при использовании класса Timer"""*class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.\_start\_time = None  
  
 def \_\_enter\_\_(self):  
 *"""Запуск нового таймера"""* if self.\_start\_time is not None:  
 raise TimerError(f"Таймер уже работает. Используйте .stop() чтобы его остановить")  
  
 self.\_start\_time = time.perf\_counter()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_value, exc\_tb):  
 *"""Отстановить таймер и сообщить о времени вычисления"""* if self.\_start\_time is None:  
 raise TimerError(f"Таймер не работает. Используйте .start() для его запуска")  
  
 elapsed\_time = time.perf\_counter() - self.\_start\_time  
 self.\_start\_time = None  
 print(f"Вычисление заняло {elapsed\_time:0.4f} секунд")  
  
  
@contextmanager  
def cm\_timer\_2(\*args, \*\*kwargs):  
 *"""Запуск нового таймера"""* start\_time = time.perf\_counter()  
 try:  
 yield start\_time  
  
 finally:  
 """Отстановить таймер и сообщить о времени вычисления"""  
 elapsed\_time = time.perf\_counter() - start\_time  
 print(f"Вычисление заняло {elapsed\_time:0.4f} секунд")

//файл process\_data.py

import json  
import sys  
  
# Сделаем другие необходимые импорты  
from lab\_python\_fp.cm\_timer import cm\_timer\_1  
from lab\_python\_fp.field import field  
from lab\_python\_fp.gen\_random import gen\_random  
from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.sort import sorted  
from lab\_python\_fp.unique import Unique  
  
path = "C:\work\Бомонка\3 сем\ВКИД\lab\_3\data\_light.json"  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
with open(path) as f:  
 data = json.load(f)  
  
# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`  
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку  
# В реализации функции f4 может быть до 3 строк  
  
@print\_result  
def f1(arg):  
 unique = Unique(arg)  
 unique\_it = Unique.\_\_iter\_\_(unique)  
 print(Unique.\_\_next\_\_(unique\_it))  
  
  
@print\_result  
def f2(arg):  
 raise NotImplemented  
  
  
@print\_result  
def f3(arg):  
 raise NotImplemented  
  
  
@print\_result  
def f4(arg):  
 raise NotImplemented  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f1(data)

//файл main.py

from lab\_python\_fp.field import field  
from lab\_python\_fp.gen\_random import gen\_random  
from lab\_python\_fp.unique import Unique  
from lab\_python\_fp.sort import sortedList  
from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.cm\_timer import cm\_timer\_1  
from lab\_python\_fp.cm\_timer import cm\_timer\_2  
# from lab\_python\_fp.process\_data import f1  
  
@print\_result  
def fun1():  
 return 1  
  
@print\_result  
def fun2():  
 return 'iu5'  
  
@print\_result  
def fun3():  
 return [1, 2, 3]  
  
@print\_result  
def fun4():  
 return {'a': 1, 'b': 2}  
  
def main():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]  
 print(field(goods, 'title', 'price'))  
  
 print(gen\_random(3, 1, 10))  
  
 count\_list = ["aaa", "b", "c", "b", "A", "b", "AAA", "Aaa", "a"]  
 unique = Unique(count\_list)  
 unique\_it = Unique.\_\_iter\_\_(unique)  
  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
 print(Unique.\_\_next\_\_(unique\_it))  
  
 list = [-3, 6, -2, 5, -4, 3, -10, -53, 8]  
 print(sortedList(list))  
 print(sortedList(list, lambda x: abs(x)))  
  
 with cm\_timer\_1():  
 fun1()  
 fun2()  
 fun3()  
 fun4()  
  
 with cm\_timer\_2():  
 fun1()  
 fun2()  
 fun3()  
 fun4()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Пример выполнения программы:



