**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **«Информатики и систем управления»** |
| **КАФЕДРА** | Системы обработки информации и управления |

Дисциплина «Разработка Интернет-Приложений»

**ЛАБОРАТОРНАЯ РАБОТА №3**

Функциональные возможности языка Python

|  |  |
| --- | --- |
| Студент | Сахарова Е. К. ИУ5-52Б |
| Преподаватель | Гапанюк Ю. Е. |

**Цель лабораторной работы:** изучение возможностей функционального программирования в языке Python.

**Задание:**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

*Задача 1 (файл field.py)*

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. Пример:

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

{'title': 'Диван для отдыха', 'color': 'black'}

]

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'

field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количество 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

*Задача 2 (файл gen\_random.py)*

Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

*Задача 3 (файл unique.py)*

* 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.

Пример:

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(1, 3, 10)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

*Задача 4 (файл sort.py)*

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted. Пример:

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

*Задача 5 (файл print\_result.py)*

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.

*Задача 6 (файл cm\_timer.py)*

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран. Пример:

with cm\_timer\_1():

sleep(5.5)

После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами.

*Задача 7 (файл process\_data.py)*

* 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.

**Код программы:**

*field.py*

**def** field(items, \*args):  
 **assert** len(args) > 0  
 **if** len(args) == 1:  
 return\_list = [item[key] **for** item **in** items **for** key **in** item **if** key **in** args]  
 *#print(\*[item[key] for item in items for key in item if key in args], sep=', ')* **else**:  
 return\_list = []  
 **for** item **in** items:  
 return\_list.append({key: item[key] **for** key **in** args})  
 *#print(return\_list)* **return** return\_list  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 goods = [  
 {**'title'**: **'Ковер'**, **'price'**: 2000, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'price'**: 5300, **'color'**: **'black'**}  
 ]  
  
 print(\*field(goods, **'title'**), sep=**', '**) *# должен выдавать 'Ковер', 'Диван для отдыха'* print(\*field(goods, **'title'**, **'price'**), sep=**', '**)  
 *# должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}*

*gen\_random.py*

**import** random  
  
  
*# Hint: типовая реализация занимает 2 строки***def** gen\_random(num\_count, begin, end):  
 list\_ans = [random.randint(begin, end) **for** \_ **in** range(num\_count)]  
 *# print(\*list\_ans, sep=', ')* **return** list\_ans  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(\*gen\_random(5, 1, 3), sep=**', '**) *# должен выдать выдать 5 случайных чисел  
 # в диапазоне от 1 до 3, например 2, 2, 3, 2, 1*

*unique.py*

**from** github.lab3.lab\_python\_fp.gen\_random **import** gen\_random  
  
  
**class** Unique:  
 *"""Итератор, оставляющий только уникальные значения."""* **def** \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = set()  
 self.items = items  
 self.index = 0  
 self.ignore\_case = [kwargs[key] **for** key **in** kwargs] **if** len(kwargs) > 0 **and** kwargs[**'ignore\_case'**] **is True** \  
 **else False  
  
 def** \_\_iter\_\_(self):  
 **return** self  
  
 **def** \_\_next\_\_(self):  
 **while True**:  
 **if** self.index >= len(self.items):  
 **raise** StopIteration  
 **else**:  
 *#if self.ignore\_case:  
 #self.items = list(map(lambda \_: str(\_).lower(), self.items))* **if** self.ignore\_case:  
 current = str(self.items[self.index]).lower()  
 **else**:  
 current = self.items[self.index]  
 self.index += 1  
 **if** current **not in** self.used\_elements:  
 *# Добавление в множество производится  
 # с помощью метода add* self.used\_elements.add(current)  
 **return** current  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**'Task 1'**)  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 **for** item **in** Unique(data):  
 print(item, end=**' '**)  
  
 print(**'\nTask 2'**)  
 data = gen\_random(5, 3, 10)  
 print(\*data, sep=**', '**)  
 **for** item **in** Unique(data):  
 print(item, end=**' '**)  
  
 print(**'\nTask 3'**)  
 data = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
 **for** item **in** Unique(data):  
 print(item, end=**' '**)  
  
 print(**'\nTask 4'**)  
 **for** item **in** Unique(data, ignore\_case=**True**):  
 print(item, end=**' '**)

*sort.py*

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 result = sorted(data, reverse=**True**, key=abs)  
 print(result)  
  
 result\_with\_lambda = sorted(data, reverse=**True**, key=**lambda** \_: abs(\_))  
 print(result\_with\_lambda)

*print\_result.py*

**def** print\_result(func\_to\_decorate):  
  
 **def** wrapped\_func(\*args, \*\*kwargs):  
 print(func\_to\_decorate.\_\_name\_\_)  
 result = func\_to\_decorate(\*args, \*\*kwargs)  
 **if** type(result) == dict:  
 **for** key **in** result:  
 *# print('{} = {}'.format(key, result[key]))* print(**'{}, зарплата {} руб.'**.format(key, result[key]))  
 **elif** type(result) == list:  
 **for** value **in** result:  
 print(value)  
 **else**:  
 print(result)  
 **return** result  
  
 **return** wrapped\_func  
  
  
@print\_result  
**def** test\_1():  
 **return** 1  
  
  
@print\_result  
**def** test\_2():  
 **return 'iu5'**@print\_result  
**def** test\_3():  
 **return** {**'a'**: 1, **'b'**: 2}  
  
  
@print\_result  
**def** test\_4():  
 **return** [1, 2]  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**'!!!!!!!!'**)  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

*cm\_timer.py*

**import** time  
**from** contextlib **import** contextmanager  
  
  
**class** cm\_timer1:  
 **def** \_\_init\_\_(self):  
 self.start = time.time()  
  
 **def** \_\_enter\_\_(self):  
 **return  
  
 def** \_\_exit\_\_(self, \*args):  
 print(**'Прошло времени: '**, time.time() - self.start)  
  
  
@contextmanager  
**def** cm\_timer2():  
 start = time.perf\_counter()  
 **yield lambda**: time.perf\_counter() - start  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer1():  
 time.sleep(2)  
 **with** cm\_timer2() **as** t:  
 time.sleep(2)  
 print(**'Прошло времени: '**, t())

*process\_data.py*

**import** json  
**from** github.lab3.lab\_python\_fp.cm\_timer **import** cm\_timer1  
**from** github.lab3.lab\_python\_fp.field **import** field  
**from** github.lab3.lab\_python\_fp.gen\_random **import** gen\_random  
**from** github.lab3.lab\_python\_fp.print\_result **import** print\_result  
**from** github.lab3.lab\_python\_fp.unique **import** Unique  
  
path = **'data\_light.json'  
  
with** open(path, **'r'**, encoding=**'utf8'**) **as** f:  
 data = json.load(f)  
@print\_result  
**def** f1(arg):  
 **return** sorted(Unique(field(arg, **'job-name'**), ignore\_case=**True**), key=**lambda** \_: str(\_).lower())  
  
@print\_result  
**def** f2(arg):  
 **return** list(filter(**lambda** \_: str(\_).startswith(**'программист'**), arg))  
@print\_result  
**def** f3(arg):  
 **return** list(map(**lambda** \_: str(\_).title() + **' с опытом Python'**, arg))  
@print\_result  
**def** f4(arg):  
 **return** dict(zip(arg, gen\_random(len(arg), 100000, 200000)))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer1():  
 f4(f3(f2(f1(data))))

**Экранные формы с примерами выполнения программы:**













