**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-51Б |  | преподаватель каф. ИУ5 |
| Удодова Ксения |  | Гапанюк Юрий Евгеньевич |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2021 г.

**Описание задания:**

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.

**Задачи лабораторной работы:**

**Задача 1(файл field.py)**

***Условие:***

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 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.

***field.py***

*# Пример:  
# goods = [  
# {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
# {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}***def** field(items, \*args):  
 **assert** len(args) > 0  
 **if** len(args) == 1:  
 res = []  
 **for** item **in** items:  
 **if** args[0] **in** item.keys():  
 res.append(item[args[0]])  
 print(res)  
 **else**:  
 **for** item **in** items:  
 res = dict()  
 **for** key **in** args:  
 **if** key **in** item.keys():  
 res[key] = item[key]  
 print(res)  
  
goods = [  
 {**'title'**: **'Ковер'**, **'price'**: 2000, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'price'**: 5300, **'color'**: **'black'**}  
]  
  
field(goods, **'title'**)  
field(goods, **'title'**, **'price'**)

**Задача 2(файл gen\_random.py)**

***Условие:***

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.

***gen\_random.py***

**import** random  
**def** gen\_random(num\_count, first, last):  
 **for** number **in** range(num\_count):  
 **yield** random.randint(first, last)  
  
**for** i **in** gen\_random(5, 1, 3):  
 print(i)

**Задача 3(файл unique.py)**

***Условие:***

* 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.

***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.used\_elements = set()  
 self.data = items  
 self.index = 0  
  
 **if 'ignore\_case' not in** kwargs:  
 self.ignore\_case = **False  
 else**:  
 self.ignore\_case = kwargs[**'ignore\_case'**]  
  
 **def** \_\_iter\_\_(self):  
 **return** self  
  
 **def** \_\_next\_\_(self):  
 **while True**:  
 **if** self.index >= len(self.data):  
 **raise** StopIteration  
 **else**:  
 current = self.data[self.index]  
 self.index = self.index + 1  
 **if** self.ignore\_case:  
 **if** current.lower() **not in** self.used\_elements:  
 self.used\_elements.add(current.lower())  
 **return** current  
 **else**:  
 **if** current **not in** self.used\_elements:  
 self.used\_elements.add(current)  
 **return** current  
  
print(**'Пример 1: '**)  
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
**for** val **in** Unique(data):  
 print(val)  
print(**'Пример 2: '**)  
data = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
**for** val **in** Unique(data):  
 print(val)  
print(**'Пример 3: '**)  
data = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
**for** val **in** Unique(data, ignore\_case=**True**):  
 print(val)

**Задача 4(файл sort.py)**

***Условие:***

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.

Необходимо решить задачу двумя способами:

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

***sort.py***

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

**Задача 5(файл print\_result.py)**

***Условие:***

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.

***print\_result.py***

*# Здесь должна быть реализация декоратора***def** print\_result(func):  
 **def** wrapper(\*args):  
  
 out = func(\*args)  
 print(func.\_\_name\_\_)  
 **if** isinstance(out, list):  
 **for** val **in** out:  
 print(val)  
 **return** out  
 **elif** isinstance(out, dict):  
 **for** key, val **in** out.items():  
 print(**'{} = {}'**.format(key, val))  
 **return** out  
 **else**:  
 print(out)  
 **return** out  
  
 **return** wrapper  
  
  
@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()

**Задача 6(файл cm\_timer.py)**

***Условие:***

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

cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

***cm\_timer.py***

**import** time  
**from** contextlib **import** contextmanager  
  
  
**class** Cm\_timer\_1:  
  
 **def** \_\_init\_\_(self):  
 self.start\_time = **None** self.end\_time = **None  
  
 def** \_\_enter\_\_(self):  
 self.start\_time = time.time()  
  
 *# Должен возвращаться значимый объект  
 # например, открытый файл* **def** \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 self.end\_time = time.time()  
 print(**'time: {}'**.format(self.end\_time - self.start\_time))  
  
  
@contextmanager  
**def** cm\_timer\_2():  
 start\_time = time.time()  
 **yield** end\_time = time.time()  
 print(**'time: {}'**.format(end\_time - start\_time))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** Cm\_timer\_1():  
 time.sleep(5.0)  
  
 **with** cm\_timer\_2():  
 time.sleep(5.0)

**Задача 7(файл process\_data.py)**

***Условие:***

* В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
* В файле [data\_light.json](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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.

***process\_data.py***

**import** json  
**import** sys  
**import** cm\_timer  
**from** print\_result **import** print\_result  
**from** gen\_random **import** gen\_random  
*# Сделаем другие необходимые импорты*path = **'C:\\Users\\ksush\\data\_light.json'***# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария***with** open(path, encoding=**'utf8'**) **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):  
 **return** sorted(set([val.lower() **for** val **in** arg]), key=str.lower)  
  
  
@print\_result  
**def** f2(arg):  
 **return** list(filter(**lambda** x: str.startswith(x, **'программист'**), arg))  
  
  
@print\_result  
**def** f3(arg):  
 **return** list(map(**lambda** x: x + **' с опытом Python'**, arg))  
  
  
@print\_result  
**def** f4(arg):  
 temp = list(zip(arg, [(**', зарплата '**+str(el) + **' руб.'**) **for** el **in** list(gen\_random(len(arg), 100000, 200000))]))  
 **return** [(el[0]+el[1]) **for** el **in** temp]  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer.Cm\_timer\_1():  
 f4(f3(f2(f1([el[**'job-name'**] **for** el **in** data]))))

**Результат выполнения программы:**



