|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования 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)** |

**Факультет «ГУИМЦ»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Дисциплина «Базовые компоненты ИТ»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Функциональные возможности языка Python»

Студент: Близнева А.Е., группа ИУ5Ц-51Б

Преподаватель: Гапанюк Ю.Е.

2021г.

**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.

**Задача 1 (файл field.py)**

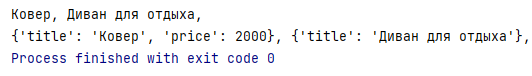
Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

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.

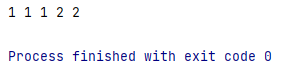
goods = [  
 {**'title'**: **'Ковер'**, **'price'**: 2000, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'color'**: **'black'**}  
]  
  
  
**def** field(items, \*args):  
 **assert** len(args) > 0  
 **if** len(args) == 1:  
 **for** i **in** items:  
 **yield** i.get(args[0])  
 **else**:  
 **for** i **in** items:  
 d = {}  
 **for** a **in** args:  
 **if** i.get(a) **is not None**:  
 d[a] = i.get(a)  
 **yield** d  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 n1 = field(goods, **'title'**)  
 **for** i **in** n1:  
 print(i, end=**', '**)  
 print()  
 n2 = field(goods, **'title'**, **'price'**)  
 **for** i **in** n2:  
 print(i, end=**', '**)



**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.

**import** random  
  
  
**def** gen\_random(num\_count, begin, end):  
 **for** i **in** range(num\_count):  
 **yield** random.randint(begin, end)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 print(\*gen\_random(5, 1, 3))



**Задача 3 (файл unique.py)**

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

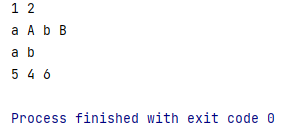
Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

При реализации необходимо использовать конструкцию \*\*kwargs.

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

Итератор не должен модифицировать возвращаемые значения.

**from** lab\_python\_fp.gen\_random **import** gen\_random  
  
  
**class** Unique(object):  
 **def** \_\_init\_\_(self, items, ignore\_case=**False**, \*\*kwargs):  
 self.set = set()  
 self.items = items  
 self.ignore\_case = ignore\_case  
 self.kwargs = kwargs  
  
 **def** \_\_next\_\_(self):  
 it = iter(self.items)  
 **while True**:  
 **try**:  
 current = next(it)  
 **except** StopIteration:  
 **raise  
 else**:  
 **if** self.ignore\_case == **True and** isinstance(current, str):  
 a = current[:]  
 **if** a.lower() **not in** self.set:  
 self.set.add(a.lower())  
 **return** current  
 **elif** current **not in** self.set:  
 self.set.add(current)  
 **return** current  
  
 **def** \_\_iter\_\_(self):  
 **return** self  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data1 = [**"a"**, **"A"**, **"b"**, **"B"**, **"a"**, **"A"**, **"b"**, **"B"**]  
 data3 = gen\_random(10, 4, 6)  
 print(\*Unique(data))  
 print(\*Unique(data1))  
 print(\*Unique(data1, **True**))  
 print(\*Unique(data3))



**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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.

2. Без использования lambda-функции.

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

**Задача 5 (файл print\_result.py)**

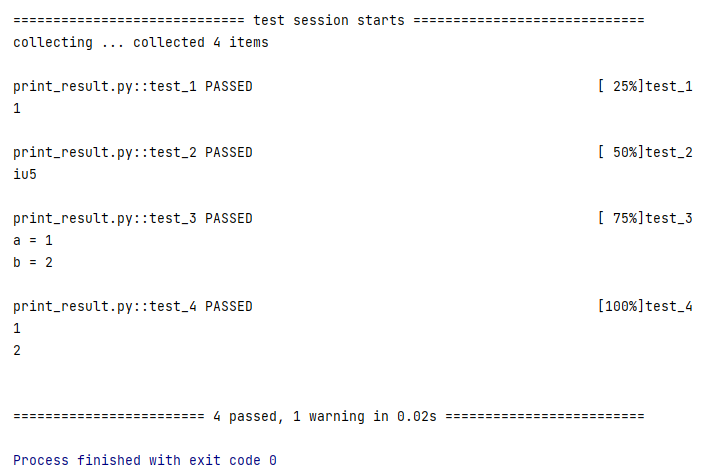
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

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.

**def** print\_result(func\_to\_decorate):  
  
 **def** decorated\_func(\*args):  
 print(func\_to\_decorate.\_\_name\_\_)  
 result = func\_to\_decorate(\*args)  
 **if** type(result) **is** list:  
 **for** i **in** result:  
 print(i)  
 **elif** type(result) **is** dict:  
 **for** i **in** result:  
 print(i, result.get(i), sep=**' = '**)  
 **else**:  
 print(result)  
  
 **return** result  
  
 **return** decorated\_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]  
  
  
**def** main():  
 print(**'!!!!!!!!'**)  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()



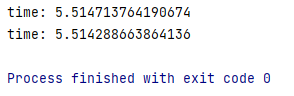
**Задача 6 (файл cm\_timer.py)**

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

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 (на основе класса и с использованием библиотеки contextlib).

**from** contextlib **import** contextmanager  
**import** time  
  
  
**class** cm\_timer\_1:  
 **def** \_\_init\_\_(self):  
 self.start\_time = **None  
  
 def** \_\_enter\_\_(self):  
 self.start\_time = time.time()  
  
 **def** \_\_exit\_\_(self, a, b, c):  
 print(**f'time: {**time.time() - self.start\_time**}'**)  
  
  
@contextmanager  
**def** cm\_timer\_2():  
 start\_time = time.time()  
 **yield** print(**f'time: {**time.time() - start\_time**}'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer\_1():  
 time.sleep(5.5)  
  
 **with** cm\_timer\_2():  
 time.sleep(5.5)



**Задача 7 (файл process\_data.py)**

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.

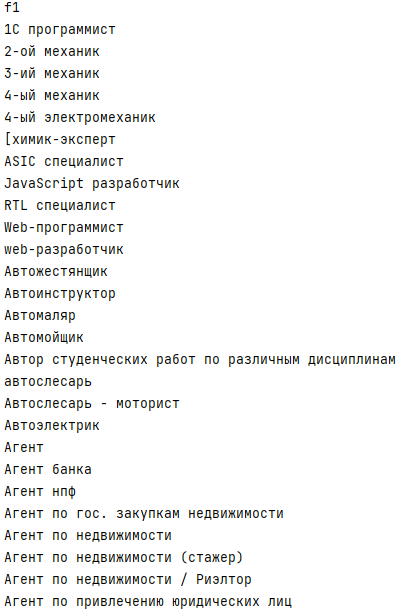
Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

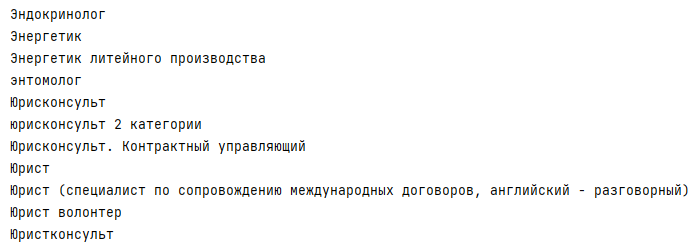
Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.

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.

**import** json  
**from** lab\_python\_fp.print\_result **import** print\_result  
**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.unique **import** Unique  
  
  
path = **'data\_light.json'  
with** open(path, encoding=**"utf-8"**) **as** f:  
 data = json.load(f)  
  
  
@print\_result  
**def** f1(arg):  
 **return** sorted(list(Unique(list(field(arg, **"job-name"**)), **True**)), key=**lambda** x: x.lower())  
  
  
@print\_result  
**def** f2(arg):  
 **return** list(filter(**lambda** x: x[:11].lower() == **'программист'**, arg))  
  
  
@print\_result  
**def** f3(arg):  
 **return** list(map(**lambda** x: x+**' с опытом Python'**, arg))  
  
  
@print\_result  
**def** f4(arg):  
 salary = list(gen\_random(len(arg), 100000, 200000))  
 **return** [**'{}, зарплата {} руб.'**.format(job, salary) **for** job, salary **in** zip(arg, salary)]  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer\_1():  
 f4(f3(f2(f1(data))))



. . .

