**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»

Отчёт по лабораторной 3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Шимолина Полина  Кирилловна |  | Гапанюк Юрий Евгеньевич |
|  |  |  |
| Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  |  |

Москва, 2021 г.

**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'

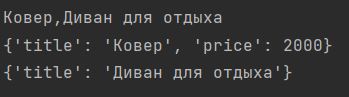
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

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

field.py

def field(items, \*args):  
 assert len(args)>0  
 if len(args)==1:  
 for i in items:  
 try:  
 if i[args[0]] is not None:  
 yield i[args[0]]  
 except:  
 pass  
 else:  
 for i in items:  
 n={}  
 for j in args:  
 try:  
 if i[j] is not None:  
 n.update({j:i[j]})  
 except:  
 pass  
 if not len(n)==0:  
 yield n  
  
if \_\_name\_\_=='\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]  
 print(','.join(field(goods, 'title')))  
 for n in field(goods,'title','price'):  
 print(n)  
 pass

**Результат:**

****

**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

gen\_random.py

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))

**Результат:**

****

**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’]

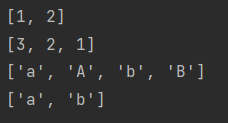
Unique(data) будет последовательно возвращать только a, A, b, B.

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

unique.py

from gen\_random import gen\_random  
  
  
class Unique(object):  
  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = set()  
 self.data = items  
 assert len(kwargs) < 2  
 if len(kwargs) == 0:  
 self.ignore\_case = False  
 else:  
 try:  
 self.ignore\_case = kwargs['ignore\_case']  
 except KeyError as k:  
 print('Неверное имя параметра итератора: ожидалось {}'.format(k))  
 raise  
  
 def \_\_next\_\_(self):  
 iterator = iter(self.data)  
 while True:  
 try:  
 current = next(iterator)  
 if self.ignore\_case and isinstance(current, str):  
 check = current[:].lower()  
 if check not in self.used\_elements:  
 self.used\_elements.add(check)  
 return current  
 elif current not in self.used\_elements:  
 self.used\_elements.add(current)  
 return current  
 except StopIteration:  
 raise  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 print([i for i in Unique(data)])  
  
 data = list(gen\_random(10, 1, 3))  
 print([i for i in Unique(data)])  
  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print([i for i in Unique(data)])  
 print([i for i in Unique(data, ignore\_case=True)])

**Результат:**

****

**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.

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 d: abs(d), reverse = True)  
 print(result\_with\_lambda)

**Результат:**

****

**Задача 5 (файл print\_result.py)**

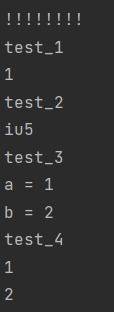
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

print\_result.py

def print\_result(func):  
 def d\_f(\*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
 result = func(\*args, \*\*kwargs)  
 #проверяем является ли результат листом  
 if isinstance(result, list):   
 for i in result:  
 print(i)  
 #проверяем является ли результат словарем  
 elif isinstance(result, dict):   
 for kw, arg in result.items():  
 print('{} = {}'.format(kw, arg))  
 else:  
 print(result)  
 return result  
 return d\_f

**Результат:**

****

**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)

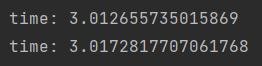
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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

cm\_timer.py

from time import time, sleep, perf\_counter  
from contextlib import contextmanager  
  
#на основе класса  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.start = 0  
   
 def \_\_enter\_\_(self):  
 self.start = time()  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 print('time: {}'.format(time() - self.start))  
   
#с использованием библиотеки contextlib   
@contextmanager  
def cm\_timer\_2():  
 start = time()  
 yield ' '  
 print('time: {}'.format(time() - start))  
   
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 sleep(3)  
 with cm\_timer\_2():  
 sleep(3)

**Результат:**

****

**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](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  
from field import field  
from gen\_random import gen\_random  
from unique import Unique  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
  
path = r'C:\Users\Полина\Desktop\f o l d e r\учеба\2 курс\бкит\lab 3\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")), ignore\_case=True)), key=lambda a: a.lower())  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda j: j[:11].lower() == "программист", arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda a: '{} с опытом Python'.format(a), list(arg)))  
  
  
@print\_result  
def f4(arg):  
 sal = [i for i in gen\_random(len(arg), 100000, 200000)]  
 return ['{}, зарплата: {} рублей'.format(job, sal) for job, sal in zip(arg, sal)]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Результат:**

