**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)**

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнил:

студент группы ИУ5-51Б Деркачев А.В.

Преподаватель:

Гапанюк Ю.Е

2020

1. Общее 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.

1. Описание, текст программы и экранные формы с примерами выполнения программы по каждой задаче
2. **Задача 1.**

Задание:

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.

Текст программы:

from typing import List *# необходим для задания типа для аннотации типов*def field(items: List[dict], \*args: str) -> iter:  
 *"""Создает генератор для отбора записей в списке словарей  
 items по ключам, указанным в args."""* assert len(args) > 0  
 if len(args) == 1:  
 for i in items:  
 if args[0] in i.keys() and not i[args[0]] is None:  
 yield i[args[0]]  
 else:  
 for i in items:  
 temp\_dict = {}  
 for key in args:  
 if key in i.keys() and not i[key] is None:  
 temp\_dict[key] = i[key]  
 if len(temp\_dict) > 0:  
 yield temp\_dict  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 goods = [  
 {**'title'**: **'Шапка'**, **'price'**: 1500, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'color'**: **'black'**},  
 {**'color'**:**'black'**}  
 ]  
 print(str(list(field(goods, **'title'**)))[1:-1])  
 print(str(list(field(goods, **'title'**, **'price'**)))[1:-1])

Экранные формы:

1. **Задача 2.**

Задание:

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

Текст программы:

import random  
  
def gen\_random(num\_count: int, begin: int, end: int) -> iter:  
 *"""Генерирует num\_count случайных чисел от begin до end, включая их."""* for i in range(num\_count):  
 yield random.randint(begin, end)  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(str(list(gen\_random(5, 1, 4)))[1:-1])

Экранные формы:

1. **Задача 3.**

Задание:

* 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.

Текст программы:

import random  
  
def gen\_random(num\_count: int, begin: int, end: int) -> iter:  
 *"""Генерирует num\_count случайных чисел от begin до end, включая их."""* for i in range(num\_count):  
 yield random.randint(begin, end)  
  
class Unique:  
 *"""Возвращает итератор, который принимает на вход массив или  
 генератор items и итерируется по элементам, пропуская дубликаты.  
 Также может принимать булевый параметр ignore\_case. Если ignore\_case =  
 True, то регистр игнорируется (то есть 'a' и 'A' - одно и тоже,  
 иначе - регистр учитывается (то есть 'a' и 'A' - разные символы.  
 По умолчанию ignore\_case = False."""* def \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = set() *# применяется для хранения уникальных элементов items* self.data = list(items)  
 self.index = 0 *# применяется для отслеживания индекса итерируемого элемента в items* if **'ignore\_case'** in kwargs.keys():  
 self.ignore\_case = kwargs[**'ignore\_case'**]  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 if self.index >= len(self.data):  
 raise StopIteration  
 current = self.data[self.index]  
 self.index += 1  
 if ((self.ignore\_case or not isinstance(current, str))  
 and current not in self.used\_elements):  
 self.used\_elements.add(current)  
 return current  
 elif (not self.ignore\_case and isinstance(current, str)  
 and current.upper() not in self.used\_elements  
 and current.lower() not in self.used\_elements):  
 self.used\_elements.add(current.upper())  
 self.used\_elements.add(current.lower())  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 data\_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data\_rand = gen\_random(10, 3, 10) *# генерируется 10 случайных чисел от 3 до 10* data\_str = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
 print(**'Отбор уникальных чисел: '**, str(list(Unique(data\_int)))[1:-1])  
 print(**'Отбор уникальных случайных чисел: '**, str(list(Unique(data\_rand)))[1:-1])  
 print(**'Отбор уникальных строк без игнорирования регистра по умолчанию: '**, str(list(Unique(data\_str)))[1:-1])  
 print(**'Отбор уникальных строк с игнорированием регистра: '**, str(list(Unique(data\_str, ignore\_case=True)))[1:-1])  
 print(**'Отбор уникальных строк без игнорирования регистра: '**, str(list(Unique(data\_str, ignore\_case=False)))[1:-1])

Экранные формы:



1. **Задача 4.**

Задание:

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, который 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.

Текст программы:

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

Экранные формы:

1. **Задача 5.**

Задание:

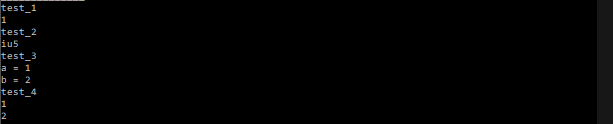
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

Текст программы:

def print\_result(func):  
 def decorated\_func(\*args): *# args - для 7-го пункта* print(func.\_\_name\_\_)  
 return\_value = func(\*args)  
 if isinstance(return\_value, list):  
 for value in return\_value:  
 print(str(value))  
 elif isinstance(return\_value, dict):  
 for key in return\_value.keys():  
 print(str(key) + **' = '** + str(return\_value[key]))  
 else:  
 print(return\_value)  
 return return\_value  
 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]  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**)  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

Экранные формы:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Задача 6.**

Задание:

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.

Текст программы:

**import** time  
**from** contextlib **import** contextmanager  
  
  
**class** cm\_timer\_1:  
 *"""Контекстный менеджер, который считает время работы блока кода и выводит его на экран,  
 основанный на классе"""* **def** \_\_enter\_\_(self):  
 self.start\_time = time.time()  
  
 **def** \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print(cm\_timer\_1.\_\_name\_\_, time.time() - self.start\_time)  
  
  
@contextmanager  
**def** cm\_timer\_2():  
 *"""Контекстный менеджер, который считает время работы блока кода и выводит его на экран,  
 основанный на библиотечной функции"""* start\_time = time.time()  
 **yield** print(cm\_timer\_2.\_\_name\_\_, time.time() - start\_time)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **with** cm\_timer\_1():  
 time.sleep(5.5)  
  
 **with** cm\_timer\_2():  
 time.sleep(5.5)

Экранные формы:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Задача 7.**

Задание:

* 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/iu5team/iu5web-fall-2020/tree/master/labs/lab3/data/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:

import json  
from lab35 import print\_result  
from lab33 import Unique  
from lab3 import field  
from lab32 import gen\_random  
from lab36 import cm\_timer\_1  
  
path = **'/Users/dankarpov/PycharmProjects/LAB3/data\_light.json'***#with open(path) as f:  
# data = json.load(f)*with open(path, encoding=**'utf-8'**) as f:  
 data = json.load(f)  
  
@print\_result  
def f1(arg):  
 return sorted(Unique(field(arg, **'job-name'**)))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: x.lower().startswith(**'программист'**), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + **' с опытом Python'**, arg))  
  
  
@print\_result  
def f4(arg):  
 gen\_salary = list(gen\_random(len(arg), 100000, 200000))  
 work\_and\_salary = list(zip(arg, gen\_salary))  
 return list(map(lambda x: x[0] + **', зарплата '** + str(x[1]) + **' руб'**, work\_and\_salary))  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Экранные формы:

