**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 «Информатика и системы управления»**

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

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:

студент группы ИУ5-33Б

Некрасов С. А.

Проверил:

Канев А.И.

2021 г.

### 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 раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

### Задача 1 (файл field.py)

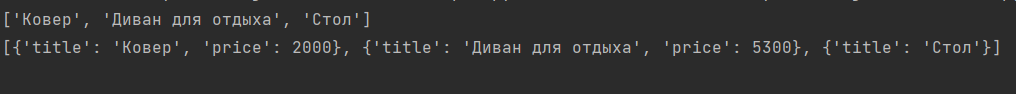
Необходимо реализовать генератор field. Генератор field последовательно выдаёт значения ключей словаря.

* 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.

**Текст программы:**

def field(items, \*args):  
 assert len(args) > 0 # если есть аргументы  
 if len(args) == 1: # если аргумент один  
 for vocabulary in items:  
 text = vocabulary.get(args[0]) # получаем значение ключа  
 if text is not None: # если значение не нулевое  
 yield text # yield возвращает генератор text  
 else: # если аргументов несколько  
 for j in items:  
 vocabulary = dict() # создали словарь с помощью dict  
 for key in args: # перебираем ключи аргументов  
 text = j.get(key) # получаем значение ключа  
 if text is not None: # если значение не нулевое  
 vocabulary[key] = text # добавили значение к ключу  
 if len(vocabulary) != 0: # если vocabulary имеет хотя бы одно значение  
 yield vocabulary # yield возвращает генератор vocabulary  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': None},  
 {'title': 'Стол', 'price': None, 'color': 'white'},  
 {'title': None, 'price': None, 'color': 'pink'}  
 ]  
 data1 = list() # объявили пременную как список  
 data2 = list() # объявили пременную как список  
  
 for i in field(goods, 'title'):  
 data1.append(i)  
 print(data1)  
  
 for i in field(goods, 'title', 'price'):  
 data2.append(i)  
 print(data2)

**Результат выполнения:**



### 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.

**Текст программы:**

import random  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield random.randint(begin, end)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(list(gen\_random(6, 1, 5)))

**Результат выполнения:**



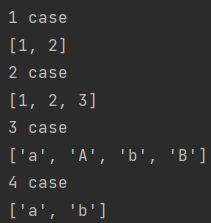
### Задача 3 (файл unique.py)

* 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.

**Текст программы:**

from gen\_random import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs): # kwargs для переменной длины аргументов  
 self.data = iter(items) # метод iter() используется для получения итератора  
 self.word = set() # создание пустого множества  
 if kwargs: # пока есть элементы  
 self.app = kwargs['ignore\_case']  
 else: # если элементы закончатся  
 self.app = False  
  
 def \_\_iter\_\_(self): # обязательно надо вернуть объект итератора  
 return self  
  
 def \_\_next\_\_(self): # вернуть следующий элемент в последовательности  
 while True:  
 x = next(self.data) # перебираются элементы  
 if self.app == True and type(x) != int: # если это буквенные символы  
 x = x.lower() # делает все элементы нижнего регистра  
 if x not in self.word:  
 self.word.add(x) # добавляет элемент если его ещё не было  
 return x  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data2 = gen\_random(10, 1, 3)  
 data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 data4 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 print('1 case')  
 print(list(Unique(data1)))  
 print('2 case')  
 print(list(Unique(data2)))  
 print('3 case')  
 print(list(Unique(data3, ignore\_case=False))) # игнорируем регистр  
 print('4 case')  
 print(list(Unique(data4, ignore\_case=True))) # не игнорируем регистр

**Результат выполнения:**



**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.

**Текст программы:**

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

**Результат выполнения:**



**Задача 5 (файл print\_result.py)**

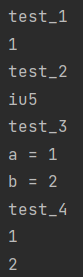
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

**Текст программы:**

def print\_result(func\_to\_decorate):  
 def decorated\_func(\*args, \*\*kwargs):  
 incoming = func\_to\_decorate(\*args, \*\*kwargs)  
 print(func\_to\_decorate.\_\_name\_\_)  
 if isinstance(incoming, list):  
 # Функцией isinstance() можно проверить класс, кортеж с классами, либо рекурсивный кортеж кортежей.  
 for i in incoming:  
 print(i)  
 elif isinstance(incoming, dict):  
 for i in incoming:  
 print(str(i) + ", " + str(incoming[i]))  
 else:  
 print(incoming)  
 return incoming  
  
 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\_\_':  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

**Результат выполнения:**



### 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.

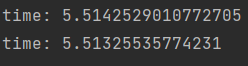
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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).

**Текст программы:**

import time  
from contextlib import contextmanager  
  
  
class cm\_timer1: # реализации на основе класса  
  
 def \_\_init\_\_(self):  
 self.startTime = time.time()  
  
 def \_\_enter\_\_(self): # этот метод создает и возвращает объект связи базы данных  
 self.startTime = time.time()  
  
 def \_\_exit\_\_(self, type, value, traceback): # этот метод закрывает файл  
 if type is not None:  
 print(type, value, traceback)  
 else:  
 print("time: {}".format(time.time() - self.startTime))  
  
  
@contextmanager  
def cm\_timer2(): # реализация с использованием библиотеки contextlib  
 startTime = time.time()  
 yield  
 print("time: {}".format(time.time() - startTime))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer1(): # with вызывает метод \_\_enter\_\_  
 time.sleep(5.5)  
 with cm\_timer2():  
 time.sleep(5.5)

**Результат выполнения:**



### 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.

**Текст программы:**

import json  
from print\_result import print\_result  
from cm\_timer import cm\_timer1  
from unique import Unique  
from field import field  
from gen\_random import gen\_random  
  
# Сделаем другие необходимые импорты  
  
path = r'C:\Users\79508\Downloads\data\_light.json'  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
with open(path, encoding='utf-8') 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(list(Unique(field(arg, 'job-name'), ignore\_case=True)), key=str.lower)  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda string: str.startswith(str.lower(string), 'программист'), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda s: s + " с опытом python", arg))  
  
  
@print\_result  
def f4(arg):  
 return dict(zip(arg, list('зарплата {} руб.'.format(val) for val in gen\_random(len(arg), 1000000, 2000000))))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer1():  
 f4(f3(f2(f1(data))))

**Результат выполнения:**

f1 представлена не полностью, так как встречается много неповторяющихся профессий.

