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

«Основные конструкции языка Python»

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:

студент группы ИУ5-35Б

Ищенко А.С.

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 выдает 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.

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

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.

)

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.

Шаблон реализации:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    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.

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.

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).

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 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:

field.py

```
import sys
import math

def field(name, *args):
    if len(args) == 1:
        if type(name) == str:
            name = eval(name)
        for name in name:
            if name.get(args[0]) and name[args[0]] is not None:
                    yield name[args[0]]
    else:
        try:
            assert len(args) > 0
            if type(name) == str:
                name = eval(name)
            for name in name:
                dict = {arg: name[arg] for arg in args if name.get(arg)
and name[arg] is not None}
                if any(dict):
                    yield dict
        except AssertionError:
            print('\nВведите названия пунктов!')


    # Необходимо реализовать генератор

# def main():
#     name = input("Введите название библиотеки: ")
#     while not name in globals():
#         name = input("Такой библиотки не сущевует!\nВведите название
библиотеки: ")
#     args = []
#
#     cur = input("Введите названия пункта: ")
#     while cur:
#         args.append(cur)
#         cur = input("Введите названия пункта: ")
#     result = field(name, *args)
#     print(list(result))
#
#
# if __name__ == "__main__":
#     main()
```

genRandom.py

```
import random

def genRandom(num, begin, end):
    for i in range(num):
```

```
        yield(random.randint(begin, end))


# def main():
#     num, begin, end = (int(i) for i in input().split())
#     gen = genRandom(num, begin, end)
#     for i in range(num):
#         print(next(gen), end=' ')
#
# if __name__ == "__main__":
#     main()
```

unique.py

```
from genRandom import genRandom

class Unique(object):
    index = 0
    mas = []
    def __init__(self, items, **kwargs):
        ignore_case = kwargs.get('ignore_case')
        cur = list(items)
        for i in cur:
            if ignore_case and type(i) == str:
                if not(i.lower() in [x.lower() for x in self.mas]):
                    self.mas.append(i)
            else:
                if not(i in self.mas):
                    self.mas.append(i)
        # 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, Абв и АБВ - разные строки
        #           ignore_case = False, Абв и АБВ - одинаковые строки,
одна из которых удалится
        # По-умолчанию ignore_case = False


    def __next__(self):
        array_length = len(self.mas)
        prev_index = self.index

        if self.index < array_length:
            self.index += 1

        if prev_index <= array_length and prev_index < array_length:
            return self.mas[prev_index]
        else:
            self.index = 0
            raise StopIteration

    def __iter__(self):
        return self
```

```
def main():
    data = ['A', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    data1 = Unique(data, ignore_case=True)

    for i in data1:
        print(i)

if __name__ == "__main__":
    main()
```

sort.py

```
def sort(list):
    return sorted(list, reverse=True, key=abs)


data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

# if __name__ == '__main__':
#     result = sort(data)
#     print(result)
#
#     result_with_lambda = sorted(data, key=lambda x: abs(x)*-1)
#     print(result_with_lambda)
```

printResult.py

```
def print_result(func):
    def wrapper(*kwargs):
        print(func.__name__)
        result = func(*kwargs)
        if type(result) is list:
            for i in result:
                print(i)
        elif type(result) is dict:
            for key in result:
                print('{} = {}'.format(key, result[key]))
        else:
            print(result)
        return result
    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()
```

cmTimer.py

```
from contextlib import contextmanager
import time

class cm_timer_1:
    def __enter__(self):
        self.time1 = time.time()
    def __exit__(self, *args):
        print('1 таймер: ', time.time() - self.time1)

@contextmanager
def cm_timer_2():
    startTime= 0
    try:
        startTime = time.time()
        yield startTime
    finally:
        print('2 таймер: {:.5f}'.format(time.time() - startTime))

# if __name__ == '__main__':
#     with cm_timer_1():
#         time.sleep(1.5)
#     with cm_timer_2():
#         time.sleep(1.5)
```

process_data.py

```
import json
import sys
from field import field
```

```python
from sort import sort
from unique import Unique
from genRandom import genRandom
from printResult import print_result
from cmTimer import cm_timer_1
# Сделаем другие необходимые импорты

path = "/Users/sergejadolevic/PycharmProjects/laba3/data_light.json"


# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path, "r", 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(list(sorted([element for element in Unique(field(arg, 'job-name'), ignore_case=True)])))

@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith("Программист"), arg))

@print_result
def f3(arg):
    return list(map(lambda x: x + ' со знанием Python', arg))

@print_result
def f4(arg):
    pay = list(genRandom(len(arg), 100000, 200000))
    strings = [' зарплата {} рублей'.format(i) for i in pay]
    return list(zip(arg, strings))


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Результат выполнения программы:

(представлены только функции f2, f3, f4, т.к. результат выполнения функции f1занимает более 40 страниц документа)

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
f3
Программист со знанием Python
Программист / Senior Developer со знанием Python
Программист 1С со знанием Python
Программист C# со знанием Python
Программист C++ со знанием Python
Программист C++/C#/Java со знанием Python
Программист/ Junior Developer со знанием Python
Программист/ технический специалист со знанием Python
Программистр-разработчик  информационных систем со знанием Python
f4
('Программист со знанием Python', ' зарплата 105430 рублей')
('Программист / Senior Developer со знанием Python', ' зарплата 108594 рублей')
('Программист 1С со знанием Python', ' зарплата 179342 рублей')
('Программист C# со знанием Python', ' зарплата 151785 рублей')
('Программист C++ со знанием Python', ' зарплата 117684 рублей')
('Программист C++/C#/Java со знанием Python', ' зарплата 119367 рублей')
('Программист/ Junior Developer со знанием Python', ' зарплата 143524 рублей')
('Программист/ технический специалист со знанием Python', ' зарплата 176428 рублей')
('Программистр-разработчик  информационных систем со знанием Python', ' зарплата 157374 рублей')
1 таймер:  2.818068265914917

Process finished with exit code 0
```