Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №2
«Функциональные возможности языка Python»

Выполнил:
Студент группы РТ5-31Б
Серик И. Н.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2023 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. Рабочая часть некоторых файлов закомментирована в связи с дальнейшим использованием их функций в других задачах.

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.

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

Шаблон для реализации генератора:

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
```

```
# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

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(10, 1, 3)

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.

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:

С использованием lambda-функции.

Без использования lambda-функции.

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. 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).

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.

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

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

field.py

```
def field(dicts, *args):
    result = []
    for i in dicts:
        if len(args) == 1:
            if i[args[0]]:
                result.append(i[args[0]])
        else:
            tmp = dict()
            for j in args:
                if i[j]:
                    tmp[j] = i[j]
            result.append(tmp)
    print(result)

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    field(goods, 'title') # 'Ковер', 'Диван для отдыха'

     field(goods, 'title', 'price') # [{'title': 'Ковер', 'price': 2000}, {'title':
                        'Диван для отдыха', 'price': 5300}]
```

get_random.py

```
import random

def gen_random(count, nmin, nmax):
    for i in range(count):
        yield random.randint(nmin, nmax)

'''print("5 random numbers from 1 to 10:")
for n in gen_random(5, 1, 10):
    print(n, end =" ")
print()'''
```

unique.py

```
from get_random import gen_random

class Unique:
    def __init__(self, items, **kwargs):
        self.items = iter(items)
```

```
        self.ignore_case = kwargs.get('ignore_case', False)
        self.seen = set()

    def __next__(self):
        while True:
            item = next(self.items)
            key = item.lower() if self.ignore_case and isinstance(item, str) else
item
            if key not in self.seen:
                self.seen.add(key)
                return item

    def __iter__(self):
        return self

if __name__ == '__main__':
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    unique_data = Unique(data)
    for item in unique_data:
        print(item, end=' ')
    print()

    data = gen_random(10, 1, 3)
    uniquesRow = Unique(data)
    for i in uniquesRow:
        print(i, end=' ')
    print()

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    unique_data = Unique(data)
    for item in unique_data:
        print(item, end=' ')

    print()
```

sort.py

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

if __name__ == '__main__':
    #С использованием lambda-функции
    result = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result)
    #Без использования lambda-функции
    result = sorted(data, key=abs, reverse=True)

    print(result)
```

print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs)
        print("Функция: ", func.__name__)
        if isinstance(res, list):
            for i in res:
                print(i)
        elif isinstance(res, dict):
            for key in res:
                print(f"{key} = {res[key]}")
        else:
            print(res)
        return res
    return wrapper

@print_result
```

```python
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()
```

cm_timer.py

```python
from contextlib import contextmanager
import time

class cm_timer_1:
    def __enter__(self):
        self.start = time.time()
        return self

    def __exit__(self, *args):
        elapsed = time.time() - self.start
        print(f"Прошло: {elapsed}")

#with cm_timer_1():
    #time.sleep(5.5)


@contextmanager
def cm_timer_2():
    start = time.time()
    yield
    elapsed = time.time() - start
    print(f"Прошло: {elapsed}")

#with cm_timer_2():
      #time.sleep(5.5)
```

process_data.py

```python
import json
from cm_timer import cm_timer_1
from print_result import print_result
from get_random import gen_random

path = "Lab3-4//data_light.json"
with open(path, encoding='utf-8') as f:
    data = json.load(f)

@print_result
def f1(data):
    return sorted(set(job['job-name'].lower() for job in data))

@print_result
def f2(data):
```

```
    return list(filter(lambda x: x.startswith('программист'), data))

@print_result
def f3(data):
    return list(map(lambda x: x + ', с опытом Python', data))

@print_result
def f4(data):
    salaries = gen_random(len(data), 100000, 200001)
    return [f'{job}, зарплата {salary} руб.' for job, salary in zip(data, salaries)]

if __name__ == '__main__':
    with cm_timer_1():

        f4(f3(f2(f1(data))))
```

Результаты

field.py

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
```

get_random.py

```
5 random numbers from 1 to 10:
1 10 1 6 5
```

unique.py

Для `'ignore_case' == False:`

```
1 2
3 2 1
a A b B
```

Для `'ignore_case' == True:`

```
1 2
2 3 1
a b
```

sort.py

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

print_result.py

```
Функция:  test_1
1
Функция:  test_2
iu5
Функция:  test_3
a = 1
b = 2
Функция:  test_4
1
2
```

cm_timer.py

```
Прошло: 5.500536680221558
Прошло: 5.500228404998779
```

process_data.py

```
Функция:  f1
1с программист
2-ой механик
3-ий механик
...
юрист
юрист (специалист по сопровождению международных договоров, английский -
разговорный)
юрист волонтер
юристконсульт
Функция:  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
Функция:  f3
программист, с опытом Python
программист / senior developer, с опытом Python
программист 1с, с опытом Python
программист c#, с опытом Python
программист c++, с опытом Python
программист c++/c#/java, с опытом Python
программист/ junior developer, с опытом Python
программист/ технический специалист, с опытом Python
программистр-разработчик  информационных систем, с опытом Python
Функция:  f4
программист, с опытом Python, зарплата 132019 руб.
программист / senior developer, с опытом Python, зарплата 126513 руб.
программист 1с, с опытом Python, зарплата 159221 руб.
программист c#, с опытом Python, зарплата 167592 руб.
программист c++, с опытом Python, зарплата 171795 руб.
программист c++/c#/java, с опытом Python, зарплата 108532 руб.
программист/ junior developer, с опытом Python, зарплата 126151 руб.
программист/ технический специалист, с опытом Python, зарплата 185084 руб.
программистр-разработчик  информационных систем, с опытом Python, зарплата 159178
руб.
Прошло: 0.07515287399291992
```