Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнил:
студент группы РТ5-51Б
Сысоев Александр

Проверил:
доцент каф. ИУ5
Гапанюк Ю. Е.

Москва, 2021 г.

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.

ЗАДАНИЕ 1

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.

Исходный код

Файл fields.py:

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'colour': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'colour': 'black'}
]


def field(items, *args):
    try:
        assert len(items) != 0 and len(args) != 0
    except:
        print("Вы не передали аргументы")
    return_list = []
    for item in items:
        temp_dict = {}
        if len(args) == 1:
            for key, value in item.items():
                if key in args and value != None:
                    return_list.append(value)
        else:
            for key, value in item.items():
                if key in args and value != None:
                    temp_dict[key] = value
            if len(temp_dict) > 0:
                return_list.append(temp_dict)

    return return_list


def main():
    print(str(field(goods, 'title'))[1:-1])
    print(str(field(goods, 'title', 'price'))[1:-1])
    print(str(field(goods, 'title', 'price', 'colour'))[1:-1])


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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/field.py
'Ковер', 'Диван для отдыха'
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
{'title': 'Ковер', 'price': 2000, 'colour': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'colour': 'black'}

Process finished with exit code 0
```

ЗАДАНИЕ 2

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

Исходный код

Файл gen_random.py:

```
from random import randint

def gen_random(count, begin, end):
    try:
        assert count > 0
    except:
        print("Вы не указали кол-во чисел")
        pass

    result_list = [randint(begin, end) for index in range(count)]
    return result_list

def main():
    print(str(gen_random(5, 1, 3))[1:-1])
    print(str(gen_random(2, 0, 10))[1:-1])
    print(str(gen_random(10, 0, 99))[1:-1])

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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/gen_random.py
2, 1, 3, 1, 1
6, 3
32, 84, 78, 29, 10, 43, 65, 71, 81, 80

Process finished with exit code 0
```

ЗАДАНИЕ 3

- 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.

Исходный код

Файл unique.py:

```
from random import randint
from gen_random import  gen_random

class Unique:
    def __init__(self, items, **kwargs):
        self.index = 0
        self.data = list(items)
        self.unique_list = set()

        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 == True:
                if current not in self.unique_list:
```

```
                    if type(current) == str:
                        self.unique_list.add(current)
                        return current
                    else:
                        self.unique_list.add(current)
                        return current
            elif self.ignore_case == False:
                if type(current) == str:
                    if current.upper() not in self.unique_list and
current.lower() not in self.unique_list:
                        self.unique_list.add(current.lower())
                        self.unique_list.add(current.upper())
                        return current
                else:
                    if current not in self.unique_list:
                        self.unique_list.add(current)
                        return current

    def __iter__(self):
        return self

data_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

def main():
    print(str(list(Unique(data_int)))[1:-1])
    print(str(list(Unique(data_str, ignore_case = True)))[1:-1])
    print(str(list(Unique(data_str, ignore_case = False)))[1:-1])
    print(str(list(Unique(gen_random(100, 1, 5))))[1:-1])

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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/unique.py
1, 2
'a', 'A', 'b', 'B'
'a', 'b'
1, 3, 4, 2, 5

Process finished with exit code 0
```

ЗАДАНИЕ 4

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.

Исходный код

Файл sort.py:

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

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

def sort_with_lambda(data):
    return sorted(data, key=lambda value: value if value > 0 else value*-1,
reverse=True)

def main():
    print(str(sort(data))[1:-1])
    print(str(sort_with_lambda(data))[1:-1])

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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/sort.py
123, 100, -100, -30, 30, 4, -4, 1, -1, 0
123, 100, -100, -30, 30, 4, -4, 1, -1, 0

Process finished with exit code 0
```

ЗАДАНИЕ 5

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.

Исходный код

Файл print_result.py:

```python
def print_result(func):
    def decorated_func(*args):
        print(func.__name__)
        return_value = func(*args)
        if type(return_value) == list:
            for value in return_value:
                print (value)
        elif type(return_value) == dict:
            for key, value in return_value.items():
                print(f'{key} = {value}')
        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()
```

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
```

ЗАДАНИЕ 6

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).

Исходный код

Файл cm_timer.py:

```
import time
from contextlib import contextmanager

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(cm_timer_1.__name__, time.time() - self.start)

@contextmanager
def cm_timer_2():
    start = time.time()
    yield
    print(cm_timer_2.__name__, time.time() - start)

def main():
    with cm_timer_2():
        time.sleep(5.5)
    with cm_timer_1():
        time.sleep(5.5)


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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/cm_timer.py
cm_timer_2 5.506127834320068
cm_timer_1 5.5031609535217285

Process finished with exit code 0
```

ЗАДАНИЕ 7

- 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.

Исходный код

Файл process_data.py:

```
import json

from cm_timer import cm_timer_1
from field import field
from gen_random import gen_random
from print_result import print_result
from unique import Unique


def process_data():
    with open('data_light.json', encoding='utf-8') as file:
        data = json.load(file)

    @print_result
    def f1(value):
        return sorted(Unique(field(value, 'job-name')))

    @print_result
    def f2(value):
        return list(filter(lambda x: x.lower().startswith('программист'),
value))

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

    @print_result
    def f4(value):
        salary = list(gen_random(len(value), 100000, 200000))
        return list(map(lambda x: x[0] + ', зарплата ' + str(x[1]) + ' руб',
list(zip(value, salary))))

    with cm_timer_1():
        f4(f3(f2(f1(data))))


def main():
    process_data()

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

Результат работы программы

```
C:\Users\AlexGood\PycharmProjects\DZ\venv\Scripts\python.exe G:/Users/alexg/PycharmProjects/Lab3/process_data.py
f1
1C программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
```

```
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 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
f4
Программист с опытом Python, зарплата 112083 руб
Программист / Senior Developer с опытом Python, зарплата 168097 руб
Программист 1С с опытом Python, зарплата 122113 руб
Программист C# с опытом Python, зарплата 133877 руб
Программист C++ с опытом Python, зарплата 169897 руб
Программист C++/C#/Java с опытом Python, зарплата 129117 руб
Программист/ Junior Developer с опытом Python, зарплата 169839 руб
Программист/ технический специалист с опытом Python, зарплата 157567 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 112001 руб
cm_timer_1 0.03200697898864746
```