Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3
«Функциональные возможности языка Python»

Выполнил:
Студент группы ИУ5-33Б
Лупарев Сергей
Подпись и дата:

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

2023 год

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.

Задача 1 (файл field.py)

Описание задачи

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.

Текст программы

```python
def field(items, *args):
    if len(args) == 1:
        res = []
        for i in items:
            if i[args[0]]:
                res.append(i[args[0]])
        return res

    res = []
    for i in items:
        temp = {}
        for a in args:
            if i[a]:
                temp[a] = i[a]
        if temp:
            res.append(temp)
    return tuple(res)


def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    print(*field(goods, 'title'), sep=', ')
    print(*field(goods, 'title', 'price'), sep=', ')


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

Примеры выполнения программы

```
========= RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\field.py =========
Ковер, Диван для отдыха
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
>>>
```

Задача 2 (файл gen_random.py)

Описание задачи

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

```python
from random import randint

def gen_random(num_count, begin, end):
    res = []
    for i in range(num_count):
        res.append(randint(begin, end))
    return res

def main():
    print(*gen_random(5, 1, 3), sep=', ')

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

Примеры выполнения программы

```
======= RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\gen_random.py =======
1, 1, 3, 1, 2
>>>
```

Задача 3 (файл unique.py)

Описание задачи

- 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.

Текст программы

```
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.res = set()
        self.mass = items
        self.i = 0

        if 'ignore_case' not in kwargs:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs['ignore_case']

    def __next__(self):
        if self.ignore_case:
            for i, el in enumerate(self.mass):
                if isinstance(el, str):
                    self.mass[i] = el.lower()

        while True:
            if self.i >= len(self.mass):
                raise StopIteration
            else:
                t = self.mass[self.i]
                self.i += 1
                if t not in self.res:
                    self.res.add(t)
                    return t

    def __iter__(self):
        return self
```

```python
def main():
    print('ЗАДАНИЕ 1')
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data):
        print(item)

    print('ЗАДАНИЕ 2')
    data = gen_random(10, 1, 3)
    for item in Unique(data):
        print(item)

    print('ЗАДАНИЕ 3')
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data):
        print(item)

    print('ЗАДАНИЕ 4')
    for item in Unique(data, ignore_case=True):
        print(item)


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

Примеры выполнения программы

```
========= RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\unique.py =========
ЗАДАНИЕ 1
1
2
ЗАДАНИЕ 2
3
2
1
ЗАДАНИЕ 3
a
A
b
B
ЗАДАНИЕ 4
a
b
>>>
```

Задача 4 (файл sort.py)

Описание задачи

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.

Текст программы

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

def main():
    result = sorted(data, key=abs, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)


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

Примеры выполнения программы

```
========== RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\sort.py ==========
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
>>>
```

Задача 5 (файл print_result.py)

Описание задачи

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

```python
def print_result(func):
    def f(*args, **kwargs):
        print(func.__name__)
        res = func(*args, **kwargs)
        if isinstance(res, list):
            for i in res:
                print(i)
            return res
        elif isinstance(res, dict):
            for key, value in res.items():
                print('{} = {}'.format(key, value))
            return res
        else:
            print(res)
            return res

    return f


@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]


def main():
    test_1()
    test_2()
    test_3()
    test_4()


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

Примеры выполнения программы

```
====== RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\print_result.py ======
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Задача 6 (файл cm_timer.py)

Описание задачи

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).

Текст программы

```
import time
from contextlib import contextmanager


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

    def __exit__(self, exc_type, exc_value, exc_tb):
        res = time.time() - self.start_time
        print('Время выполнения: {}'.format(res))


@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    end_time = time.time()
    res = end_time - start_time
    print('Время выполнения: {}'.format(res))


def main():
    with cm_timer_1():
        time.sleep(1.5)

    with cm_timer_2():
        time.sleep(1.5)


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

Примеры выполнения программы

```
======== RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\cm_timer.py ========
Время выполнения: 1.5093994140625
Время выполнения: 1.5145704746246338
>>>
```

Задача 7 (файл process_data.py)

Описание задачи

- 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

```python
import json
import io
import codecs


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

'''
with open("data_light.json") as f:
    data = json.load(f)
'''

file = codecs.open("data_light.json", "r", "utf-8")
data = json.load(file)
file.close()

@print_result
def f1(arg):
    return sorted(list(Unique([item['job-name'] for item in arg], ignore_case = True)))



@print_result
def f2(arg):
    return list(filter(lambda text: (text.split())[0] == 'программист', arg))


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


@print_result
def f4(arg):
    tmp = list(zip(arg, [', зарплата ' + str(salary) + ' руб.' for salary in gen_random(len(arg), 100000, 200000)]))
    return [item[0] + item[1] for item in tmp]


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

Примеры выполнения программы

```
====== RESTART: D:\CBeer\Labs_PICKUP\lab3\lab_python_fp\process_data.py ======
f1
1c программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
агент нпф
```

```
энтомолог
юрисконсульт
юрисконсульт 2 категории
юрисконсульт. контрактный управляющий
юрист
юрист (специалист по сопровождению международных договоров, английский - разговорный)
юрист волонтер
юристконсульт
f2
программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
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
f4
программист с опытом Python, зарплата 194524 руб.
программист / senior developer с опытом Python, зарплата 156094 руб.
программист 1с с опытом Python, зарплата 183299 руб.
программист c# с опытом Python, зарплата 197419 руб.
программист c++ с опытом Python, зарплата 155244 руб.
программист c++/c#/java с опытом Python, зарплата 164282 руб.
Время выполнения: 19.495301246643066
```