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
Грызин Алексей
Подпись и дата:

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

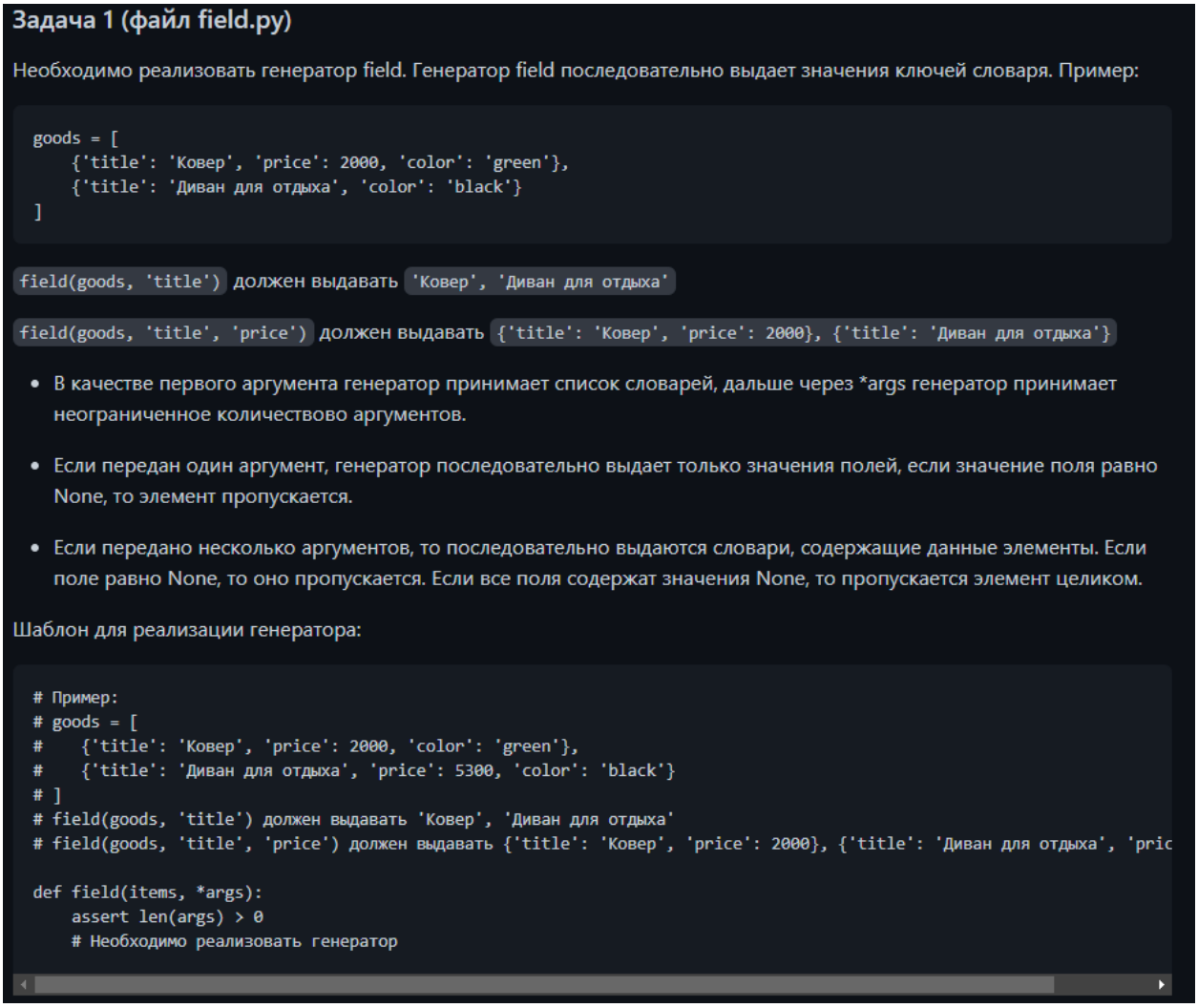
Описание задания

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

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. 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.

Шаблон для реализации генератора:

```
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, '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', 'pric

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

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(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.

Шаблон для реализации класса-итератора:

Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # 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
        pass

    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

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:

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.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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 выполнения:

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

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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

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

with open(path) 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):
    raise NotImplemented


@print_result
def f2(arg):
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
    raise NotImplemented


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

Тексты программ

Задача 1: field.py

```python
from typing import Dict, List


def field(dicts: List[Dict], *args: List[str]) -> List:
    assert len(args) > 0 and len(dicts) > 0

    return_value = []

    for dict in dicts:
        buf_dict = {}
        for key in dict.keys():
            if len(args) == 1:
                if key in args:
```

```
                    return_value.append(dict[key])
            else:
                if key in args:
                    buf_dict[key] = dict[key]
        if len(args) > 1: return_value.append(buf_dict)

    return return_value

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black', 'material': 'К
ожа'}
]
data_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
data_sort = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

def test_field():
    print('Task 1-1: ', str(field(goods, 'title'))[1:-1])
    print('Task 1-2: ', str(field(goods, 'title', 'price'))[1:-1])
    print('Task 1-3: ', str(field(goods, 'title', 'price', 'material'))[1:-1])
    print('------------------------------------------------------')

def main():
    test_field()

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

Результат

```
Task 1-1:  'Ковер', 'Диван для отдыха'
Task 1-2:  {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
Task 1-3:  {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300, 'material': 'Кожа'}
------------------------------------------------------
```

Задача 2: gen_random.py

```
from typing import List
from random import randint


def gen_random(num_count, begin, end) -> List:
    assert num_count >= 0
    return [randint(begin, end) for i in range(num_count)]

def test_gen_random():
    print('Task 2-1: ', str(gen_random(5, 1, 3))[1:-1])
    print('Task 2-2: ', str(gen_random(3, 1, 100))[1:-1])
    print('Task 2-3: ', str(gen_random(5, -100, 100))[1:-1])
    print('------------------------------------------------------')

def main():
```

```
    test_gen_random()

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

Результат

```
Task 2-1:  2, 2, 2, 3, 2
Task 2-2:  91, 27, 63
Task 2-3:  -50, 42, 40, 63, -42
----------------------------------------------------
```

Задача 3: unique.py

```
# Итератор для удаления дубликатов
from gen_random import gen_random
class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.data = list(items)
        self.index = 0

        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 or not isinstance(current, str)) and current no
t in self.used_elements):
                self.used_elements.add(current)
                return current

            elif (not self.ignore_case and isinstance(current, str) and current.u
pper() not in self.used_elements
                    and current.lower() not in self.used_elements):
                self.used_elements.add(current.upper())
                self.used_elements.add(current.lower())
                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 test_unique():
    print('Task 3-1: ', str(list(Unique(data_int)))[1:-1])
    print('Task 3-
2 ignoring case: ', str(list(Unique(data_str, ignore_case = True)))[1:-1])
    print('Task 3-
2 not ignoring case: ', str(list(Unique(data_str, ignore_case = False)))[1:-1])
    print('Task 3-3: ', str(list(Unique(gen_random(100, 1, 5))))[1:-1])
    print('-----------------------------------------------------')

def main():
    test_unique()

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

Результат

```
Task 3-1:  1, 2
Task 3-2 ignoring case:  'a', 'A', 'b', 'B'
Task 3-2 not ignoring case:  'a', 'b'
Task 3-3:  2, 4, 1, 3, 5
-----------------------------------------------------
```

Задача 4: sort.py

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

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

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

def test_sort():
    print('Task 4 without lambda: ', str(sort(data_sort))[1:-1])
    print('Task 4 with lambda: ', str(sort_lambda(data_sort))[1:-1])
    print('-----------------------------------------------------')

def main():
    test_sort()

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

Результат

```
Task 4 without lambda:  123, 100, -100, -30, 30, 4, -4, 1, -1, 0
Task 4 with lambda:  123, 100, -100, -30, 30, 4, -4, 1, -1, 0
-------------------------------------------------------
```

Задача 5: print_result.py

```
def print_result(func):
    def decorated_func(*args):
        print(func.__name__)
        return_value = func(*args)
        if isinstance(return_value, list):
            for value in return_value:
                print(str(value))
        elif isinstance(return_value, dict):
            for key in return_value.keys():
                print(str(key) + ' = ' + str(return_value[key]))
        else:
            print(return_value)
        return return_value
    return decorated_func

def test_print_result():
    @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]

    print('Task 5:\n')
    test_1()
    test_2()
    test_3()
    test_4()
    print('-------------------------------------------------------')

def main():
    test_print_result()
```

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

Результат

```
Task 5:

test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
------------------------------------------------------------
```

Задача 6: cm_timer.py

```
import time
from contextlib import contextmanager


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

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


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

def test_timer():
    print('Task 6:\n')
    with cm_timer_1():
        time.sleep(5.5)
    with cm_timer_2():
        time.sleep(5.5)
    print('------------------------------------------------------------')


def main():
    test_timer()
```

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

Результат

```
Task 6:

cm_timer_1 5.500380992889404
cm_timer_2 5.500359058380127
----------------------------------------------------------
```

Задача 7: 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

path = 'lab3/json/data_light.json'

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

    @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 test_process_data():
```

```
    print('Task 7:\n')
    process_data()

def main():
    test_process_data()

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

Результат

```
юрисконсульт 2 категории
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 125615 руб
Программист / Senior Developer с опытом Python, зарплата 122770 руб
Программист 1С с опытом Python, зарплата 178611 руб
Программист C# с опытом Python, зарплата 103412 руб
Программист C++ с опытом Python, зарплата 124447 руб
Программист C++/C#/Java с опытом Python, зарплата 167967 руб
Программист/ Junior Developer с опытом Python, зарплата 111203 руб
Программист/ технический специалист с опытом Python, зарплата 196659 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 184995 руб
cm_timer_1 0.4834930896759033
```