

Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»



Гапанюк Ю.Е.

Отчет по Лабораторной работе №4
По курсу
“Разработка интернет-приложений”

Выполнил:
Постникова М.А.
Студент группы ИУ5-54

Задание и порядок выполнения

Задача 1 (ex_1.py)

Необходимо реализовать генераторы `field` и `gen_random`

Генератор `field` последовательно выдает значения ключей словарей массива

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'}`

1. В качестве первого аргумента генератор принимает `list` , дальше через `*args` генератор принимает неограниченное кол-во аргументов.

2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно `None` , то элемент пропускается

3. Если передано несколько аргументов, то последовательно выдаются словари, если поле равно `None` , то оно

пропускается, если все поля `None` , то пропускается целиком весь элемент

Генератор `gen_random` последовательно выдает заданное количество случайных чисел в заданном диапазоне

Пример:

`gen_random(1, 3, 5)` должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1

В `ex_1.py` нужно вывести на экран то, что они выдают *одной строкой*

Генераторы должны располагаться в `librip/ gen.py`

Задача 2 (ex_2.py)

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной `bool`-параметр `ignore_case` , в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен `False` . 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(gen_random(1, 3, 10))` будет последовательно возвращать только 1 , 2 и 3

```
data = ['a', 'A', 'b', 'B']
```

`Unique(data)` будет последовательно возвращать только `a` , `A` , `b` , `B`

```
data = ['a', 'A', 'b', 'B']
```

`Unique(data, ignore_case=True)` будет последовательно возвращать только `a` , `b`

В `ex_2.py` нужно вывести на экран то, что они выдают *одной строкой*. **Важно** продемонстрировать работу как массивами, так и с генераторами (`gen_random`).

Итератор должен располагаться в `librip/ iterators .py`

Задача 3 (ex_3.py)

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции `sorted`.

Пример:

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

Вывод: [0, 1, -1, 4, -4, -30, 100, -100, 123]

Задача 4 (ex_4.py)

Необходимо реализовать декоратор `print_result` , который выводит на экран результат выполнения функции.

Файл `ex_4.py` **не нужно** изменять.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение.

Если функция вернула список (`list`), то значения должны выводиться в столбик.

Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через знак равно.

Пример:

```
@print_result
def test_1():
    return 1
@print_result
def test_2():
    return 'iu'
@print_result
def test_3():
    return {'a': 1, 'b': 2}
@print_result
def test_4():
    return [1, 2]
test_1()
test_2()
test_3()
test_4()
```

На консоль выведется:

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

Декоратор должен располагаться в `librip/ decorators .py`

Задача 5 (ex_5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

Пример:

```
with timer():
```

```
    sleep(5.5)
```

После завершения блока должно вывестись в консоль примерно 5.5

Задача 6 (ex_6.py)

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл `data_light.json`. Он содержит облегченный список вакансий в России в формате `json` (ссылку на полную версию размером ~ 1 Гб. в формате `xml` можно найти в файле `README.md`).

Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

В `ex_6.py` дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора `@print_result` печатается результат, а контекстный менеджер `timer` выводит время работы цепочки функций.

Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне.

Функции `f1-f3` должны быть реализованы в 1 строку, функция `f4` может состоять максимум из 3 строк.

Что функции должны делать:

1. Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна **игнорировать регистр**. Используйте наработки из предыдущих заданий.

2. Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием. Для фильтрации используйте функцию `filter`.

3. 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`.

4. 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

ex_1.py

```
from librip.gens import field, gen_random
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

field_1 = field(goods, 'title')
print(', '.join(map(str, field_1)))

field_2 = field(goods, 'title', 'price')
print(', '.join(map(str, field_2)))
```

```
gen_1 = gen_random(1,3,5)
print(', '.join(map(str,gen_1)))
```

ex_2.py

```
from librip.gens import gen_random
from librip.iterators import Unique
data = [1,1,1,2,2,2,2,2]
my_order = Unique(data)
for i in my_order:
    print(i, end = ' ')
print()

data2 = gen_random(1,3,10)
my_order = Unique(data2)
for i in my_order:
    print(i, end = ' ')
print()

data3 = ['a','A','b','B','a']
my_order = Unique(data3) # ignore_case = False
for i in my_order:
    print(i, end = ' ')
print()

data4 = ['a','A','b','B']
my_order = Unique(data4, True) # ignore_case = True
for i in my_order:
    print(i, end = ' ')
print()
```

ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
print(sorted(data, key=lambda number: abs(number)))
```

ex_4.py

```
from librip.decorators import print_result

# Необходимо верно реализовать print_result
# и задание будет выполнено

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
```

```
def test_4():
    return [1, 2]
```

```
test_1()
test_2()
test_3()
test_4()
```

ex_5.py

```
from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(5.5)
```

ex_6.py

```
import json
import sys
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

path = 'data_light.json'

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

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

# Далее необходимо реализовать все функции по заданию, заменив `raise
# NotImplemented`
# Важно!
# Функции с 1 по 3 должны быть реализованы в одну строку
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов

@print_result
def f1(arg):
    return sorted(unique(field(arg, "job-name")))

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

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

@print_result
def f4(arg):
    empList = list(zip(arg, list(gen_random(100000, 200000, len(arg)))))
    return list(map(lambda x: x[0] + ", зарплата " + str(x[1]) + " руб",
```

```
empList))

with timer():
    f4(f3(f2(f1(data))))
```

Результаты работы программы

```
Run ex_1
C:\Python34\python.exe C:/Users/student/Documents/lr4/ex_1.py
Ковер, Диван для отдыха, Стелаж, Вешалка для одежды
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стелаж', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}
1, 3, 2, 1, 2
Process finished with exit code 0
```

```
Run ex_2
C:\Python34\python.exe C:/Users/student/Documents/lr4/ex_2.py
1 2
1 2 3
a A b B
a b
Process finished with exit code 0
```

```
Run ex_3
C:\Python34\python.exe C:/Users/student/Documents/lr4/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
Process finished with exit code 0
```

```
Run ex_4
C:\Python34\python.exe C:/Users/student/Documents/lr4/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

4: Run 6: TODO Terminal Python Console

```
Run ex_5
C:\Python34\python.exe C:/Users/student/Documents/lr4/ex_5.py
5.500314950942993

Process finished with exit code 0
```

4: Run 6: TODO Terminal Python Console

f2
Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
программист
программист 1C
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1C 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
Программист-разработчик информационных систем с опытом Python
программист с опытом Python
программист 1C с опытом Python
f4
Программист с опытом Python, зарплата 194623 руб
Программист / Senior Developer с опытом Python, зарплата 128079 руб
Программист 1C с опытом Python, зарплата 162625 руб
Программист C# с опытом Python, зарплата 177682 руб
Программист C++ с опытом Python, зарплата 139736 руб
Программист C++/C#/Java с опытом Python, зарплата 108524 руб
Программист/ Junior Developer с опытом Python, зарплата 118219 руб
Программист/ технический специалист с опытом Python, зарплата 125093 руб
Программист-разработчик информационных систем с опытом Python, зарплата 155509 руб
программист с опытом Python, зарплата 133635 руб
программист 1C с опытом Python, зарплата 185603 руб
0.4580259323120117

Process finished with exit code 0