Московский государственный технический университет им. Н.Э.Баумана

Демонстрация ЛР:
Гапанюк. Ю.Е.

Отчет по лабораторной работе № 4 по курсу РИП

"Python. Функциональные возможности"

ИСПОЛНИТЕЛЬ:

студент группы **ИУ5-54**

Наседкин И.А.

Москва, МГТУ - 201**6**

Задание ЛР

Важно выполнять все задачи последовательно . С 1 по 5 задачу формируется модуль librip , с помощью
которого будет выполняться задание 6 на реальных данных из жизни. Весь вывод на экран (даже в столбик)
необходимо реализовывать одной строкой .

Подготовительный этап

1. Зайти на github.com и выполнить fork проекта с заготовленной структурой https://github.com/iu5team/ex-lab4
2. Переименовать репозиторий в lab_4
3. Выполнить git clone проекта из вашего репозитория

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 *о дной строкой*. **Важно** продемонстрировать работу как
с 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 выводить 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). П ример: *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.__

Текст программы
ex_1.py

```
#!/usr/bin/env python3

from librip.gens import field
from librip.gens import gen_random

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

```
    {'title': 'Стеллаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

# Реализация задания 1
print(list(field(goods, 'title')), end=',')
print(list(field(goods, 'title', 'price')), end=',')
#list(gen_random(1, 3, 5))
```

gens.py

```
import random


# Генератор вычленения полей из массива словарей
# 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', 'price': 5300}

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
    for item in items:
        if len(args) ==1:
            if args[0] in item:
               yield item[args[0]]
        else:
            yield {arg: item[arg]
                   for arg in args
                        if arg in item}


# Генератор списка случайных чисел
# Пример:
# gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1
# Hint: реализация занимает 2 строки
def gen_random(begin, end, num_count):
    # Необходимо реализовать генератор
    for el in range(num_count):
        print(random.randint(begin, end))
```

ex_2.py

```
#!/usr/bin/env python3
from librip.gens import gen_random
from librip.iterators import Unique

# Реализация задания 2
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
test=Unique(data1)
for i in test:
    print(i, end=',')
data2 = gen_random(1,3,10)
test = Unique(data2)
for i in test:
      print(i, end=',')
```

iterators.py

```
# 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 одинаковые строки в
разном регистре
        # Например: ignore_case = True, Абв и АБВ разные строки
        #           ignore_case = False, Абв и АБВ одинаковые строки, одна из них
удалится
        # По-умолчанию ignore_case = False
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.uniq_items = list()
        self.index = 0

    def __next__(self):
        # Нужно реализовать __next__
        while self.items:
            value = next(self.items)
            if self.ignore_case and type(value) == type(str):
                value = str(value).lower()
            if value not in self.uniq_items:
                self.uniq_items.append(value)
                return value
    def __iter__(self):
        return self
```

ex_3.py

```
#!/usr/bin/env python3


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


# Реализация задания 3

import math
print(sorted(data, key = lambda item: abs(item)))
```

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()
```

decorators.py

```
# Здесь необходимо реализовать декоратор, print_result который принимает на вход функцию,
# вызывает её, печатает в консоль имя функции, печатает результат и возвращает значение
# Если функция вернула список (list), то значения должны выводиться в столбик
# Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак
равно
# Пример из ex_4.py:
# @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

def print_result(f):
    def decor(*args, **kwargs):
        myresult = f(*args, **kwargs)
        print(f.__name__)
        if type(myresult) == dict:
            for key in sorted(myresult):
                print(key+' = '+str(myresult[key]))
        elif type(myresult) == list:
            for item in myresult:
                print(item)
        else:
            print(myresult)
        return myresult
    return decor
```

ex_5.py

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

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

ctxmngrs.py

```
# Здесь необходимо реализовать
# контекстный менеджер timer
# Он не принимает аргументов, после выполнения блока он должен вывести время
выполнения в секундах
# Пример использования
# with timer():
#   sleep(5.5)
#
# После завершения блока должно вывестись в консоль примерно 5.5
import time

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time.time() - self.start_time)
        return False   # no errors, could be zero
```

ex_6.py

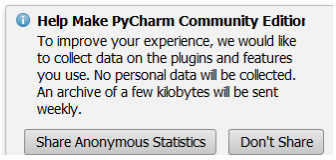
```
#!/usr/bin/env python3
import json
import sys
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

path = None

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске
path = sys.argv[1]

with open(path) 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):
    ''' Функция f1 должна вывести отсортированный список профессий
    без повторений (строки в разном регистре считать равными).
    Сортировка должна игнорировать регистр. Используйте наработки
    из предыдущих заданий.'''
    #raise NotImplemented
    return sorted(unique(field(arg,'job-name'), ignore_case=True))
```

```
@print_result
def f2(arg):
    ''' 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.'''
    #raise NotImplemented
    return list(filter(lambda x: str(x).startswith('программист'),arg))


@print_result
def f3(arg):
    ''' 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.'''
    #raise NotImplemented
    return list(map(lambda x: x+' с опытом Python',arg))


@print_result
def f4(arg):
    ''' 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.'''
    #raise NotImplemented
    salaries = gen_random(100000,200000,len(arg))
    return [job+', зарплата '+str(salary)+' руб.' for (job, salary) in
            zip(arg, salaries)]


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

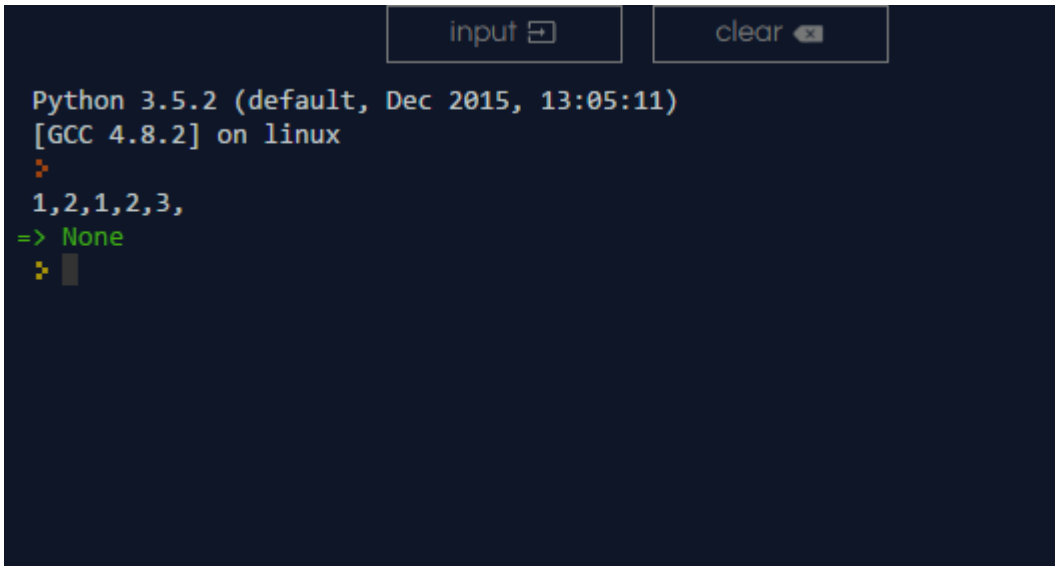
Результаты работы программы

1

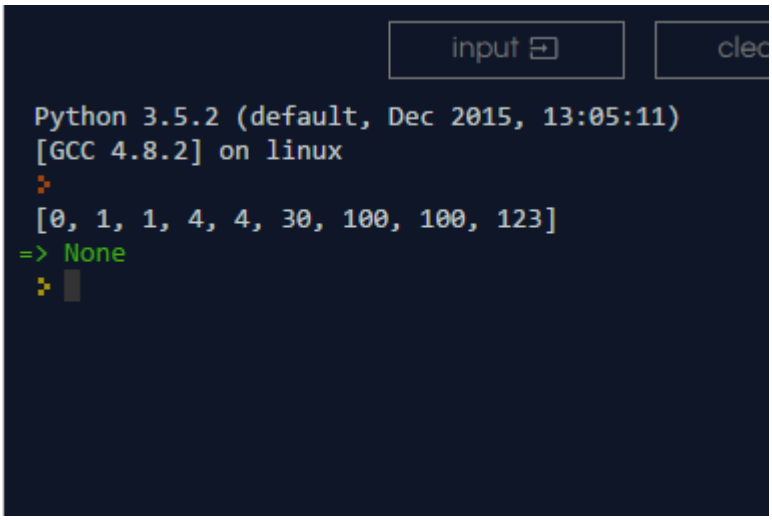
```
C:\Python34\python.exe C:/Users/student/PycharmProjects/untitled/ex_1.py
Ковер,Диван для отдыха,Стеллаж,Вешалка для одежды,{'price': 2000, 'title': 'Ковер'},{'price': 5300, 'title': 'Диван для отдыха'},{'price': 7000, 'title': 'Стеллаж'},{'price': 800, 'title':
2
1
3
1

Process finished with exit code 0
```



2



3



4

```
C:\Python34\python.exe "D:/5 семестр/РИП/ЛР №4/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
```

5

```
C:\Python34\python.exe "D:/5 семестр/РИП/ЛР №4/ex_5.py"
5.500314950942993

Process finished with exit code 0
```

6

```
C:\Python34\python.exe D:/5semester/RIP/LR4/ex_6.py D:\5semester\RIP\LR4\data_light_cp1251.json
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
Web-разработчик
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
```

```
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 142232 руб.
Программист / Senior Developer с опытом Python, зарплата 150862 руб.
Программист 1С с опытом Python, зарплата 164047 руб.
Программист C# с опытом Python, зарплата 109807 руб.
Программист C++ с опытом Python, зарплата 198596 руб.
Программист C++/C#/Java с опытом Python, зарплата 184886 руб.
Программист/ Junior Developer с опытом Python, зарплата 199585 руб.
Программист/ технический специалист с опытом Python, зарплата 122911 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 130701 руб.
0.08500504493713379
```