Московский государственный технический университет им. Н.Э.Баумана

Проверил:
Гапанюк Ю.Е.

"__"_____2019 г.

Отчет по лабораторной работе № 2 по курсу "Разработка интернет-приложений"

«Python. Функциональные возможности»

2
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы **ИУ5-54** _____
(подпись)

Меркулова Н. А.

"__"_____2019 г.

Москва, МГТУ - 2019

1. Задание

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 запрограммировать 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_2`
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, с помощью кода в *одну строку*
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 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). 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.

2. Исходный код

2.1. gens.py

```
import random


# Генератор вычленения полей из массива словарей
def field(items, *args):
  assert len(args) > 0

  if len(args) == 1:
    for item in items:
      yield (item.get(args[0]))
  else:
    for item in items:
      yield {arg: item.get(arg) for arg in args}


# Генератор списка случайных чисел
def gen_random(begin, end, num_count):
  assert num_count != 0

  if begin > end:
```

```
    begin, end = end, begin

  return (random.randint(begin, end) for _ in range(num_count))
```

2.2. decorators.py

```
from __future__ import print_function


def print_result(func_to_decorate):
  def decorated_func(*args):
    print('Function name: ' + func_to_decorate.__name__)

    print('Function results:', end='\n')
    t = func_to_decorate(*args)
    if type(t) is dict:
      for key in t.keys():
        print('{} = {}'.format(key, t[key]))
    else:
      if type(t) is list:
        for values in t:
          print(values)
      else:
        print(t)
    print()
    return t

  return decorated_func
```

2.3. ctxmngrs.py

```
import time


class timer:
  def __init__(self):
    super

  def __enter__(self):
    self.begin_time = time.time()
    pass

  def __exit__(self, exp_type, exp_value, traceback):
    print(time.time() - self.begin_time)
    pass
```

2.4. iterators.py

```
class Unique(object):
```

```
  def __init__(self, items, **kwargs):
    if type(items) is list:
      self.items = items
    else:
      self.generated_items = [i for i in items]
      self.items = self.generated_items

    self.limit = len(self.items)
    self.current = 0

    self.ignore_case = kwargs.get('ignore_case')

    if self.ignore_case:
      self.unique_dict = dict()
    else:
      self.unique = set()

  def __next__(self):
    if self.ignore_case:
      while self.items[self.current].lower() in self.unique_dict:
        if self.current < self.limit:
          self.current += 1

          if self.current >= self.limit:
            raise StopIteration
        else:
          break

      self.unique_dict.update({self.items[self.current].lower(): self.items[self.current]})
      return self.items[self.current]
    else:
      while self.items[self.current] in self.unique:
        if self.current < self.limit:
          self.current += 1

          if self.current >= self.limit:
            raise StopIteration
        else:
          break

      self.unique.add(self.items[self.current])
      return self.items[self.current]

  def __iter__(self):
    return self
```

2.5. ex_1.py

```
from __future__ import print_function

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

```python
# Тест генератора gen_random
def test_gen_random():
    print('TEST GENERATOR gen_random()', end='\n')

    print('Rand begin:', end=' ')
    begin = int(input())

    print('Rand end:', end=' ')
    end = int(input())

    print('Rand num count:', end=' ')
    num_count = int(input())

    print('Results:', end=' ')
    for i in gen_random(begin, end, num_count):
        print(i, end=' '),


# Тест генератора field
def test_field_with_goods():
    print('TEST GENERATOR field()', end='\n')

    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'}
    ]

    print('Available keys:', end=' ')
    for key in goods[0].keys():
        print(key, end=' ')

    print('(enter some of these arguments or none)', end='\n')

    arguments = str(input())
    arguments = arguments.split(' ')

    for a in arguments:
        if a not in goods[0].keys():
            print('No such key(-s)')
            return

    if len(arguments) == 0:
        for g in field(goods):
            print (g, end=' ')
    elif len(arguments) == 1:
        for g in field(goods, arguments[0]):
            print (g, end=' ')
    elif len(arguments) == 2:
        for g in field(goods, arguments[0], arguments[1]):
            print (g, end=' ')
    else:
```

```
    for g in field(goods, arguments[0], arguments[1], arguments[2]):
        print (g, end=' ')


if __name__ == "__main__":
    test_gen_random()

    print('\n')

    test_field_with_goods()
```

2.6. ex_2.py

```
from __future__ import print_function

from librip.gens import gen_random
from librip.iterators import Unique


def test_unique_iterator():
    print('TEST WITH LIST OF INTS')
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(data1)
    for i in Unique(data1, ignore_case=False):
        print(i, end=' ')
    print('\n')

    print('TEST WITH GENERATOR OF INTS\ngen_random(1, 3, 10)')
    data2 = gen_random(1, 3, 10)
    for i in Unique(data2, ignore_case=False):
        print(i, end=' ')
    print('\n')

    print('TEST WITH LIST OF STR (ignore_case=False)')
    data3 = ['a', 'A', 'b', 'B']
    print(data3)
    for i in Unique(data3, ignore_case=False):
        print(i, end=' ')
    print('\n')

    print('TEST WITH LIST OF STR (ignore_case=True)')
    print(data3)
    for i in Unique(data3, ignore_case=True):
        print(i, end=' ')


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

2.7. ex_3.py

```
def main():
```

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

    print(list(sorted(data, key=lambda num: abs(num))))


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

2.8. ex_4.py

```
# coding=utf-8
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()
```

2.9. ex_5.py

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

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

2.10. ex_6.py

```
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

with open(path) as f:
  data = json.load(f)


@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


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

3. Скриншоты с результатами выполнения

3.1. ex_1.py

3.2. ex_2.py

```
TEST WITH LIST OF INTS
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2

TEST WITH GENERATOR OF INTS
gen_random(1, 3, 10)
2 1 3

TEST WITH LIST OF STR (ignore_case=False)
['a', 'A', 'b', 'B']
a A b B

TEST WITH LIST OF STR (ignore_case=True)
['a', 'A', 'b', 'B']
a b
Process finished with exit code 0
```

3.3. ex_3.py

```
TEST DATA TO SORT
[4, -30, 100, -100, 123, 1, 0, -1, -4]
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

3.4. ex_4.py

```
Function name: test_1
Function results:
1

Function name: test_2
Function results:
iu

Function name: test_3
Function results:
a = 1
b = 2

Function name: test_4
Function results:
1
2


Process finished with exit code 0
```

3.5. ex_5.py

```
5.50187397003

Process finished with exit code 0
```

3.6. ex_6.py

```
Function name: f3
Function results:
Системный программист (C, Linux) с опытом Python
Веб-программист с опытом Python
1С программист с опытом Python
Инженер-программист ККТ с опытом Python
инженер - программист с опытом Python
Инженер-программист (Клинский филиал) с опытом Python
Инженер-программист (Орехово-Зуевский филиал) с опытом Python
Ведущий программист с опытом Python
Инженер - программист АСУ ТП с опытом Python
инженер-программист с опытом Python
Инженер-электронщик (программист АСУ ТП) с опытом Python
Старший программист с опытом Python
Web-программист с опытом Python
Веб - программист (PHP, JS) / Web разработчик с опытом Python
Инженер-программист 1 категории с опытом Python
Ведущий инженер-программист с опытом Python
Инженер-программист САПОУ (java) с опытом Python
Помощник веб-программиста с опытом Python
педагог программист с опытом Python
Инженер-программист ПЛИС с опытом Python

Function name: f4
Function results:
Системный программист (C, Linux) с опытом Python, зарплата 126066 руб.
Веб-программист с опытом Python, зарплата 162651 руб.
1С программист с опытом Python, зарплата 107653 руб.
Инженер-программист ККТ с опытом Python, зарплата 140383 руб.
инженер - программист с опытом Python, зарплата 103619 руб.
Инженер-программист (Клинский филиал) с опытом Python, зарплата 162415 руб.
Инженер-программист (Орехово-Зуевский филиал) с опытом Python, зарплата 135236 руб.
Ведущий программист с опытом Python, зарплата 163162 руб.
Инженер - программист АСУ ТП с опытом Python, зарплата 127627 руб.
инженер-программист с опытом Python, зарплата 170654 руб.
Инженер-электронщик (программист АСУ ТП) с опытом Python, зарплата 104530 руб.
Старший программист с опытом Python, зарплата 180334 руб.
Web-программист с опытом Python, зарплата 146751 руб.
Веб - программист (PHP, JS) / Web разработчик с опытом Python, зарплата 151226 руб.
Инженер-программист 1 категории с опытом Python, зарплата 151735 руб.
Ведущий инженер-программист с опытом Python, зарплата 106761 руб.
Инженер-программист САПОУ (java) с опытом Python, зарплата 106351 руб.
Помощник веб-программиста с опытом Python, зарплата 197767 руб.
педагог программист с опытом Python, зарплата 193360 руб.
Инженер-программист ПЛИС с опытом Python, зарплата 105753 руб.

0.024458885192871094
```