Московский государственный технический

университет им. Н.Э. Баумана.

Факультет «Системы обработки информации и управления»

Отчет по лабораторной работе №3 по курсу

«Разработка Интернет-Приложений»

Тема: «Python. Функциональные возможности»

Выполнил:

студент группы ИУ5-53

Коционова А.А.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю.Е.

Подпись и дата:

Москва, 2018 г.

Задание:

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_3`
3. Выполнить `git clone` проекта из вашего репозитория

Задания, исходный код и результаты:

Задача 1 (`ex_1.py`)

Необходимо реализовать генераторы `field` и `gen_random`

Генератор `field` последовательно выдает значения ключей словарей массива

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

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`

gens.py

```python
def field(arr, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
    for el in arr:  # где el - словарь
        slovar = {}
        for arg in args:
            if (arg in el.keys()) and (len(args) == 1):
                yield el[arg]  # генератор выдает только
                #  значения полей
            elif arg in el is not None:
                slovar[arg] = el[arg]  # формируем новый словарь,
                # где пропускаем элементы равные None
        if len(slovar) > 0 and len(args) > 1:
            yield slovar


def gen_random(begin, end, num_count):
    # Необходимо реализовать генератор
    for i in range(num_count):
        yield random.randint(begin, end)
```

ex_1.py

```python
from librip.gens import *

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'}
```

```
]

# Реализация задания 1

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))

print(list(gen_random(1, 3, 5)))
```

РЕЗУЛЬТАТ П.1

```
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'ti
[1, 2, 1, 1, 1]
```

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.
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`

iterators.py

```
class Unique(object):
    IGNORE_CASE = False
    INDEX = 0
    OBJECTS = []
    PUSTOTA = []

    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, то Абв и АБВ разные строки
        #           ignore_case = False, то Абв и АБВ одинаковые строки,
одна из них удалится
        # По-умолчанию ignore_case = False(то есть регистры важны)
        if 'ignore_case' in kwargs.keys():
            self.IGNORE_CASE = kwargs['ignore_case']
        if type(items == GeneratorType):
            self.OBJECTS = list(items)
        else:
            self.OBJECTS = items
        # self.ITEMS = len(items)

    def __next__(self):
        while True:  # чтобы избавиться от None
            if self.INDEX == (len(self.OBJECTS) - 1):
                raise StopIteration
            self.INDEX += 1

            val = self.OBJECTS[self.INDEX]
            val2 = str(val).lower()
```

```
            if self.IGNORE_CASE:
                val = val2
            if val not in self.PUSTOTA:
                self.PUSTOTA.append(val)
                return val

    def __iter__(self):
        del self.PUSTOTA[:]
        self.INDEX = -1
        return self
```

ex_2.py

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

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)
data3 = ['a', 'A', 'b', 'B']

# Реализация задания 2
print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(data3)))
print(list(Unique(data3, ignore_case=True)))
```

РЕЗУЛЬТАТЫ П.2

```
[1, 2]
[3, 1, 2]
['a', 'A', 'b', 'B']
['a', 'b']
```

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`

ex_3.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
print(sorted(data, key=lambda x: abs(x)))
```

РЕЗУЛЬТАТЫ П.2

```
[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
Декоратор должен располагаться в `librip/decorators.py`

decorators.py

```
# РЕАЛИЗАЦИЯ
def print_result(func):
    def decorated_func(*args):
        if len(args) == 0:
            result = func()
        else:
            result = func(args[0])
        print(func.__name__)
        if type(result) == list:
            for i in result:
                print(i)
        elif type(result) == dict:
            for key in result:
                print(str(key) + " = " + str(result[key]))
        else:
            print(result)
        return result

    return decorated_func
```

ex_4.py

```
from librip.decorators import print_result


# Необходимо верно реализовать print_result
# и задание будет выполнено

@print_result  # test_1=print_result(test_1)
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()
```

РЕЗУЛЬТАТЫ П.4

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

Задача 5 (`ex_5.py`)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран
После завершения блока должно вывестись в консоль примерно 5.5

ctxmngrs.py

```python
import time


# 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
class timer:
    def __enter__(self):
        self.start = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        ti = (time.time()) - self.start
        print(ti)
```

ex_5.py

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

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

РЕЗУЛЬТАТЫ П.5

```
C:\Users\kotsi\lab
5.511963367462158
```

Задача 6 (`ex_6.py`)

В репозитории находится файл `data_light.json`. Он содержит облегченный список вакансий в России в формате `json`.
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.

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

path = "data_light_cp1251.json"

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

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 дожны 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 list(Unique(list(field(arg, "job-name")), ignore_case=True))


@print_result
def f2(arg):
    return list(filter(lambda s: "программист" in s[0:12], arg))


@print_result
def f3(arg):  # map(func, arr)
    return list(map(lambda s: s + " с опытом Python", arg))


@print_result
def f4(arg):
    Sal = gen_random(100000, 200000, len(arg))
    return list(map(lambda s: '{}, зарплата {} руб.'.format(
        s[0], s[1]), zip(arg, Sal)))
```

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

РЕЗУЛЬТАТЫ П.6.1-6.4

```
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор - тренер
помощник руководителя
автоэлектрик
врач ультразвуковой диагностики в детскую поликлинику
менеджер по продажам ит услуг (b2b)
менеджер по персоналу
аналитик
воспитатель группы продленного дня
инженер по качеству
инженер по качеству 2 категории (класса)
водитель автомобиля
пекарь
переводчик
терапевт
врач-анестезиолог-реаниматолог
инженер-конструктор в наружной рекламе
монтажник-сборщик рекламных конструкций
оператор фрезерно-гравировального станка
зоотехник
```

```
f2
программист
программист c++/c#/java
программист 1с
программистр-разработчик  информационных систем
программист c++
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
f3
программист с опытом Python
программист c++/c#/java с опытом Python
программист 1с с опытом Python
программистр-разработчик  информационных систем с опытом Python
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист c# с опытом Python
f4
программист с опытом Python, зарплата 110193 руб.
программист c++/c#/java с опытом Python, зарплата 107110 руб.
программист 1с с опытом Python, зарплата 162842 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 193506 руб.
программист c++ с опытом Python, зарплата 196003 руб.
программист/ junior developer с опытом Python, зарплата 108468 руб.
программист / senior developer с опытом Python, зарплата 115235 руб.
программист/ технический специалист с опытом Python, зарплата 182457 руб.
```