Отчёт по лабораторной работе №3 по курсу «Разработка интернет приложений»

Тема: «Python. Функциональные возможности»

Выполнил: студент группы ИУ5-53
Кондратьев Максим Владимирович

Дата: 23/09/2018 Подпись:_____

Проверил: Гапанюк Ю. Е.

Дата: _____ Подпись:_____

Москва, 2018

Задание лабораторной работы
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

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`

Исходный код
librip/gen.py

```python
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
    for item in items:
```

```
        if len(args) > 1:
            dict = {}
            for key in args:
                if item[key]:
                    dict.update({key: item[key]})
            yield dict
        else:
            yield item[args[0]]


# 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):
    for i in range(num_count):
        yield random.randint(begin,end)
```

ex_1.py

```
#!/usr/bin/env python3
from librip.gens import field

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(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
# Реализация задания 1
```

Результат выполнения

['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}]

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`

Исходный код librip/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.counter = -1
        self.items = []
        if not kwargs.get("ignore_case"):
            if items:
                for x in items:
                    if x not in self.items:
                        self.items.append(x)
        else:
            if items:
                items = map(str.lower, items)
                for x in items:
                    if x not in self.items:
                        self.items.append(x)
        self.size = len(self.items)

    def __next__(self):
        if self.counter + 1 < self.size:
            self.counter += 1
        else:
            raise StopIteration()
        return self.items[self.counter]

    def __iter__(self):
        return self
```

ex_2.py

```
#!/usr/bin/env python3
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', 'b', 'C', 'c']
print(list(data2))
print("-----------------")
# Реализация задания 2
i = Unique(data3, ignore_case=True)
for _ in range(i.size):
```

```
    print(next(i), end=" ")
print("\n-----------------")

a = Unique(gen_random(1, 3, 10))
for _ in range(a.size):
    print(next(a), end=" ")
```

Результат работы

[3, 1, 3, 3, 1, 3, 2, 3, 2, 1]

---

a b c

---

3 1 2

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]`

Исходный код

ex_3.py

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

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

Результат работы

[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`

Исходный код librip/decorators.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs)
        print("Функция - {}, Результат:".format(func.__name__))
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for i in res:
                print(i, ' = ', res[i])
        else:
            print(res)
        return res
    return wrapper
```

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()
```

Результат работы

Функция - test_1, Результат:
1
Функция - test_2, Результат:
iu
Функция - test_3, Результат:
a = 1
b = 2
Функция - test_4, Результат:
1
2

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

ex_5.py

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

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

ctxmngts.py

```
import time


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

    def __exit__(self, *args, **kwargs):
        print(time.time()-self.start_time)
```

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.

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

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


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


@print_result
def f4(arg):
    return list("{}, зарплата - {} руб.".format(*i) for i in zip(arg,
list(gen_random(100000, 200000, len(arg)))))


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

Результат выполнения

ункция - f1, Результат:
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
…
электроэрозионист
эндокринолог
энергетик
энергетик литейного производства
энтомолог
юрисконсульт
юрисконсульт 2 категории
юрисконсульт. контрактный управляющий
юрист
юрист (специалист по сопровождению международных договоров, английский - разговорный)
юрист волонтер
юристконсульт
Функция - f2, Результат:

программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
программист/ junior developer
программист/ технический специалист
программистр-разработчик информационных систем
Функция - f3, Результат:
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
Функция - f4, Результат:
программист с опытом Python, зарплата - 190508 руб.
программист / senior developer с опытом Python, зарплата - 168381 руб.
программист 1с с опытом Python, зарплата - 163346 руб.
программист c# с опытом Python, зарплата - 182465 руб.
программист c++ с опытом Python, зарплата - 124653 руб.
программист c++/c#/java с опытом Python, зарплата - 173917 руб.
программист/ junior developer с опытом Python, зарплата - 126872 руб.
программист/ технический специалист с опытом Python, зарплата - 122188 руб.
программистр-разработчик информационных систем с опытом Python, зарплата - 104997 руб.
0.08777952194213867