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

Отчет

Лабораторная работа № 2

По курсу «Разработка интернет-приложений»

«Python. Функциональные возможности»

ИСПОЛНИТЕЛЬ:
Матиенко Андрей
Группа ИУ5-51
_____

"__"_____2019 г.

ПРЕПОДАВАТЕЛЬ:
Гапанюк Ю.Е.
_____

"__"_____2019 г.

Москва 2019

Задание

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

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`

```python
#!/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, None)))

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

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

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

print(list(field(goods, 'title', 'price', None, 'color', None)))

# Доп задания: кортежи квадрата чисел

print([(x, x * x) for x in range(1, 5)])

arr = [1, 2, 3, 4, 5]
a = map(lambda x: (x, x * x), arr)
print(list(a))
```

```python
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for es in items:
            if (args[0] in es and args[0] != 'None'):
                yield es[args[0]]
    else:
        for es in items:
            yield {j : es[j] for j in args if j != None}


# 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 i in range(0, num_count):
        yield random.randint(begin, end)
        pass
```

```
[]
['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}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}, {'title': 'Стелаж', 'price': 7000, 'color': 'white'}, {'title': 'Вешалка для оде
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}, {'title': 'Стелаж', 'price': 7000, 'color': 'white'}, {'title': 'Вешалка для оде
[(1, 1), (2, 4), (3, 9), (4, 16)]
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

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.*
Итератор должен располагаться в `librip/iterators.py`

```python
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.lst = items
        self.index = 0
        self.array = []
        self.bool = 0
        pass

    def __next__(self):
        if self.index == len(self.lst):
            raise StopIteration
        if self.lst[self.index] not in self.array:
            self.array.append(self.lst[self.index])
            self.index += 1
            return self.lst[self.index - 1]
        self.index += 1

    def __iter__(self):
        return self
```

```python
data1 = [1, 1, 1, 1, 3, 4, 4, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)

# Реализация задания 2

a = Unique(data1)
print([i for i in a if i != None])

data = ['a', 'B', 'A', 'b']
c = Unique(data)
print([i for i in c if i != None])
```

```
[1, 3, 4, 2]
['a', '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`

```python
import math

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3

# Самый короткий вариант
print(sorted(data, key=abs))
```

```
[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`

```python
def print_result(decorator_arg):
    def decorated_func():
        # Печать имени функции,  которую вызвали
        print(decorator_arg.__name__)
        #Если возвращаемое значение у функции 'int' or 'string' => просто выводим их на экран
        if isinstance(decorator_arg(), int) == True or isinstance(decorator_arg(), str) == True:
            print(decorator_arg())
        #Если возвращаемое значение у функции 'словарь' => печатаем ключ и значение, между ними '='
        elif isinstance(decorator_arg(), dict) == True:
            for key, value in decorator_arg().items():
                print(str(key) + ' = ' + str(value))
        # Если возвращаемое значение у функции 'массив' => печатаем в столбик его элементы
        elif isinstance(decorator_arg(), list) == True:
            for i in decorator_arg():
                print(i)
        # Возвращаем указатель на функцию
        return decorator_arg
    return decorated_func()
```

```python
@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
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

```python
def timer(func):
    import time

    def wrapper(*args, **kwargs):
        t = time.time()
        res = func(*args, **kwargs)
        print("Function " + str(func.__name__) + " : " + str(time.time() - t) + "(sec)")
    return wrapper

@timer
def good():
    import time
    time.sleep(5.5)

good()
```

```
Function good : 5.5001044273376465(sec)
```

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.

```
f1
1C программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
```

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`.

```
f2
1C программист
Web-программист
Веб - программист (PHP, JS) / Web разработчик
Веб-программист
```

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`.

```
f3
1C программист с опытом  Python
Web-программист с опытом  Python
Веб - программист (PHP, JS) / Web разработчик с опытом  Python
Веб-программист с опытом  Python
```

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.

```
f4
('1C программист с опытом  Python', 193958)
('Web-программист с опытом  Python', 100836)
('Веб - программист (PHP, JS) / Web разработчик с опытом  Python', 189541)
('Веб-программист с опытом  Python', 191854)
```

Вывод:

В данной лабораторной работе узнал о многих функциональных возможностях Python и научился применять их на практике.