Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Лабораторная работа №3

по курсу «Разработка интернет-приложений»

Выполнил

студент группы ИУ5-54Б

Сысойкин Е.М.

Москва, 2020

1 Общее описание задания

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

2 Выполнение задания

2.1 Задача 1(файл field.py)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. Пример:

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

`field(goods, 'title')` должен выдавать Ќовер́, Д́иван для отдыха

`field(goods, 'title', 'price')` должен выдавать

`{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}`

- В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количествово аргументов.
- Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
- Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

2.1.1 Код

```python
#!/usr/bin/env python

def field(items, *args):
    assert len(args) > 0
    for i in items:
        for j in args:
            yield i[j]
```

2.2 Задача 2 (файл gen_random.py)

Необходимо реализовать генератор gen_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

`gen_random(5, 1, 3)` должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

2.2.1 Код

```
#!/usr/bin/env python

import random

def gen_random(num_count, begin, end):
    while num_count>0:
        num_count-=1
        yield random.randint(begin,end+1)
```

2.3 Задача 3 (файл unique.py)

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

Конструктор итератора также принимает на вход именованный bool-параметр ignore_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

- При реализации необходимо использовать конструкцию **kwargs.
- Итератор должен поддерживать работу как со списками, так и с генераторами.
- 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(data) будет последовательно возвращать только 1, 2 и 3.

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

Unique(data) будет последовательно возвращать только a, A, b, B.

`Unique(data, ignore_case=True)` будет последовательно возвращать только a, b.

2.3.1 Код

```python
#!/usr/bin/env python

class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = list(items)
        self.uniqItems = list()
        self.index = 0
        try:
            self.ignore_case = bool(kwargs['ignore_case'])
        except (KeyError, TypeError):
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index >= len(self.items):
                raise StopIteration
            else:
                current = self.items[self.index]
                if (self.ignore_case == True) & (type(current) == str):
                    current = current.upper()
                if current not in self.uniqItems:
                    self.uniqItems.append(current)
                    return self.items[self.index]
                self.index = self.index + 1

    def __iter__(self):
        return self
```

2.4 Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted. Пример:

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

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

2.4.1 Код

```python
#!/usr/bin/env python

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

if __name__ == '__main__':
    result = sorted(data)
    print(result)

    result_with_lambda = lambda data: sorted(data)
    print(result_with_lambda(data))
```

2.5 Задача 5 (файл print_result.py)

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.

Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

2.5.1 Код

```python
#!/usr/bin/env python

def print_result(func_to_decorate):
    def decorated_func(*args,**kwargs):
        a = func_to_decorate(*args,**kwargs)
        print(func_to_decorate.__name__)
        if type(a) == list:
            for i in a:
                print(i)
            print("\n")
        elif type(a) == dict:
            for i, j in dict(a.items()):
                print(i, "=", j)
            print("\n")
        else:
            print(a,"\n")
        return a
```

```
    return decorated_func
```

2.6 Задача 6 (файл cm_timer.py)

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран. Пример:

```
with cm_timer_1():
    sleep(5.5)
```

После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами.

2.6.1 Код

```
#!/usr/bin/env python

from contextlib import contextmanager
import time

class cm_timer_2:

    def __enter__(self):
        self.enter_time=time.perf_counter()
        return 333

    def __exit__(self, exp_type, exp_value, traceback):
        print(time.perf_counter()-self.enter_time)




@contextmanager
def cm_timer_1():
    t=time.perf_counter()
    yield 333
    print(time.perf_counter()-t)

if __name__=="main":
    with cm_timer_1():
        time.sleep(2)

```

```
    with cm_timer_2():
        time.sleep(2)
```

2.7 Задача 7 (файл process_data.py)

- В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
- В файле data_light.json содержится фрагмент списка вакансий.
- Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
- Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print_result печатается результат, а контекстный менеджер cm_timer_1 выводит время работы цепочки функций.
- Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
- Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
- Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
- 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.
- 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.

2.7.1 Код

```
#!/usr/bin/env python

import json
import cm_timer
import field
import gen_random
import print_result
import unique
```

```python

@print_result.print_result
def f1(arg):
    return sorted(unique.Unique(
        list(field.field(arg, "job-name")),
        ignore_case = True),
            key=str.lower)


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


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


@print_result.print_result
def f4(arg):
    return list(zip(arg, map(
        lambda x: "зарплата " + str(x),
        gen_random.gen_random(len(arg), 100000, 200000))))


path = "data_light.json"

if __name__ == '__main__':
    with cm_timer.cm_timer_1():
        with open(path) as f:
            data = json.load(f)
            f4(f3(f2(f1(data))))
```

```
Электросварщик на автоматических и полуавтоматических машинах
Электросварщик ручной сварки
Электросварщики ручной сварки
Электрослесарь (слесарь) дежурный и по ремонту оборудования, старший
Электрослесарь по ремонту и обслуживанию автоматики и средств измерений электростанций
Электрослесарь по ремонту оборудования в карьере
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


f2
Программист
Программист / Senior Developer
Программист 1С
Программист С#
Программист С++
Программист С++/С#/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 С# с опытом Python
Программист С++ с опытом Python
Программист С++/С#/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 177711')
('Программист / Senior Developer с опытом Python', 'зарплата 154882')
('Программист 1С с опытом Python', 'зарплата 160377')
('Программист С# с опытом Python', 'зарплата 134716')
('Программист С++ с опытом Python', 'зарплата 193720')
('Программист С++/С#/Java с опытом Python', 'зарплата 159264')
('Программист/ Junior Developer с опытом Python', 'зарплата 155926')
('Программист/ технический специалист с опытом Python', 'зарплата 190416')
('Программистр-разработчик  информационных систем с опытом Python', 'зарплата 141034')


0.28994129807688296
[tujh@tujhNotebook lab3]$
```