Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет приложений»

Отчет по лабораторной работе №3
Вариант 12

Выполнил:
студент группы ИУ5-53Б
Кузнецов Г.И.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2021 г.

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.

Задача 1 (файл field.py)

Описание задачи:

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

- 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.

Текст программы:

```python
from typing import Any, Dict, Generator, Union

def field(items: list[dict], *args) -> Generator[Union[Any, dict], None, None]:
    assert len(args) > 0

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

Скриншот выполнения программы:

```
>>> for item in field(goods, 'title'): print(item)
...
Ковер
Диван для отдыха
>>> for item in field(goods, 'title', 'price'): print(item)
...
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
```

Задача 2 (файл gen_random.py)

Описание задачи:

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

Текст программы:

```python
from typing import Generator
from random import randint

def gen_random(amount: int, begin: int, end: int) -> Generator[int, None, None]:
    for i in range(amount):
        yield randint(begin, end)
```

Скриншот выполнения программы:

```
>>> for item in gen_random(5, 1, 3): print(item)
...
3
1
2
2
1
```

Задача 3 (файл unique.py)

Описание задачи:

- 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.

Текст программы:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self._iterator = iter(items)
        if (kwargs.get('ignore_case') is None):
            self._ignoreCase = False
        else:
            self._ignoreCase = kwargs.get('ignore_case')

    def __next__(self):
        item = next(self._iterator)

        try:
            if (self._ignoreCase == True):
                item = item.casefold()
        except: pass

        if (item not in self._found):
            self._found.append(item)
            return item

        return Unique.__next__(self)

    def __iter__(self):
        self._found = list()
        return self
```

Скриншот выполнения программы:

```
>>> data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
>>> for item in Unique(data): print(item)
...
1
2
```

Задача 4 (файл sort.py)

Описание задачи:

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.

Текст программы:

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

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

    result_with_lambda = sorted(data, key=lambda x: -x)
    print(result_with_lambda)
```

Скриншот выполнения программы:

```
PS C:\Users\Zid\Documents\Stuff\Учёба\5 сем\iu5-web-dev> py ./3/lab_python_fp/sort.py
[123, 100, 4, 1, 0, -1, -4, -30, -100]
[123, 100, 4, 1, 0, -1, -4, -30, -100]
```

Задача 5 (файл print_result.py)

Описание задачи:

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.

Текст программы:

```python
from typing import Iterable

def print_result(func):
    def wrapper(*args, **kwargs):
        print("{} : ".format(func.__name__), end='')

        result = func(*args, **kwargs)

        if (isinstance(result, Iterable)):
            print('')
            for i in result:
                print(i)
        elif (isinstance(result, dict)):
```

```
            print('')
            for i in result.items():
                print("{} = {}".format(i[0], i[1]))
        else:
            print("{}".format(result))

        return func(*args, **kwargs)

    return wrapper

@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 'iu5'


@print_result
def test_3():
    return {'a': 1, 'b': 2}


@print_result
def test_4():
    return [1, 2]


if __name__ == '__main__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

Скриншот выполнения программы:

```
PS C:\Users\Zid\Documents\Stuff\Учёба\5 сем\iu5-web-dev> py ./3/lab_python_fp/print_result.py
!!!!!!!!
test_1 : 1
test_2 :
i
u
5
test_3 :
a
b
test_4 :
1
2
```

Задача 6 (файл cm_timer.py)

Описание задачи:

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 (на основе класса и с использованием библиотеки contextlib).

Текст программы:

```
from typing import ContextManager
import time
from contextlib import contextmanager

@contextmanager
def cm_timer_2():
    tick = time.time()
    yield None
    print('time: {}'.format(time.time() - tick))

class cm_timer_1(ContextManager):
    def __init__(self):
        pass

    def __enter__(self):
        self._time = time.time()

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print("time: {}".format(time.time() - self._time))

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(1.2)
    with cm_timer_2():
        time.sleep(1.5)
```

Скриншот выполнения программы:

```
PS C:\Users\Zid\Documents\Stuff\Учёба\5 сем\iu5-web-dev> py ./3/lab_python_fp/cm_timer.py
time: 1.2007999420166016
time: 1.5005130767822266
```

Задача 7 (файл process_data.py)

Описание задачи:

- 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.

Текст программы:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```python
import json
import sys
from print_result import print_result
from unique import Unique
from field import field
from gen_random import gen_random
from cm_timer import cm_timer_1
# Сделаем другие необходимые импорты

path = sys.argv[1]

# Необходимо в переменную path сохранить путь к файлу, который был передан при
запуске сценария

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

# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
# В реализации функции f4 может быть до 3 строк

@print_result
def f1(arg):
    return sorted(Unique(field(arg, "job-name")))


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


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


@print_result
def f4(arg):
    joined = zip(arg, gen_random(len(arg), 100000, 200000))
    for i in joined:
        yield "{}, с зарплатой {} руб.".format(i[0], i[1])


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Скриншот выполнения программы:

```
художественный руководитель
художник
художник-постановщик
швея - мотористка
шеф-повар
шиномонтаж
шлифовщик 5 разряда
шлифовщик механического цеха
эколог
экономист
электрик
электрогазосварщик
электромонтер
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер по ремонту и обслуживанию электрооборудования
электромонтер станционного телевизионного оборудования
электросварщик
электросварщик на автоматических и полуавтоматических машинах
энтомолог
юрисконсульт
юрисконсульт 2 категории
юрист
f2 :
программист
программист 1С
f3 :
программист с опытом Python
программист 1С с опытом Python
f4 :
программист с опытом Python, с зарплатой 103829 руб.
программист 1С с опытом Python, с зарплатой 122824 руб.
time: 0.5959148406982422
```