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

ФАКУЛЬТЕТ **«Информатики и систем управления»**

КАФЕДРА **«Систем обработки информации и управления»**

Лабораторная работа 3

Функциональные возможности языка Python

ПО ДИСЦИПЛИНЕ: «Разработка интернет приложений»

Студент ИУ5-52Б Г.А. Савченко
(Группа) (И.О. Фамилия)
Email: sgfox4@gmail.com

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)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. 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'}`

- 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.

Шаблон для реализации генератора:

```
# 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
    # Необходимо реализовать генератор
```

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`
Шаблон для реализации генератора:

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

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.
Шаблон для реализации класса-итератора:

```
# 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 одинаковыми строки в
разном регистре
        # Например: ignore_case = True, Абв и АБВ - разные строки
        #           ignore_case = False, Абв и АБВ - одинаковые строки, одна из
которых удалится
        # По-умолчанию ignore_case = False
        pass

    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

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.

Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

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.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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()
```

Результат выполнения:

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

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.

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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

# 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):
    raise NotImplemented


@print_result
def f2(arg):
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
    raise NotImplemented


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

Текст программы

process_data.py:

```
import json
import sys
import requests

from lab_python_fp.unique import Unique
from lab_python_fp.gen_random import gen_random
from lab_python_fp.field import field
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1
# Сделаем другие необходимые импорты

path = 'data_light.json'

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

with open(path, 'r', encoding='utf8') as f:
    data = json.load(f)
    #data = requests.get(path).json()

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemen
ted`
# 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')), key=lambda i: str(i).lower())


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


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

@print_result
def f4(arg):
    return dict(zip(arg, gen_random(len(arg), 100000, 200000)))


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

cm_timer.py:

```
from time import time
from time import sleep
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.start_time = time()

    def __enter__(self):
        return

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print(f'time: {round(time() - self.start_time, 3)}')

@contextmanager
def cm_timer_2():
    start_time = time()
    yield
    print(f'time: {round(time() - start_time, 3)}')

with cm_timer_2():
    sleep(2.2)
```

field.py:

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            if item[args[0]] is not None: yield item[args[0]]
    elif len(args) > 1:
        for item in items:
            # Проверим, не являются ли все значения None
            count = 0
            for arg in args:
                if item[arg] is None: count += 1
            # Если нет, продолжаем
            if count != len(args):
                # Создаем словарь
                result = {}
                # Добавляем в него != None элементы
                for arg in args:
                    if item[arg] is not None:
                       result[arg] = item[arg]
                # Возвращаем словарь
                yield result
```

gen_random.py:

```
import random
def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)
```

print_result.py:

```
def print_result(func_to_print):

    def decorated_func(arg):
        # Вывод названия функции
        print(func_to_print.__name__)

        # Исполнение фукнции
        result = func_to_print(arg)

        # Вывод результата в зависимости от его типа
        if type(result) is list:
            for item in result:
                print(item)
        elif type(result) is dict:
            for key, item in result.items():
                print(f'{key} = {item}')
        else:
            print(result)

        # Возвращение значения функции
        return result

    return decorated_func
```

sort.py:

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

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

    result_with_lambda = sorted(data, key=lambda i: abs(i), reverse=True)
    print(result_with_lambda)
```

unique.py:

```
from lab_python_fp.gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.used = set()
        self.index = 0
        self.items = items if type(items) is list else [i for i in items]
        self.ignore_case = True if len(kwargs) > 0 and kwargs['ignore_case'] == T
rue else False

    def __next__(self):
        while True:
            if (self.index >= len(self.items)):
                raise StopIteration
            else:
                current = self.items[self.index]
                self.index += 1
                if (self.ignore_case and current not in self.used or not self.ign
ore_case and str(current).lower() not in self.used):
                    self.used.add(
                        current if self.ignore_case else str(current).lower())
                    return current

    def __iter__(self):
        return self
```

Пример выполнения программы

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
электросварщик
Электросварщик  на полуавтомат
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 C#
Программист C++
Программист C++/C#/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 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
f4
Программист с опытом Python = 142257
Программист / Senior Developer с опытом Python = 199315
Программист 1С с опытом Python = 189398
Программист C# с опытом Python = 129385
Программист C++ с опытом Python = 102112
Программист C++/C#/Java с опытом Python = 136830
Программист/ Junior Developer с опытом Python = 148273
Программист/ технический специалист с опытом Python = 112199
Программистр-разработчик  информационных систем с опытом Python = 131849
time: 3.989
```