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)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:
студент группы ИУ5-33Б
Шагиахметов А.Л.

Проверил:
Канев А.И.

2021 г.

Оглавление

Постановка задачи:.....3

Текст программы:.....4

Экранные формы с примерами выполнения программы:.....7

Постановка задачи:

Задание:

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 (на основе класса и с использованием библиотеки contextlib).

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:

Файл cm_timer.py

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


class cm_timer_2:
    def __init__(self):
        pass

    def __enter__(self):
        self.tick = time.perf_counter()
        pass

    def __exit__(self, exc_type, exc_val, exc_tb):
        toc = time.perf_counter()
        print(f"Вычисление заняло {toc - self.tick:0.4f} секунд")
        pass


@contextmanager
def cm_timer_1():
```

```
    try:
        tick = time.perf_counter()
        yield
    finally:
        toc = time.perf_counter()
        print(f"Вычисление заняло {toc - tick:0.4f} секунд")
        pass


def main():
    with cm_timer_1():
        sleep(1.5)
    with cm_timer_2():
        sleep(0.5)


if __name__ == "__main__":
    main()
```

Файл circle.py

```
def field(items, *args):
    assert len(args) > 0, 'Can`t be zero encounters'
    if len(args) == 1:
        for dictionary in items:
            note = dictionary.get(args[0])
            if note is not None:
                yield note
    else:
        for d in items:
            dictionary = dict()
            for key in args:
                note = d.get(key)
                if note is not None:
                    dictionary[key] = note
            if len(dictionary) != 0:
                yield dictionary


def main():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    print(field(goods, 'title', 'price'))


if __name__ == "__main__":
    main()
```

Файл gen_random.py

```
import random


def gen_random(num_count, begin, end):
    for i in range(num_count):
        yeild random.randint(begin, end)

def main():
```

```
    print(gen_random(3, 1, 2))


if __name__ == "__main__":
    main()
```

Файл print_result.py

```
# Здесь должна быть реализация декоратора
def print_result(func_to_decorate):
    def decorated_func(*args, **kwargs):
        incoming = func_to_decorate(*args, **kwargs)
        print(func_to_decorate.__name__)
        if isinstance(incoming, list):
            for i in incoming:
                print(i)
        elif isinstance(incoming, dict):
            for i in incoming:
                print(str(i) + ", " + str(incoming[i]))
        else:
            print(incoming)
        return incoming

    return decorated_func

@print_result
def f1():
    return 1


@print_result
def f2():
    return 'iu5'


@print_result
def f3():
    return {'a': 1, 'b': 2}


@print_result
def f4():
    return [1, 2]


def main():
    f1()
    f2()
    f3()
    f4()


if __name__ == '__main__':
    main()
```

Файл process_data.py

```
import json
import sys
```

```
from print_result import print_result
from cm_timer import cm_timer_1
from unique import Unique
from fields import field
from gen_random import gen_random

# Сделаем другие необходимые импорты

path = r'./data_light.json'

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

with open(path, encoding='utf-8') 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(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.lower)


@print_result
def f2(arg):
    return list(filter(lambda string: str.startswith(str.lower(string),
'программист'), arg))


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


@print_result
def f4(arg):
    return dict(zip(arg, list('зарплата {} руб.'.format(val) for val in
gen_random(len(arg), 1000000, 2000000))))


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

Файл 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 v: -v if v<0 else v,
reverse=True)
    print(result_with_lambda)
```

Файл uniq.py

```
# Итератор для удаления дубликатов
from gen_random import gen_random


class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.data = items
        self.ignore_case = False
        if len(kwargs) > 0:
            self.ignore_case = kwargs['ignore_case']

    def __next__(self):
        it = iter(self.data)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise StopIteration
            else:
                if self.ignore_case is True and isinstance(current, str):
                    current = current.lower()
                if current not in self.used_elements:
                    self.used_elements.add(current)
                    return current

    def __iter__(self):
        return self


def main():
    data = gen_random(3, 1, 2)
    iter = Unique(data, ignore_case=True)
    for i in iter:
        print(i)
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    iter = Unique(data, ignore_case=False)
    for i in iter:
        print(i)
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    iter = Unique(data, ignore_case=True)
    for i in iter:
        print(i)


if __name__ == "__main__":
    main()
```

Экранные формы с примерами выполнения программы:

Process data:

```
электромонтер по ремонту и обслуживанию оборудования подстанций
электромонтер по ремонту и обслуживанию электрооборудования
электромонтер по ремонту и обслуживанию электрооборудования 4 разряда-5 разряда
электромонтер по ремонту и обслуживанию электрооборудования 5 р.
электромонтер по ремонту и обслуживанию электрооборудования 5 разряда
электромонтер по ремонту и обслуживанию электрооборудования 6 разряда-6 разряда
электромонтер по ремонту оборудования зиф
электромонтер по ремонту электрооборудования гпм
электромонтер по эксплуатации и ремонту оборудования
электромонтер станционного телевизионного оборудования
электронщик
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 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 1546851 руб.
программист / senior developer с опытом python, зарплата 1443867 руб.
программист 1с с опытом python, зарплата 1632989 руб.
программист с# с опытом python, зарплата 1211913 руб.
программист с++ с опытом python, зарплата 1518833 руб.
программист с++/с#/java с опытом python, зарплата 1131620 руб.
программист/ junior developer с опытом python, зарплата 1886951 руб.
программист/ технический специалист с опытом python, зарплата 1370919 руб.
программистр-разработчик  информационных систем с опытом python, зарплата 1147635 руб.
Вычисление заняло 0.7885 секунд
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```

Sort:

```
PS C:\Users\Teacher\BKIT_2021\code\lab3_code> python .\sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```

CM Timer:

```
PS C:\Users\Teacher\BKIT_2021\code\lab3_code> python .\cm_timer.py
Вычисление заняло 1.5239 секунд
Вычисление заняло 0.5096 секунд
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```

Fields:

```
PS C:\Users\Teacher\BKIT_2021\code\lab3_code> python .\fields.py
<generator object field at 0x000001FFF67F1B48>
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```

Gen random:

```
PS C:\Users\Teacher\BKIT_2021\code\lab3_code> python .\gen_random.py
[2, 1, 1]
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```

Print result:

```
PS C:\Users\Teacher\BKIT_2021\code\lab3_code> python .\print_result.py
f1
1
f2
iu5
f3
a, 1
b, 2
f4
1
2
PS C:\Users\Teacher\BKIT_2021\code\lab3_code>
```