

Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Базовые компоненты интернет-технологий»
Отчёт по лабораторной работе №3

Выполнил:
студент группы ИУ5-31Б
Смыслов Дмитрий
Олегович

Подпись: _____

Дата: _____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Подпись: _____

Дата: _____

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_fr`. Решение каждой задачи должно располагаться в отдельном файле. При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Описание задачи 1

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 количество 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.

Файл `field.py`

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for dict in items:
            try:
                if dict[args[0]] is not None:
                    yield dict[args[0]]
            except KeyError:
                pass
    else:
        for dict in items:
            res = {}
            for key in args:
                try:
                    if dict[key] is not None:
                        res.update({key: dict[key]})
                except KeyError:
```

```

        pass
    if not len(res) == 0:
        yield res

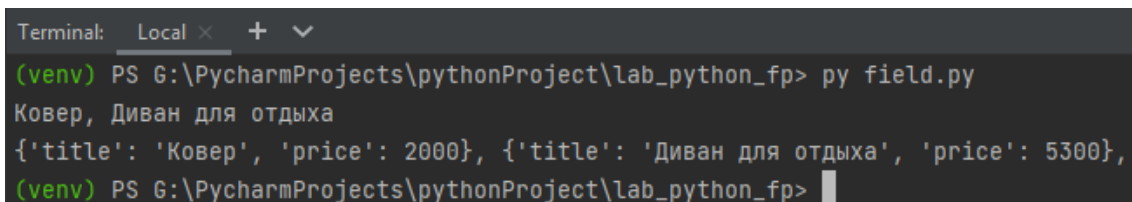
if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    print(', '.join(field(goods, 'title')))

    for i in field(goods, 'title', 'price'):
        print(i, end=', ')
    print()

```

Результат выполнения программы



```

Terminal: Local x + v
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py field.py
Ковер, Диван для отдыха
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300},
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp>

```

Описание задачи 2

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

Файл `gen_random.py`

```

from random import randint

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin, end)

if __name__ == '__main__':
    for i in gen_random(5, 1, 3):
        print(i, end=' ')
    print()

```

Результат выполнения программы

```
Terminal: Local x + v
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py gen_random.py
2 2 3 2 1
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> █
```

Описание задачи 3

- 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.

Файл unique.py

```
from gen_random import gen_random
```

```
class Unique(object):
```

```
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.data = items
        assert len(kwargs) < 2
        if len(kwargs) == 0:
```

```

        self.ignore_case = False
    else:
        try:
            self.ignore_case = kwargs['ignore_case']
        except KeyError as k:
            print('Неверное имя параметра итератора: ожидалось {}'.format(k))
            raise

    def __next__(self):
        iterator = iter(self.data)
        while True:
            try:
                current = next(iterator)
                if self.ignore_case and isinstance(current, str):
                    check = current[:].lower()
                    if check not in self.used_elements:
                        self.used_elements.add(check)
                        return current
                elif current not in self.used_elements:
                    self.used_elements.add(current)
                    return current
            except StopIteration:
                raise

    def __iter__(self):
        return self

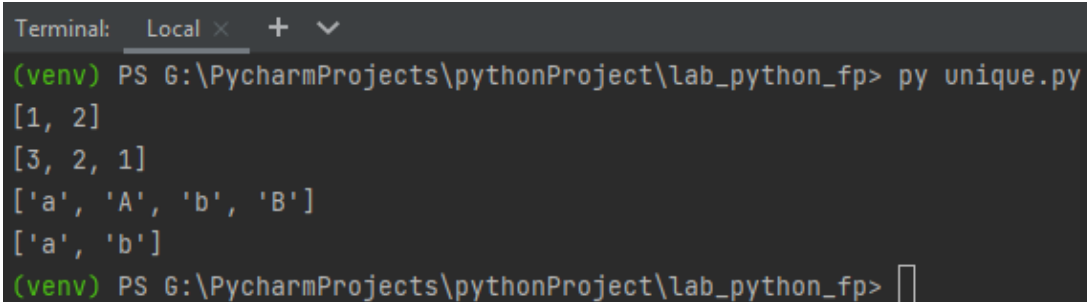
if __name__ == '__main__':
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print([i for i in Unique(data)])

    data = gen_random(10, 1, 3)
    print([i for i in Unique(data)])

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print([i for i in Unique(data)])
    print([i for i in Unique(data, ignore_case=True)])

```

Результат выполнения программы



```

Terminal: Local x + v
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py unique.py
[1, 2]
[3, 2, 1]
['a', 'A', 'b', 'B']
['a', 'b']
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> 

```

Описание задачи 4

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, который 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.

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 elem: -abs(elem))
    print(result_with_lambda)
```

Результат выполнения программы

```
Terminal: Local x + v
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> |
```

Описание задачи 5

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 выводиться в столбик через знак равенства.

Файл print_result.py

```
def print_result(func):  
  
    def decorated_func(*args, **kwargs):  
        print(func.__name__)  
        res = func(*args, **kwargs)  
        if isinstance(res, list):  
            for i in res:  
                print(i)  
        elif isinstance(res, dict):  
            for kw, arg in res.items():  
                print('{k} = {v}'.format(kw, arg))  
        else:  
            print(res)  
        return res  
  
    return decorated_func  
  
@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__':  
    test_1()  
    test_2()  
    test_3()  
    test_4()
```

Результат выполнения программы

```
Terminal: Local x + v  
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py print_result.py  
test_1  
1  
test_2  
iu5  
test_3  
a = 1  
b = 2  
test_4  
1  
2  
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp>
```

Описание задачи 6

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`).

Файл `cm_timer.py`

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

class cm_timer_1:

    def __init__(self):
        self.start_time = 0

    def __enter__(self):
        self.start_time = time()
        return 'Entering context'

    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() - self.start_time))

@contextmanager
def cm_timer_2():
    start_time = time()
    yield 'Entering context'
    print('time: {}'.format(time() - start_time))

if __name__ == '__main__':
    with cm_timer_1():
        sleep(5.5)

    with cm_timer_2():
        sleep(5.5)
```

Результат выполнения программы

```
Terminal: Local x + v
(venv) PS 6:\PycharmProjects\pythonProject\lab_python_fp> py cm_timer.py
time: 5.500076055526733
time: 5.500710964202881
(venv) PS 6:\PycharmProjects\pythonProject\lab_python_fp> |
```


Описание задачи 7

- 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 С# 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 С# с опытом Python, зарплата 137287 руб. Используйте `zip` для обработки пары специальность — зарплата.

Файл `process_data.py`

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

path = r'G:\PycharmProjects\pythonProject\data_light.json'

with open(path, encoding='utf-8') as f:
    data = json.load(f)

@print_result
def f1(arg):
```

```

        return sorted(list(Unique(list(field(arg, "job-name"))), ignore_case=True)),
key=lambda x: x.lower())

@print_result
def f2(arg):
    return list(filter(lambda job: job[:11].lower() == 'программист', arg))

@print_result
def f3(arg):
    return list(map(lambda x: '{} с опытом Python'.format(x), list(arg)))

@print_result
def f4(arg):
    salary = [i for i in gen_random(len(arg), 100000, 200000)]
    return ['{} зарплата {} руб.'.format(job, salary) for job, salary in zip(arg, salary)]

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

```

Результат выполнения программы

Начало выполнения функции f1:

```

Terminal: Local x + v
(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp> py process_data.py
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестящик

```

Конец выполнения:

```

юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский - разговорный)
Юрист волонтер
Юристконсульт
f2

```

f2

Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем

f3

Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1C 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
Программист-разработчик информационных систем с опытом Python

f4

f4

Программист с опытом Python, зарплата 127346 руб.
Программист / Senior Developer с опытом Python, зарплата 123908 руб.
Программист 1C с опытом Python, зарплата 193067 руб.
Программист C# с опытом Python, зарплата 167336 руб.
Программист C++ с опытом Python, зарплата 161414 руб.
Программист C++/C#/Java с опытом Python, зарплата 176382 руб.
Программист/ Junior Developer с опытом Python, зарплата 134457 руб.
Программист/ технический специалист с опытом Python, зарплата 143638 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 173640 руб.

time: 4.102105379104614

(venv) PS G:\PycharmProjects\pythonProject\lab_python_fp>