

**Московский государственный технический
университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра РТ5 «Системы обработки информации и управления»

Отчёт по лабораторной работе №3-4

Выполнил:
студент группы РТ5-31Б:
Пересыпко Александр Владимирович
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2023 г.

Задание

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.

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 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.

Код программы

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

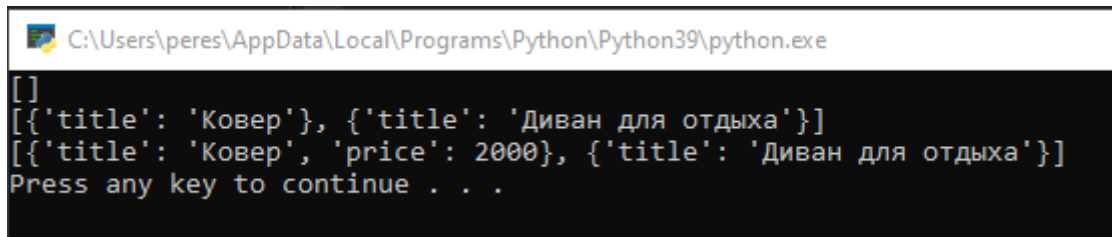
```
def field(items, *args):  
    result = []  
    for item in items:  
        constructed_item = {}  
        for key in args:  
            if key in item:  
                constructed_item[key] = item[key]  
        if constructed_item:  
            result.append(constructed_item)  
    return result
```

```
def main():  
    print(field(goods))  
    print(field(goods, 'title'))
```

```
print(field(goods, 'title', 'price'))

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

Результат выполнения



```
C:\Users\peres\AppData\Local\Programs\Python\Python39\python.exe

[{'title': 'Ковер'}, {'title': 'Диван для отдыха'}]
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]
Press any key to continue . . .
```

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

Код программы

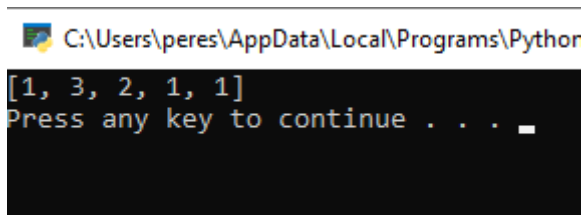
```
import random

def gen_random(amount, min, max):
    return [random.randint(min, max) for i in range(amount)]

def main():
    print(gen_random(5, 1, 3))

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

Результат выполнения



```
C:\Users\peres\AppData\Local\Programs\Python\Python39\python.exe

[1, 3, 2, 1, 1]
Press any key to continue . . .
```

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.

Код программы

```
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = 'ignore_case' in kwargs and kwargs['ignore_case']
        self.items = items
        self.originals = []
        self.item_index = -1
        pass

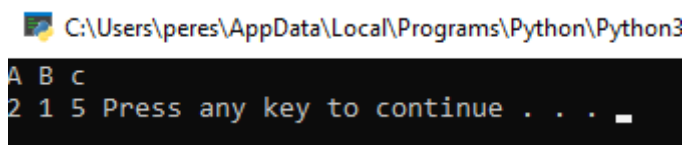
    def __next__(self):
        self.item_index += 1
        if self.item_index >= len(self.items): raise StopIteration
        is_item_string = isinstance(self.items[self.item_index], str)
        if (not self.items[self.item_index] in self.originals and (self.ignore_case and is_item_string or not is_item_string)) or (is_item_string and not self.ignore_case and len(list(filter(lambda item: isinstance(item, str) and item.lower() == self.items[self.item_index].lower(), self.originals))) == 0):
            self.originals.append(self.items[self.item_index])
            return self.items[self.item_index]
        return self.__next__()

    def __iter__(self):
        return self

def main():
    unique = Unique(['A', 'B', 'a', 'b', 'c', 'c'], ignore_case=False)
    for element in unique:
        print(element, end=' ')
    print("")
    unique = Unique(gen_random(10, 1, 6))
    for element in unique:
        print(element, end=' ')

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

Результат выполнения



```
C:\Users\peres\AppData\Local\Programs\Python\Python3
A B c
2 1 5 Press any key to continue . . . _
```

Задача 4 (файл sort.py)

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:

С использованием lambda-функции.

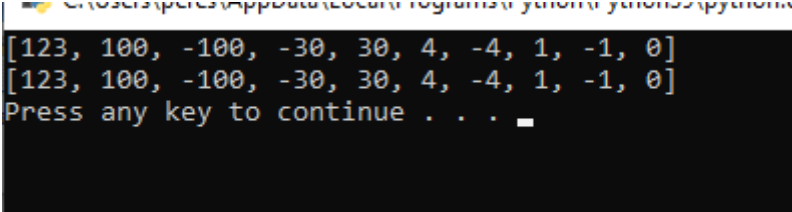
Без использования lambda-функции.

Код программы

```
def main():
    data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
    # Без lambda-функции
    print(sorted(data, key=abs, reverse=True))
    # С lambda-функцией
    print(sorted(data, key=lambda x: abs(x), reverse=True))

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

Результат выполнения



```
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Press any key to continue . . .
```

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 выводиться в столбик через знак равенства.

Код программы

```
def print_result(func):
    def wrapper(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if isinstance(result, list):
            [print(element) for element in result]
        elif isinstance(result, dict):
            [print('{} = {}'.format(key, value)) for key, value in result.items()]
        else:
            print(result)
        return result
    return wrapper

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu5'
```

```

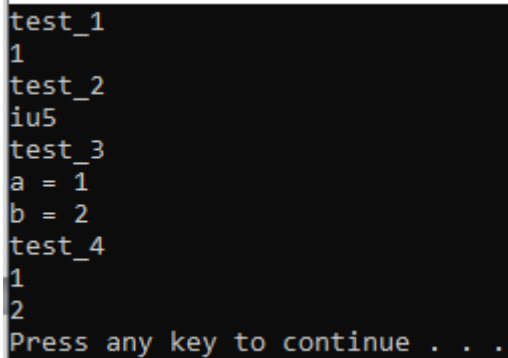
@print_result
def test_3():
    return {'a': 1, 'b': 2}

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

if __name__ == '__main__':
    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
Press any key to continue . . .

```

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 выводиться `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`).

Код программы

```

from contextlib import contextmanager
from time import time, sleep

# С помощью библиотеки
@contextmanager
def cm_timer_2():
    start = time()
    yield lambda: time() - start
    print('time: {}'.format(time() - start))

# С помощью класса

```

```

class cm_timer_1():
    def __enter__(self):
        self.start = time()
        return self

    def __exit__(self, type, value, traceback):
        print('time: {}'.format(time() - self.start))

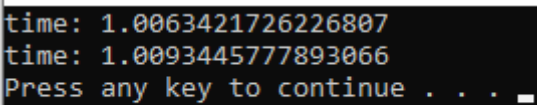
def main():
    with cm_timer_1():
        sleep(1)

    with cm_timer_2():
        sleep(1)

if __name__ == '__main__':
    main()

```

Результат выполнения



```

time: 1.0063421726226807
time: 1.0093445777893066
Press any key to continue . . . _

```

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 повторов (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.

Код программы

```
import json

# Импортируем предыдущие микрозадачи
from print_result import print_result
from cm_timer import cm_timer_1
from unique import Unique
from gen_random import gen_random

path = './data_light.json'

@print_result
def f1(arg):
    return sorted([x['job-name'] for x in Unique(arg)])

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

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

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

def main():
    with open(path, encoding='utf-8') as f:
        data = json.load(f)
        f.close()
    with cm_timer_1():
        f4(f3(f2(f1(data))))

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

Результат выполнения

f1

1С программист

2-ой механик

3-ий механик

4-ый механик

4-ый электромеханик

ASIC специалист

...

Программист

Программист / Senior Developer

Программист 1С

Программист C#

Программист C++

Программист C++/C#/Java

Программист/ Junior Developer

Программист/ технический специалист

Программист-разработчик информационных систем

программист 1С

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

Программист-разработчик информационных систем с опытом Python

программист 1С с опытом Python

f4

Программист с опытом Python, зарплата 164184 руб.

Программист / Senior Developer с опытом Python, зарплата 104709 руб.

Программист 1С с опытом Python, зарплата 141174 руб.

Программист C# с опытом Python, зарплата 173229 руб.

Программист C++ с опытом Python, зарплата 126632 руб.

Программист C++/C#/Java с опытом Python, зарплата 109000 руб.

Программист/ Junior Developer с опытом Python, зарплата 117675 руб.

Программист/ технический специалист с опытом Python, зарплата 167829 руб.

Программист-разработчик информационных систем с опытом Python, зарплата 192783 руб.

time: 1.1578314304351807

