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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «БКИТ»
Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-35Б
Трифонов Дмитрий
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Нардид А.Н.
Подпись и дата:

Москва, 2022 г.

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(10, 1, 3)`
`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

Файлы пакета lab_python_fp:

field.py

```
def field(items, *args):
    assert len(args) > 0
    if not(len(items)):
        return
    for arg in args:
        if not(arg in items[0].keys()):
            return
    if len(args) == 1:
        for item in items:
            yield item[args[0]]
    else:
        for item in items:
            record = {}
            for arg in args:
                record[arg] = item[arg]
            yield record



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

    gen1 = field(goods, 'title') # должен выдавать 'Ковер', 'Диван для отдыха'
    gen2 = field(goods, 'title', 'price') # должен выдавать {'title': 'Ковер',
'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}

    for i in gen1:
        print(i, end=' ')
    print()

    for i in gen2:
        print(i, end=' ')
    print()

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

gen_random.py

```
import random
def gen_random(num_count, begin, end):
    assert begin < end
    assert num_count >= 0

    for i in range(num_count):
        yield random.randint(begin, end)

gen1 = gen_random(1, 2, 3)
gen2 = gen_random(3, 5, 25)

def main():
    for i in gen1:
        print(i, end=' ')
    print()

    for i in gen2:
        print(i, end=' ')

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

unique.py

```
class Unique(object):
    def __init__(self, items, ignore_case = False, **kwargs):
        self.ignore_case = ignore_case
        self.data = items
        self.occured = set()
        self.index = -1
        pass

    def __next__(self):
        while True:
            self.index += 1
            if self.index >= len(self.data):
                raise StopIteration
            if self.check():
                current = self.data[self.index]
                self.occured.add(current)
                return current
        pass
```

```
    def __iter__(self):
        return self

    def check(self):
        el = self.data[self.index]
        if self.ignore_case:
            if type(el) == str:
                return not((el.lower() in self.occured) or (el.upper() in self.occured))
            else:
                return not(el in self.occured)
        else:
            return not(el in self.occured)
        pass


def main():
    for i in Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']):
        print(i)

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

sort.py

```
def sort_wo_lambda(data):
    return sorted(data, key=abs, reverse=True)

def sort_lambda(data):
    return sorted(data, key = lambda x: abs(x), reverse=True)

def main():
    data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
    print(sort_lambda(data))
    print(sort_wo_lambda(data))

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

print_result.py

```
def print_result(fun):
    def decorator(lst = [], *args, **kwargs):
        print(fun.__name__)

        if len(lst):
            result = fun(lst, *args, **kwargs)
        else:
```

```
            result = fun(*args, **kwargs)

        if type(result) == dict:
            for key, el in result.items():
                print(f'{key} = {el}')

        elif type(result) == list:
            for i in result:
                print(i)
        else:
            print(result)

        return result

    return decorator


@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]


def main():
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()

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

square.py

```
    from time import *
    from contextlib import *
    class cm_timer_1():

        def __init__(self):
            self.start = 0
            self.finish = 0

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

        def __exit__(self, ex_type, ex_value, ex_traceback):
            self.finish = time()
            print("time: ", self.finish - self.start)


    @contextmanager
    def cm_timer_2():
        start = time()
        yield None
        finish = time()
        print("time: ", finish - start)

    def main():
        with cm_timer_1():
            sleep(1.5)
        with cm_timer_2():
            sleep(2)

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

Финальная задача
process_data.py

```
import json

from lab_python_fp.print_result import print_result
from lab_python_fp.gen_random import gen_random
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.unique import Unique
from lab_python_fp.field import field

path = 'lab/3-4/data_light.json'

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

```python
@print_result
def f1(arg):
    return sorted(list(j for j in Unique(list(i for i in field(arg, 'job-name')))),
key=lambda x: x.lower())


@print_result
def f2(arg):
    return list(filter(lambda s: (s.split())[0].lower() == 'программист', arg))


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


@print_result
def f4(arg):
    return dict(zip(arg, ['зарплата ' + str(i) + ' руб.' for i in gen_random(len(arg),
100000, 200000)]))


def main():
    with cm_timer_1():
        f4(f3(f2(f1(data))))

main()
```

Экранные формы

Задача 1

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python field.py
Ковер Диван для отдыха
{'title': 'Ковер', 'price': 2000} {'title': 'Диван для отдыха', 'price': 5300}
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 2

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python gen_random.py
3
22 5 12
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 3

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ ТЕРМИНАЛ JUPYTER КОНСОЛЬ ОТЛАДКИ

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python unique.py
a
A
b
B
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 4

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python sort.py
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 5

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 6

```
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp> python cm_timer.py
time:  1.5084717273712158
time:  2.010050058364868
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4\lab_python_fp>
```

Задача 7

```
f2
Программист
программист
Программист / Senior Developer
Программист 1С
программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
f3
Программист с опытом Python
программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С 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
f4
Программист с опытом Python = зарплата 152934 руб.
программист с опытом Python = зарплата 194339 руб.
Программист / Senior Developer с опытом Python = зарплата 178987 руб.
Программист 1С с опытом Python = зарплата 186536 руб.
программист 1С с опытом Python = зарплата 109866 руб.
Программист C# с опытом Python = зарплата 125793 руб.
Программист C++ с опытом Python = зарплата 144970 руб.
Программист C++/C#/Java с опытом Python = зарплата 161493 руб.
time:  0.3591740131378174
PS C:\Users\Dmitriy\Documents\study\BKIT\lab\3-4>
```

Фрагмент вывода для f1:

Электромонтер
ЭЛЕКТРОМОНТЕР ПО РЕМОНТУ И ОБСЛУЖИВАНИЮ ОБОРУДОВАНИЯ
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
Электромонтер контактной сети
Электромонтер линейных сооружений телефонной связи и радиофикации
Электромонтер охранно-пожарной сигнализации
Электромонтер охранно-пожарной сигнализации и систем видеонаблюдения
электромонтер по испытаниям и измерениям 4-6 разряд
Электромонтер по обслуживанию электрооборудования электростанций
Электромонтер по ремонту и обслуживанию и ремонту электрооборудования
Электромонтер по ремонту и обслуживанию оборудования
Электромонтер по ремонту и обслуживанию оборудования подстанций
Электромонтер по ремонту и обслуживанию электрооборудования
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