Московский государственный технический

Университет им Н.Э.Баумана

Факультет «Информатика и системы управление»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3-4

Выполнил:

Студент группы ИУ5-34Б

Малышко А. В.

Подпись и дата:

Проверил:

Преподаватель каф. ИУ5

Нардид А. Н.

Подпись и дата:

Москва 2024 г.

Постановка задачи

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

field.py (Задача 1)

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]
def field(items, *args):
    assert len(args)>0
    for item in items:
        result={key: item.get(key) for key in args if item.get(key) is not
None}
        if len(args)==1:
            yield result[args[0]]
        else:
            yield result

Nekto = list(field(goods,'title'))
print(', '.join(Nekto))
Nekto2 =list(field(goods,'title','price'))
print(', '.join(str(Nekto3) for Nekto3 in Nekto2))
```

gen_random.py (Задача 2)

```
from random import randint

def get_random(n,minn,maxx):
    l=[]
    for i in range(n):
        l.append(randint(minn,maxx))
    yield from l
if __name__ =='__main__':
    print(*get_random(5, 1, 3))
```

unique.py (Задача 3)

```
from gen_random import get_random
class Unique(object):
    def __init__(self, items, **kwargs):
        self.data=iter(items)
        self.ignore_case=kwargs.get('ignore_case',False)
        self.unique_items=set()
```

```
    def __next__(self):
        while True:
            item=next(self.data)
            nekto = item.lower() if self.ignore_case else item
            if nekto not in self.unique_items:
                self.unique_items.add(nekto)
                return item

    def __iter__(self):
        return self
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
unique_iter = Unique(data)
print(*unique_iter)
data = get_random(10, 1, 3)
unique_iter = Unique(data)
print(*unique_iter)
data = ['a', 'A','b', 'B', 'a', 'A', 'b', 'B']
unique_iter = Unique(data)
print(*unique_iter)
unique_iter_ignore_case = Unique(data, ignore_case=True)
print(*unique_iter_ignore_case)
```

sort.py (Задача 4)

```
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 x: abs(x),reverse=True)
    print(result_with_lambda)
```

print_result.py (Задача 5)

```
def print_result(func):
    def wrapper(*args, **kwargs):
        nekto = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(nekto,list):
            for i in nekto:
                print(i)
        elif isinstance(nekto,dict):
            for k,v in nekto.items():
                print(f'{k}={v}')
        else:
            print(nekto)
        return nekto
    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__':
    print('!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

cm_timer.py (Задача 6)

```
import time
from contextlib import contextmanager
from time import sleep
class cm_timer_1():
    def __enter__(self):
        self.start_time = time.time()
        return self
    def __exit__(self, exc_type, exc_value, traceback):
        end_time = time.time()
        n_time = end_time - self.start_time
        print(f'time: {n_time}')

@contextmanager
def cm_timer_2():
    start_time=time.time()
    try:
        yield
    finally:
        end_time=time.time()
        n_time=end_time-start_time
        print(f'time: {n_time}')
if __name__ =='__main__':
    with cm_timer_1():
        sleep(5.5)
    with cm_timer_2():
        sleep(5.5)
```

process_data.py (Задача 7)

```
from cm_timer import cm_timer_1
from print_result import print_result
from random import randint
```

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

path = "C:/Artem/Labs/3semPython/3-4 лаба/pythonProject/data.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(set(item["job-name"].lower() for item in 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: f"{x} с опытом Python", arg))


@print_result
def f4(arg):
    job_names = arg
    salaries = [f"зарплата {randint(100000, 200000)} руб." for _ in
job_names]
    return [f"{job}, {salary}" for job, salary in zip(job_names, salaries)]

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

Экранные формы с примерами выполнения программы

field.py (Задача 1)

```
Ковер, Диван для отдыха
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

Process finished with exit code 0
```

gen_random.py (Задача 2)

```
1 2 2 3 3

Process finished with exit code 0
```

unique.py (Задача 3)

```
1 2
2 1 3
a A b B
a b

Process finished with exit code 0
```

sort.py (Задача 4)

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

Process finished with exit code 0
```

print_result.py (Задача 5)

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

Process finished with exit code 0
```

cm_timer.py (Задача 6)

```
time: 5.511083126068115
time: 5.505635738372803

Process finished with exit code 0
```

process_data.py (Задача 7)

```
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
```

```
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 115276 руб.
программист / senior developer с опытом Python, зарплата 143880 руб.
программист 1с с опытом Python, зарплата 189290 руб.
программист с# с опытом Python, зарплата 183505 руб.
программист с++ с опытом Python, зарплата 156351 руб.
программист с++/с#/java с опытом Python, зарплата 113332 руб.
программист/ junior developer с опытом Python, зарплата 157159 руб.
программист/ технический специалист с опытом Python, зарплата 111837 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 132806 руб.
time: 0.023960113525390625

Process finished with exit code 0
```