Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3-4
“Функциональные возможности языка Python.”

Выполнил:
студент группы ИУ5-35Б:
Тенишев А.А.
Подпись и дата:

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

Москва, 2024 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.

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

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.

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:
С использованием lambda-функции.
Без использования lambda-функции.

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.

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.

Текст программы.

Файл field.py

```
# 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, "Нет ни одного аргумента"

    result = []
    if len(args) == 1:
        for it in items:
            value = str(it.get(args[0]))
            if value:
```

```
                result.append(f"{value}")

    else:
        for it in items:
            value = {key: it.get(key) for key in args if it.get(key) is not None}
            if value:
                result.append(value)

    return result

if __name__ == '__main__':
    titles = field(goods, 'title')
    print(", ".join(f"'{title}'" for title in titles))
    titles = field(goods, 'title', 'price')
    print(", ".join(f"{title}" for title in titles))
```

Файл gen_random.py

```
import random
# 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):
    result = [random.randint(begin,end) for _ in range(num_count)]
    return result

if __name__ == '__main__':
    print(gen_random(5, 1, 3))
```

Файл unique.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = iter(items)
        self.myset = set()
        self.ignore_case = kwargs.get('ignore_case', False)  # Получаем параметр
ignore_case, по умолчанию False


    def __next__(self):
        item = next(self.items)  # Получаем следующий элемент
        if self.ignore_case and isinstance(item, str):
            item_to_check = item.lower()
        else:
            item_to_check = item
        while item_to_check in self.myset:
            item = next(self.items)  # Получаем следующий элемент (уже в цикле)
            if self.ignore_case and isinstance(item, str):
                item_to_check = item.lower()
            else:
                item_to_check = item
```

```
        self.myset.add(item_to_check)  # Добавляем в множество
        return item

    def __iter__(self):
        return self

if __name__ == '__main__':
    data = ['a', 'A', 'b', 'b', 'c', 'C', 'd']
    unique_iterator = Unique(data, ignore_case=True)

    for item in unique_iterator:
        print(item)
```

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

Файл print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(res, list):
            for i in res:
                print(i)
            return res
        if isinstance(res, dict):
            for key, value in res.items():
                print("{0} = {1}".format(key,value))
            return res
        print(res)
        return res
    return wrapper

@print_result
def test_1():
    return 1


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

```python
@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()
```

Файл cm_timer.py

```python
import time
import contextlib
from time import sleep
from contextlib import contextmanager

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time} ")

@contextmanager
def cm_timer_2():
    start_time = time.time()
    try:
        yield  # Позволяем выполнить блок кода
    finally:
        elapsed_time = time.time() - start_time
        print(f"time: {elapsed_time} ")

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

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

Файл process_data.py

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

path = './data_light.json'

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

@print_result
def f1(arg):
    return list(Unique(field(arg, "job-name"), ignore_case=True))


@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):
    salaries = gen_random(len(arg), 100000, 200000)
    return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg,
salaries)]


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

Результаты выполнения.

Файл field.py

```
PS C:\vsc\sem3\lol> python field.py
'Ковер', 'Диван для отдыха'
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}
```

Файл gen_random.py

```
PS C:\vsc\sem3\lol> python gen_random.py
[2, 3, 1, 3, 1]
PS C:\vsc\sem3\lol> python gen_random.py
[2, 2, 1, 1, 3]
```

Файл unique.py

```
PS C:\vsc\sem3\lol> python unique.py
a
b
c
d
```

Файл sort.py

```
PS C:\vsc\sem3\lol> python sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

Файл print_result.py

```
PS C:\vsc\sem3\lol> python print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Файл cm_timer.py

```
PS C:\vsc\sem3\lol> python cm_timer.py
time: 5.500978469848633
time: 5.500800609588623
```

Файл process_data.py

```
изма', 'Разработчик мобильных приложений', 'директор загородного лагеря', 'Портной', 'специалист отдела аренды', 'Инженер-механик', 'Разработчик импульсных источник
ов питания', 'Механик по эксплуатации транспортного отдела', 'Инженер-технолог по покраске', 'Бетонщик - арматурщик', 'главный инженер финансово-экономического отде
ла', 'Секретарь судебного заседания в аппарате мирового судьи Железнодорожного судебного района города Ростова-на-Дону', 'варщик зефира', 'варщик мармеладных издели
й', 'Оператор склада', 'Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем', 'Заведующий музеем в д.Копорье', 'Документовед', 'Спец
иалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем', 'Менеджер (в промышленности)']
f2
Программист
Программист C++/C#/Java
Программист 1С
Программистр-разработчик  информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
['Программист', 'Программист C++/C#/Java', 'Программист 1С', 'Программистр-разработчик  информационных систем', 'Программист C++', 'Программист/ Junior Developer',
'Программист / Senior Developer', 'Программист/ технический специалист', 'Программист C#']
1
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1С с опытом Python
Программистр-разработчик  информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python, зарплата 115387 руб.
Программист C++/C#/Java с опытом Python, зарплата 100584 руб.
Программист 1С с опытом Python, зарплата 197413 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 159660 руб.
Программист C++ с опытом Python, зарплата 190760 руб.
Программист/ Junior Developer с опытом Python, зарплата 109049 руб.
Программист / Senior Developer с опытом Python, зарплата 118182 руб.
Программист/ технический специалист с опытом Python, зарплата 149119 руб.
Программист C# с опытом Python, зарплата 176316 руб.
time: 0.3925504684448242
```