**Московский государственный технический**

**университет им. Н.Э. Баумана**

**Факультет ИУ**

**Кафедра ИУ5**

**Курс «Парадигмы и конструкции языков программирования»**

**Отчет лабораторной работе №3-4**

Выполнил студент группы ИУ5-33Б:

Костяев В.Д.

Подпись и дата:

Проверил преподаватель каф.:

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 г

**Задание:**

**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, Aбв и АБВ - разные строки**

**# ignore\_case = False, Aбв 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**](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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:**

**gen\_random.py**

**import random**

**def gen\_random(num\_count, begin, end):**

**for \_ in range(num\_count):**

**yield random.randint(begin, end)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("Случайные числа от 1 до 3:")**

**for num in gen\_random(5, 1, 3):**

**print(num)**

**print\_result.py**

**python**

**Копировать код**

**from functools import wraps**

**def print\_result(func):**

**@wraps(func)**

**def wrapper(\*args, \*\*kwargs):**

**result = func(\*args, \*\*kwargs)**

**print(func.\_\_name\_\_)**

**if isinstance(result, list):**

**for item in result:**

**print(item)**

**elif isinstance(result, dict):**

**for key, value in result.items():**

**print(f"{key} = {value}")**

**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\_\_':**

**print('!!!!!!!!')**

**test\_1()**

**test\_2()**

**test\_3()**

**test\_4()**

**process\_data.py**

**import json**

**import sys**

**import os**

**import random**

**from print\_result import print\_result**

**from cm\_timer import cm\_timer\_1**

**from unique import Unique**

**from field import field**

**from gen\_random import gen\_random**

**def get\_json\_data(path):**

**with open(path, encoding='utf-8') as f:**

**data = json.load(f)**

**return data**

**@print\_result**

**def f1(data):**

**job\_names = list(field(data, 'job-name'))**

**unique\_jobs = sorted(Unique(job\_names, ignore\_case=True), key=lambda x: x.lower())**

**return unique\_jobs**

**@print\_result**

**def f2(job\_list):**

**return list(filter(lambda x: x.lower().startswith('программист'), job\_list))**

**@print\_result**

**def f3(job\_list):**

**return list(map(lambda x: f"{x} с опытом Python", job\_list))**

**@print\_result**

**def f4(job\_list):**

**salaries = gen\_random(len(job\_list), 100000, 200000)**

**return [f"{job}, зарплата {salary} руб." for job, salary in zip(job\_list, salaries)]**

**if \_\_name\_\_ == '\_\_main\_\_':**

**path = os.path.join(os.path.dirname(\_\_file\_\_), 'data\_light.json')**

**if not os.path.exists(path):**

**print(f"Файл {path} не найден.")**

**sys.exit(1)**

**data = get\_json\_data(path)**

**with cm\_timer\_1():**

**f4(f3(f2(f1(data))))**

**sort.py**

**data = [4, -30, 100, -100, 123, 1, 0, -1, -4, 30]**

**if \_\_name\_\_ == "\_\_main\_\_":**

**#lambda-функция**

**result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)**

**print("С использованием lambda-функции:")**

**print(result\_with\_lambda)**

**#Без lambda-функции**

**def abs\_key(x):**

**return abs(x)**

**result\_without\_lambda = sorted(data, key=abs\_key, reverse=True)**

**print("\nБез использования lambda-функции:")**

**print(result\_without\_lambda)**

**unique.py**

**class Unique:**

**def \_\_init\_\_(self, items, \*\*kwargs):**

**self.items = iter(items)**

**self.ignore\_case = kwargs.get('ignore\_case', False)**

**self.seen = set()**

**def \_\_iter\_\_(self):**

**return self**

**def \_\_next\_\_(self):**

**while True:**

**current = next(self.items)**

**comparison\_value = current**

**if self.ignore\_case and isinstance(current, str):**

**comparison\_value = current.lower()**

**if comparison\_value not in self.seen:**

**self.seen.add(comparison\_value)**

**return current**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("Пример с числами:")**

**data\_numbers = [1, 1, 2, 2, 3, 3, 3]**

**for item in Unique(data\_numbers):**

**print(item)**

**print("\nПример со строками без игнорирования регистра:")**

**data\_strings = ['a', 'A', 'b', 'B', 'a', 'A']**

**for item in Unique(data\_strings):**

**print(item)**

**print("\nПример со строками с игнорированием регистра:")**

**for item in Unique(data\_strings, ignore\_case=True):**

**print(item)**

**field.py**

**def field(items, \*args):**

**assert len(args) > 0, "Необходимо указать хотя бы одно поле"**

**if len(args) == 1:**

**field\_name = args[0]**

**for item in items:**

**value = item.get(field\_name)**

**if value is not None:**

**yield value**

**else:**

**for item in items:**

**result = {field\_name: item.get(field\_name) for field\_name in args if item.get(field\_name) is not None}**

**if result:**

**yield result**

**if \_\_name\_\_ == "\_\_main\_\_":**

**goods = [**

**{'title': 'Ковер', 'price': 2000, 'color': 'green'},**

**{'title': 'Диван для отдыха', 'color': 'black'}**

**]**

**print("Вывод одного поля 'title':")**

**for value in field(goods, 'title'):**

**print(value)**

**print("\nВывод полей 'title' и 'price':")**

**for value in field(goods, 'title', 'price'):**

**print(value)**

**cm\_timer.py**

**import time**

**from contextlib import contextmanager**

**class cm\_timer\_1:**

**def \_\_enter\_\_(self):**

**self.start\_time = time.time()**

**return self**

**def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):**

**end\_time = time.time()**

**print(f"time: {end\_time - self.start\_time:.2f} сек")**

**@contextmanager**

**def cm\_timer\_2():**

**start\_time = time.time()**

**try:**

**yield**

**finally:**

**end\_time = time.time()**

**print(f"time: {end\_time - start\_time:.2f} сек")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("Тестирование cm\_timer\_1:")**

**with cm\_timer\_1():**

**time.sleep(2)**

**print("\nТестирование cm\_timer\_2:")**

**with cm\_timer\_2():**

**time.sleep(3)**

**Экранные формы с примерами выполнения программы:**

