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:
Сикоринский Артемий,
ИУ5-33Б

Проверил:
Гапанюк Е.Ю.

2024 г.

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

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.

Текст программы:

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            value = item.get(args[0])
            if value is not None:
                yield value
    else:
        for item in items:
            result = {arg: item.get(arg) for arg in args if item.get(arg) is not
None}
            if result:
                yield result

goods = [
```

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

print([i for i in field(goods, 'title')])
print([i for i in field(goods, 'title', 'price')])
```

Вывод:

```
C:\Users\Gkovd\PycharmProjects\lab2\.venv\Scripts\python.exe C:\Users\Gko
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]
```

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`

```
import random

def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)

if __name__ == "__main__":
    print([i for i in gen_random(5, 1, 3)])
```

```
C:\Users\Gkovd\PycharmProjects\lab2\.ven
[1, 1, 3, 2, 3]

Process finished with exit code 0
```

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.

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = items
        self.ignore_case = kwargs.get('ignore_case', False)
        self.seen = set()

    def __iter__(self):
        return self

    def __next__(self):
        for item in self.items:
            key = item.lower() if not self.ignore_case and isinstance(item, str)
else item
            if key not in self.seen:
                self.seen.add(key)
```

```
                return item
        raise StopIteration

if __name__ == "__main__":
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print([i for i in Unique(data)])
```

```
C:\Users\Gkovd\PycharmProjects\lab2\.venv\Scripts\python.exe C:\Users
[1, 2]

Process finished with exit code 0
```

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.

```
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)
```

```
C:\Users\Gkovd\PycharmProjects\lab2\.venv\Scripts\python.exe C:\Users\Gkovd\PycharmProjects\lab2\sort
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

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.

```
def print_result(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()
```

```
C:\Users\Gkovd\PycharmProjects\lab2\.venv\Scripts\python.exe (
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

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).

```
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_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time cm_timer_1: {elapsed_time:.2f}")

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time cm_timer_2: {elapsed_time:.2f}")

with cm_timer_1():
    sleep(5.5)
with cm_timer_2():
    sleep(5.5)
```

```
C:\Users\Gkovd\PycharmProjects\lab2\.venv\Scripts\python.exe C:\Users\Gkovd\Pych
time cm_timer_1: 5.50
time cm_timer_2: 5.50

Process finished with exit code 0
```

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.

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

path = "data_light.json"

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

@print_result
def f1(arg):
    return sorted(Unique(item['job-name'] for item in arg), key = lambda x:
x.lower())

@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"{profession}, зарплата {salary} руб." for profession, salary in
zip(arg, salaries)]

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

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестянщик
```

```
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
```

```
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
Программистр-разработчик  информационных систем с опытом Python
```

```
систем
```

```
f4
Программист с опытом Python, зарплата 141498 руб.
Программист / Senior Developer с опытом Python, зарплата 160096 руб.
Программист 1С с опытом Python, зарплата 162540 руб.
Программист C# с опытом Python, зарплата 100329 руб.
Программист C++ с опытом Python, зарплата 179766 руб.
Программист C++/C#/Java с опытом Python, зарплата 148540 руб.
Программист/ Junior Developer с опытом Python, зарплата 161422 руб.
Программист/ технический специалист с опытом Python, зарплата 195804 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 176951 руб.
```