осударственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

ЛАБОРАТОРНАЯ РАБОТА

№ 3-4

Функциональные возможности языка Python

Группа ИУ5-35Б

Студент _____ **16.12.2024 /А. А. Торопыгин/**

(Подпись, дата) (И.О.Фамилия)

Преподаватель _____ _____ **/Ю. Е. Гапанюк/**

(Подпись, дата) (И.О.Фамилия)

2024

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 располагаться 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.

- 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:

```python
def field(items, *args):
    assert len(args) > 0
    result = []

    if len(args) == 1:
        for item in items:
            result.append(item[args[0]])
    else:
        for item in items:
            result.append({i: item[i] for i in args if i in item.keys()})

    return result

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

print(field(goods, 'title'))
print(field(goods, 'title', 'price'))
```

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyth
on_fp/field.py
['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.

Текст программы:

```
import random as rd



def gen_random(num_count, begin, end):
    return [rd.randint(begin, end) for _ in range(num_count)]

print(gen_random(5, 0, 10))
```

Вывод:

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.

Текст программы:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = []
        if 'ignore_case' in kwargs.keys() and kwargs['ignore_case']:
            self.items = list(set(map(lambda x: x.lower() if type(x) == str
else x, items)))
        else:
            self.items = list(set(items))

        self.start = 0
        self.stop = len(self.items)
        self.position = -1


    def __next__(self):
        self.position += 1
        if self.position < self.stop:
            return self.items[self.position]
        else:
            return ''

    def __iter__(self):
        return self
```

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

Udata = Unique(data)
print(Udata.items)

Udata = Unique(data, ignore_case = True)
print(Udata.items)
```

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyth
on_fp/unique.py
['b', 'B', 'a', 'A']
['b', 'a']
```

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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы:

```
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: x if x > 0 else -x,
reverse=True)
    print(result_with_lambda)
```

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyth
on_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 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.

Текст программы:

```
def print_result(func):

    def wrapper(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if type(result) == list:
                print("\n".join(map(str, result)))

        elif type(result) == dict:
                for i in result.keys():
                    print(i, '=', result[i])

        else:
            print(result)
        return func(*args, **kwargs)

    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]


test_1()
test_2()
test_3()
test_4()
```

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyth
on_fp/print_result.py
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.

`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).

Текст программы:

```
import time
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):
        print(time.time() - self.start_time)


@contextmanager
def cm_timer_2():
    start_time = time.time()

    try:
        yield

    finally:
        print(time.time() - start_time)


with cm_timer_1():
    time.sleep(5.5)

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

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyth
on_fp/cm_timer.py
5.504686594009399
5.504794597625732
```

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:

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


path = '/home/andrey/BKIT/lab_3-4/lab_python_fp/data_light.json'

with open(path) as f:
    data = json.load(f)


def f1(arg):
    return sorted(Unique(field(arg, 'job-name')).items)


@print_result
def f2(arg):
    return list(filter(lambda x: True if x.lower().startswith('программист')
else False, arg))


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


@print_result
def f4(arg):
    salaries = gen_random(len(arg), 100000, 200000)
    return [i + ', зарплата {} руб.'.format(j) for i, j in zip(arg,
salaries)]


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

Вывод:

```
andrey@andrey-HP-EliteBook-840-G5:~/BKIT/lab_1/rust$ /bin/python3 /home/andrey/BKIT/lab_3-4/lab_pyt
on_fp/process_data.py
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
программист
программист 1С
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
программист с опытом Python
программист 1С с опытом Python
```

```
f4
Программист с опытом Python, зарплата 177838 руб.
Программист / Senior Developer с опытом Python, зарплата 127766 руб.
Программист 1С с опытом Python, зарплата 122158 руб.
Программист C# с опытом Python, зарплата 193401 руб.
Программист C++ с опытом Python, зарплата 106924 руб.
Программист C++/C#/Java с опытом Python, зарплата 173958 руб.
Программист/ Junior Developer с опытом Python, зарплата 138960 руб.
Программист/ технический специалист с опытом Python, зарплата 165922 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 126309 руб.
программист с опытом Python, зарплата 150002 руб.
программист 1С с опытом Python, зарплата 100501 руб.
0.004831075668334961
```