Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»
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 ИУ5-32Б
Плюшко Дмитрий Андреевич

Проверил:
преподаватель каф. ИУ5

Подпись и дата:

Подпись и дата:

Москва, 2024 г.

Задача 1

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 количество 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
    result = []
    dict_count = 0
    for item in items:
        result.append(dict())
        for key in item.keys():
            if key in args and item[key] is not None:
                result[dict_count].update({key: item[key]})
        dict_count += 1
    return [x[args[0]] for x in result]


def test():
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print(field(goods, 'title'))

test()
```

Пример выполнения

```
['Ковер', 'Диван для отдыха']

Process finished with exit code 0
```

Задача 2

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

def gen_random(count, minValue, maxValue):
    a = [random.randint(minValue, maxValue) for i in
range(count)]
    return a

print(gen_random(5, 1, 3))
```

Пример выполнения

```
[1, 2, 3, 1, 3]

Process finished with exit code 0
```

Задача 3

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):
        if "ignore_case" in kwargs.keys():
            self.ignore_case = kwargs.get("ignore_case")
        else:
```

```
            self.ignore_case = False
        self.data = items
        self.arr = []
        self.number = -1

    def __next__(self):
        self.number += 1
        try:
            if isinstance(self.data[self.number], str):
                if self.ignore_case:
                    arg = self.data[self.number].lower()
                else:
                    arg = self.data[self.number]
                if arg not in self.arr:
                    self.arr.append(arg)
                    return arg
                else:
                    return self.__next__()
            elif self.data[self.number] not in self.arr:
                self.arr.append(self.data[self.number])
                return self.data[self.number]
            else:
                return self.__next__()
        except IndexError:
            raise StopIteration

    def __iter__(self):
        return self


def test():
    from gen_random import gen_random
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data2 = gen_random(10, 1, 3)
    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print([i for i in Unique(data1)])
    print([i for i in Unique(data2)])
    print([i for i in Unique(data3)])
    print([i for i in Unique(data3, ignore_case=True)])

test()
```

Пример выполнения

```
[1, 2]
[1, 2, 3]
['a', 'A', 'b', 'B']
['a', 'b']
```

Задача 4

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.

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

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

Пример выполнения

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

Задача 5

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(f'{func.__name__}')
        res = func(*args, **kwargs)
        if isinstance(res, dict):
            for i, j in res.items():
                print(f'{i} = {j}')
        elif isinstance(res, list):
            for i in res:
                print(f'{i}')
        else:
            print(f'{res}')
        return res
    return wrapper


def test():
    @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()
test()
```

Пример выполнения

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

Задача 6

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран.

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).

Текст программы

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


class cm_timer_1:
    def __init__(self):
        self.startTime = 0

    def __enter__(self):
        self.startTime = time.time()
        return self

    def __exit__(self, a, b, c):
```

```
        endTime = time.time() - self.startTime
        print(f"cm_timer_1: Time {endTime}s")


@contextmanager
def cm_timer_2():
    nowTime = time.time()
    try:
        yield
    finally:
        print(f'cm_timer_2: Time {time.time() - nowTime}s')


def test():
    with cm_timer_1():
        sleep(2.3)

    with cm_timer_2():
        sleep(2.2)

test()
```

Пример выполнения

```
cm_timer_1: Time 2.308208703994751s
cm_timer_2: Time 2.2006888389587402s

Process finished with exit code 0
```

Задача 7

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 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

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


@print_result
def f1(arg):
    return sorted(list(str(x) for x in Unique(field(arg,
'job-name'), ignore_case=True)))


@print_result
def f2(arg):
    return list(filter((lambda x: 'программист' in x), arg))
```

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


    @print_result
    def f4(arg):
        return list(f'{i}, зарплата {j} руб.'
                    for i, j in zip(arg, gen_random(len(arg),
100_000, 200_000)))

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

Пример выполнения

Первые строчки вывода функций

```
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
```

f2

1с программист

web-программист

веб - программист (php, js) / web разработчик

веб-программист

ведущий инженер-программист

ведущий программист

инженер - программист

инженер - программист асу тп

инженер-программист

инженер-программист (клинский филиал)

инженер-программист (орехово-зуевский филиал)

инженер-программист 1 категории

инженер-программист ккт

инженер-программист плис

f3

1с программист с опытом Python

web-программист с опытом Python

веб - программист (php, js) / web разработчик с опытом Python

веб-программист с опытом Python

ведущий инженер-программист с опытом Python

ведущий программист с опытом Python

инженер - программист с опытом Python

инженер - программист асу тп с опытом Python

инженер-программист с опытом Python

инженер-программист (клинский филиал) с опытом Python

инженер-программист (орехово-зуевский филиал) с опытом Python

инженер-программист 1 категории с опытом Python

инженер-программист ккт с опытом Python

инженер-программист плис с опытом Python

f4

1с программист с опытом Python, зарплата 184261 руб.

web-программист с опытом Python, зарплата 170724 руб.

веб - программист (php, js) / web разработчик с опытом Python, зарплата 128933 руб.

веб-программист с опытом Python, зарплата 154788 руб.

ведущий инженер-программист с опытом Python, зарплата 116901 руб.

ведущий программист с опытом Python, зарплата 185615 руб.

инженер - программист с опытом Python, зарплата 173944 руб.

инженер - программист асу тп с опытом Python, зарплата 148522 руб.

инженер-программист с опытом Python, зарплата 134178 руб.

инженер-программист (клинский филиал) с опытом Python, зарплата 117984 руб.

инженер-программист (орехово-зуевский филиал) с опытом Python, зарплата 112488 руб.

инженер-программист 1 категории с опытом Python, зарплата 146104 руб.

инженер-программист ккт с опытом Python, зарплата 136784 руб.

инженер-программист плис с опытом Python, зарплата 192022 руб.