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 лабораторным работам №3-4
«Функциональные возможности языка Python.»

Выполнил:
студент группы ИУ5-31Б
Ларкин Б. В.
Подпись и дата:

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

Москва, 2023 г.

Цель работы

Изучение возможностей функционального программирования в языке Python.

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.

Код программы:

```
def field(items, *args):
    assert len(args) > 0, 'Empty parameter sweep'
    for item in items:
        if len(args)==1:
            yield item[args[0]]
        else:
            line = {}
            for arg in args:
                if arg in item:
                    line[arg] = item[arg]
            yield line

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

if __name__ == "__main__":
    main()
```

Результат выполнения:

```
== RESTART: C:\Прога\ProgrammingProjects\BKIT2023\Lab3 Functional Py\field.py ==
[{'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.

Код программы:

```
from random import randint
def gen_random(num_count, begin, end):
    assert num_count != 0, 'Empty generator'
    for iteration in range(num_count):
        yield randint(begin,end)


def main():
    print(list(gen_random(5,1,3)))
    return

if __name__ == "__main__":
    main()
```

Результат выполнения:

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

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.

Код программы:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        if 'ignore_case' in kwargs:
            ignore_case = kwargs['ignore_case']
        else:
            ignore_case = False
        self.collection = []
        self.cursor = 0
        for item in items:
            if ignore_case:
                if (item not in self.collection):
                    self.collection.append(item)
            else:
                for x in self.collection:
                    if (item.lower() == x.lower()):
                        break
                else:
                    self.collection.append(item)
        pass

    def __next__(self):
        self.cursor+=1
        if self.cursor <= len(self.collection):
            return self.collection[self.cursor-1]
        raise StopIteration

    def __iter__(self):
        return self

def main():
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(list(Unique(data, ignore_case = True)))
    return

if __name__ == "__main__":
    main()
```

Результат выполнения:

```
['a', 'A', 'b', '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.

Код программы:

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

def main():
    result = sorted(data)
    print(result)

    result_with_lambda = sorted(data, key=lambda item: item)
    print(result_with_lambda)

if __name__ == '__main__':
    main()
```

Результат выполнения:

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

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.

Код программы:

```
def print_result(func):
    def format(*arg):
        print('\n'+func.__name__, ':')
```

```
        output = func(*arg)
        if type(output) is list:
            for item in output:
                print(item)
        elif type(output) is dict:
            for key, value in output.items():
                print(str(key) + ' = ' + str(value))
        else:
            print(output)
        return output

    return format

@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:

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).

Код программы:

```
from time import *
from contextlib import contextmanager

class cm_timer_1():
    def __enter__(self):
        self.start = perf_counter()
        return self

    def __exit__(self, type, value, traceback):
        self.time = perf_counter() - self.start
        self.read = f'CMT1. Time elapsed: {self.time:.3f} seconds'
        print(self.read)

@contextmanager
def cm_timer_2():
    start = perf_counter()
    yield lambda: perf_counter() - start
    print(f'CMT2. Time elapsed: {perf_counter() - start:.3f} seconds')

def main():
    with cm_timer_1():
        sleep(5.5)

    with cm_timer_2():
        sleep(5.5)

if __name__ == "__main__":
    main()
```

Результат выполнения:

```
CMT1. Time elapsed: 5.507 seconds
CMT2. Time elapsed: 5.501 seconds
```

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.

Код программы:

```
import json
from pathlib import *
from sys import *
from cm_timer import *
from field import *
from gen_random import *
from print_result import *
from sort import *
from unique import *

with open('data_light.json', 'r', encoding='utf-8') as f:
```

```
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(list(Unique([s['job-name'] for s in arg], ignore_case = True)))


@print_result
def f2(arg):
    return list(filter(lambda s: (s.lower().startswith("программист")), arg))


@print_result
def f3(arg):
    return [(s + " с опытом Python") for s in arg]


@print_result
def f4(arg):
    return list([(s +', зарплата ' + str(list(gen_random(1,100000,200000))[0])) for s in
arg])

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

Результат выполнения:

```
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
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
...
электросварщик
электросварщик на автоматических и полуавтоматических машинах
энтомолог
юрисконсульт
юрисконсульт 2 категории
юрист
```

```
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 189628
Программист / Senior Developer с опытом Python, зарплата 141023
Программист 1С с опытом Python, зарплата 122056
Программист C# с опытом Python, зарплата 147274
Программист C++ с опытом Python, зарплата 149055
Программист C++/C#/Java с опытом Python, зарплата 188755
Программист/ Junior Developer с опытом Python, зарплата 167888
Программист/ технический специалист с опытом Python, зарплата 182412
Программистр-разработчик  информационных систем с опытом Python, зарплата 121202
программист с опытом Python, зарплата 110793
программист 1С с опытом Python, зарплата 123178
CMT1. Time elapsed: 26.153 seconds
```