**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: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Ларкин Б. В. |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | 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:



**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:



**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:



**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:



**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\_\_':

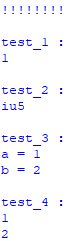
    print('!!!!!!!!')

    test\_1()

    test\_2()

    test\_3()

    test\_4()

Результат выполнения:

**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:



**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.

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

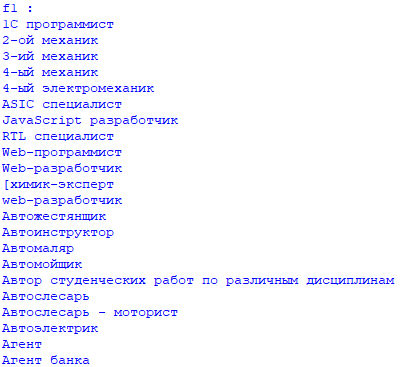
def f4(arg):

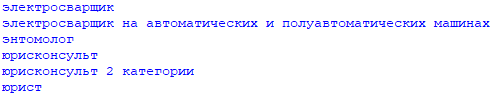
    return list([(s +', зарплата ' + str(list(gen\_random(1,100000,200000))[0])) for s in arg])

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

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

Результат выполнения:

…

