**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 ЛР№4

Выполнил: Проверил:

студент группы ИУ5-34Б преподаватель каф. ИУ5

Сергеев Максим Гапанюк Ю.Е. Подпись и дата: Подпись и дата:

Москва, 2023 г.

**Задача 1:**

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'}

1. В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количество аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
3. 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 dic in items:  
 for key in dic.keys():  
 if dic[key] is None:  
 continue  
 if key in args:  
 yield dic[key]  
 else:  
 for dic in items:  
 yield {key: value for (key, value) in dic.items() if key in args and value is not None}

**Результат работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**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.

Пример:

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

**Текст программы:**

from random import randint  
  
  
def gen\_random(num\_count, begin, end):  
 return [randint(begin, end) for i in range(num\_count)]

**Результат работы программы:**

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

**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):  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
 self.items = list(set([str(i) for i in items]))  
 self.index = 0  
 if self.ignore\_case:  
 tmp = []  
 for i in self.items:  
 if i.lower() not in [j.lower() for j in tmp]:  
 tmp.append(i)  
 self.items = tmp  
  
 def \_\_next\_\_(self):  
 try:  
 item = self.items[self.index]  
 except IndexError:  
 raise StopIteration()  
 self.index += 1  
 return item  
  
 def \_\_iter\_\_(self):  
 return self

**Результат работы программы:**

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

**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. Сортировку необходимо осуществлять с помощью функции sorted. Пример:

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

Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

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: abs(x), reverse=True)  
 print(result\_with\_lambda)

**Результат работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**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):  
 a = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
 if isinstance(a, list):  
 for i in a:  
 print(i)  
 elif isinstance(a, dict):  
 for key, value in a.items():  
 print(key, ' = ', value)  
 else:  
 print(a)  
 return a  
 return wrapper

**Результат работы программы:**

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание**Задача 6:**

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).

**Текст программы:**

from time import time  
from contextlib import contextmanager  
  
  
class cm\_timer1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time()  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print('time: ', time() - self.start\_time, 's', sep='')  
  
  
@contextmanager  
def cm\_timer2():  
 start\_time = time()  
 try:  
 yield  
 finally:  
 print('time: ', time() - start\_time, 's', sep='')

**Результат работы программы:**

**Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание**

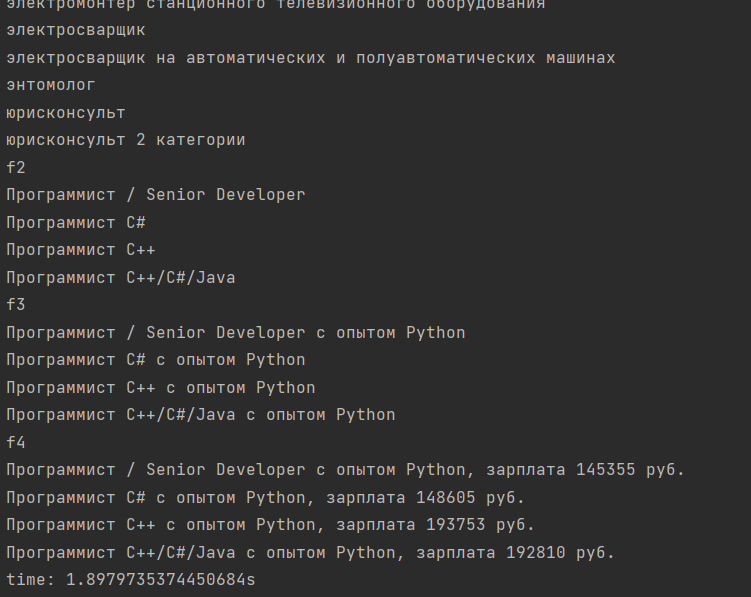
**Задача 7:**

* 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.

**Текст программы:**

import json  
from print\_result import print\_result  
from cm\_timer import cm\_timer1  
from gen\_random import gen\_random  
from unique import Unique  
from field import field  
  
path = 'D:\GitRepos\Proga3sem\Python lab3-4\data\_light.json'  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return sorted([x for x in Unique(field(arg, 'job-name'), ignore\_case=True)])  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: 'Программист' == x.split()[0], arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 return [str(i) + ', зарплата ' + str(j) + ' руб.' for i, j in zip(arg, gen\_random(len(arg), 100000, 200000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer1():  
 f4(f3(f2(f1(data))))

**Результат работы программы:**

****