**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования »

Отчет по лабораторной работе №2

«Функциональные возможности языка Python.»

|  |  |
| --- | --- |
| Выполнил: |  |
| студент группы  РТ5-31Б |  |
| Ермаков И.А. |  |
|  |  |

Москва, 2024 г

**Описание 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.

### **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. Пример:

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

### **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.

### **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. 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.

### **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.

### **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.

### **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.

**Текст программы:**

**field.py**

def field(items, \*args):

assert len(args) > 0, "Нужно передать хотя бы один аргумент."

for item in items:

if len(args) == 1:

value = item.get(args[0])

if value is not None:

yield value

else:

result = {key: item.get(key) for key in args if item.get(key) is not None}

if result:

yield result

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

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

{'title': 'Диван для отдыха', 'color': 'black'}

]

print(list(field(goods, 'title')))

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

**gen\_random.py**

import random

def gen\_random(num\_count, begin, end):

for \_ in range(num\_count):

yield random.randint(begin, end)

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

print(list(gen\_random(5, 1, 3)))

**unique.py**

class Unique:

def \_\_init\_\_(self, items, \*\*kwargs):

self.ignore\_case = kwargs.get('ignore\_case', False)

self.seen = set()

self.items = iter(items)

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

while True:

item = next(self.items)

check\_item = item.lower() if self.ignore\_case and isinstance(item, str) else item

if check\_item not in self.seen:

self.seen.add(check\_item)

return item

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

data = [1, 1, 1, 2, 2, 3]

print(list(Unique(data)))

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

print(list(Unique(data)))

print(list(Unique(data, ignore\_case=True)))

**sort.py**

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

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

# С использованием lambda-функции

result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

print(result\_with\_lambda)

# Без использования lambda-функции

def abs\_key(x):

return abs(x)

result = sorted(data, key=abs\_key, reverse=True)

print(result)

**print\_result.py**

def print\_result(func):

def wrapper(\*args, \*\*kwargs):

result = func(\*args, \*\*kwargs)

print(func.\_\_name\_\_)

if isinstance(result, list):

print(\*result, sep="\n")

elif isinstance(result, dict):

for key, value in result.items():

print(f"{key} = {value}")

else:

print(result)

return result

return wrapper

@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()

**cm\_timer.py**

import time

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_enter\_\_(self):

self.start\_time = time.time()

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

print(f"time: {time.time() - self.start\_time:.1f}")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

try:

yield

finally:

print(f"time: {time.time() - start\_time:.1f}")

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

with cm\_timer\_1():

time.sleep(1.5)

with cm\_timer\_2():

time.sleep(2.5)

**process\_data.py**

import json

import sys

import random

from contextlib import contextmanager

from time import time

# Декоратор для вывода результата

class print\_result:

def \_\_init\_\_(self, func):

self.func = func

def \_\_call\_\_(self, \*args, \*\*kwargs):

result = self.func(\*args, \*\*kwargs)

print(result)

return result

# Контекстный менеджер для измерения времени

@contextmanager

def cm\_timer\_1():

start\_time = time()

yield

print(f"Time elapsed: {time() - start\_time:.2f} seconds")

# Определяем путь к файлу

path = 'lab4/data\_light.json'

# Загрузка данных

with open(path, encoding='utf-8') as f:

data = json.load(f)

# Реализация функций

@print\_result

def f1(arg):

return sorted(set(job["job-name"].strip().lower() for job in arg), key=str.casefold)

@print\_result

def f2(arg):

return list(filter(lambda x: x.lower().startswith("программист"), arg))

@print\_result

def f3(arg):

return list(map(lambda x: f"{x} с опытом Python", arg))

@print\_result

def f4(arg):

salaries = [random.randint(100000, 200000) for \_ in range(len(arg))]

return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg, salaries)]

# Основной блок

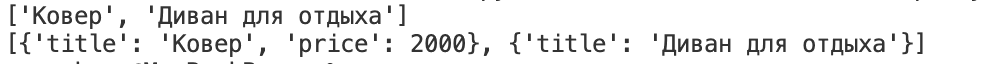
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

f4(f3(f2(f1(data))))

**Пример выполнения программ:**

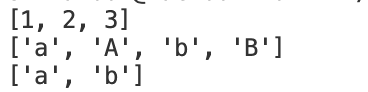
**Задача 1 field**



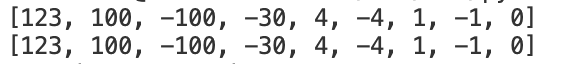
**Задача 2 gen\_random**



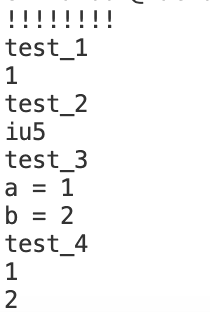
**Задача 3 unique**

****

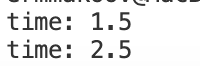
**Задача 4 sort**

****

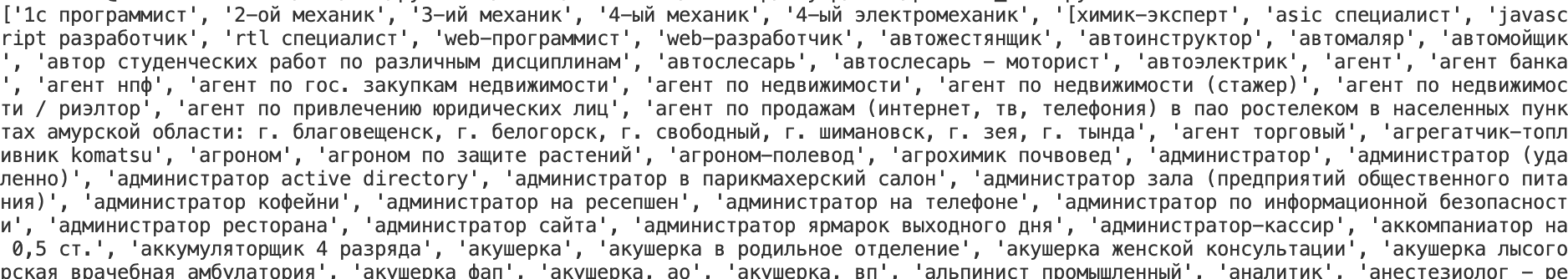
**Задача 5 print\_result**

****

**Задача 6 cm\_timer**

****

**Задача 7 process\_data**

****