**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 №3

«Функциональные возможности языка Python»

Вариант №11

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б: |  | преподаватель каф. ИУ5 |
| Карпенко Д.А. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2023 г.

**Постановка задачи:**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего 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.

**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.

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.

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).

**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**

1. **field.py**

def field(items**,** \*args):  
 assert len(args) > **0** for item in items:  
 if len(args) == **1**:  
 var = item.get(args[**0**])  
 if var is not None:  
 yield var  
 else:  
 new\_dic = {}  
 all\_none = True  
 for key in args:  
 var = item.get(key)  
 if var is not None:  
 new\_dic[key] = var  
 all\_none = False  
 if not all\_none:  
 yield var  
goods = [  
 {'title': 'Ковер'**,** 'price': **2000,** 'color': 'green'}**,** {'title': 'Диван для отдыха'**,** 'color': 'black'}  
]  
print(list(field(goods**,** 'title')))  
print(list(field(goods**,** 'title'**,** 'price')))



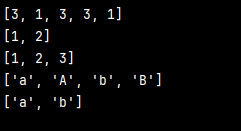
1. **gen\_random.py**

import random  
  
  
def gen\_random(count**,** start**,** finish):  
 for \_ in range(count):  
 yield random.randint(start**,** finish)  
  
  
random\_numbers = list(gen\_random(**5, 1, 3**))  
print(random\_numbers)  
  
pass



1. **unique.py**

from gen\_random import gen\_random  
  
  
class Unique:  
 def \_\_init\_\_(self**,** items**,** \*\*kwargs):  
 self.items = iter(items)  
 self.unique\_set = set()  
 self.ignore\_case = kwargs.get('ignore\_case'**,** False)  
  
 def \_\_next\_\_(self):  
 while True:  
 item = next(self.items)  
 key = item.lower() if self.ignore\_case and isinstance(item**,** str) else item  
 if key not in self.unique\_set:  
 self.unique\_set.add(key)  
 return item  
 def \_\_iter\_\_(self):  
 return self  
data1 = [**1, 1, 1, 1, 1, 2, 2, 2, 2, 2**]  
unique\_iter1 = Unique(data1)  
print(list(unique\_iter1)) # [1, 2]  
data2 = gen\_random(**10, 1, 3**)  
unique\_iter2 = Unique(data2)  
print(list(unique\_iter2)) # [1, 2, 3]  
data3 = ['a'**,** 'A'**,** 'b'**,** 'B'**,** 'a'**,** 'A'**,** 'b'**,** 'B']  
unique\_iter3 = Unique(data3)  
print(list(unique\_iter3)) # ['a', 'A', 'b', 'B']  
data4 = ['a'**,** 'A'**,** 'b'**,** 'B'**,** 'a'**,** 'A'**,** 'b'**,** 'B']  
unique\_iter4 = Unique(data4**,** ignore\_case=True)  
print(list(unique\_iter4)) # ['a', 'b']



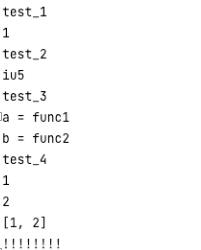
1. **sort.py**

data = [**4,** -**30, 100,** -**100, 123, 1, 0,** -**1,** -**4**]  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Решение без lambda-функции  
 def key\_function(x):  
 return abs(x)  
  
  
 result = sorted(data**,** key=key\_function**,** reverse=True)  
 print(result)  
 # Решение с lambda-функцией  
 result\_with\_lambda = sorted(data**,** key=lambda x: abs(x)**,** reverse=True)  
 print(result\_with\_lambda)



1. **print\_result.py**

def print\_result(func):  
 def write(\*args**,** \*\*kwargs):  
 result = func(\*args**,** \*\*kwargs)  
 print(func.\_\_name\_\_)  
 if isinstance(result**,** list):  
 for item in result:  
 print(item)  
 if isinstance(result**,** dict):  
 for key**,** var in result.items():  
 print(f'{key} = func{var}')  
 else:  
 print(result)  
 return result  
 return write()  
  
  
@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()



1. **cm\_timer.py**

import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
 return self  
 def \_\_exit\_\_(self**,** exc\_type**,** exc\_value**,** traceback):  
 self.end\_time = time.time()  
 elapsed\_time = self.end\_time - self.start\_time  
 print(f"time: {elapsed\_time}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
 print(f"time: {elapsed\_time}")  
  
  
with cm\_timer\_1():  
 time.sleep(**5.5**)  
with cm\_timer\_2():  
 time.sleep(**5.5**)



1. process\_data.py

import json  
import time  
from contextlib import contextmanager  
  
  
@contextmanager  
def cm\_timer\_1():  
 start\_time = time.time()  
 yield  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
 print(f"time: {elapsed\_time}")  
  
  
def print\_result(func):  
 def write(\*args**,** \*\*kwargs):  
 result = func(\*args**,** \*\*kwargs)  
 if isinstance(result**,** list):  
 for item in result:  
 print(item)  
 elif isinstance(result**,** dict):  
 for key**,** value in result.items():  
 print(f"{key} = {value}")  
 else:  
 print(result)  
 return result  
 return write  
  
  
path = "data\_light.json"  
with open(path) as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return (sorted(set(job['job-name'].lower() for job in arg if isinstance(job**,** dict))**,** key=str.lower)) if isinstance(  
 arg**,** list) else []  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda job: job.startswith('программист')**,** arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda job: job + ', с опытом Python'**,** arg))  
  
  
@print\_result  
def f4(arg):  
 return [job + f", зарплата {salary} руб." for job**,** salary in zip(arg**,** range(**100000, 200001, 1000**))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

