**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет приложений»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-52Б |  | преподаватель каф. ИУ5 |
| Васильченко Дарья |  | Гапанюк Ю.Е. |
|  |  |  |

Москва, 2021 г.

**Описание задания:**

Задание лабораторной работы состоит из решения нескольких задач.

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

**Текст программы**

def field(items, \*args):  
 assert len(args) > 0  
 result = []  
 if len(args) > 1:  
 i = -1  
 for d in items:  
 result.append({})  
 i += 1  
 for arg in args:  
 val = d.get(arg)  
 if val:  
 result[i].update({arg: val})  
 else:  
 for arg in args:  
 for d in items:  
 val = d.get(arg)  
 if val:  
 result.append(val)  
 return result  
  
  
  
def main():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]  
 print(\*field(goods, 'title'), sep=", ")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Экранная форма с примером выполнения программы:**



**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.

**Текст программы**

import random  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield random.randint(begin, end)  
  
  
def main():  
 for i in gen\_random(5, 1, 3):  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Экранная форма с примером выполнения программы:**



**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.

**Текст программы**

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.alphabet = set()  
 if isinstance(items, list):  
 self.items = iter(items)  
 else:  
 self.items = items  
 b = kwargs.get('ignore\_case')  
 if b:  
 self.ignore\_case = bool(b)  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 nowElem = next(self.items)  
 if self.ignore\_case & isinstance(nowElem, str):  
 nowElemLow = nowElem.lower()  
 if nowElemLow not in self.alphabet:  
 self.alphabet.add(nowElemLow)  
 return nowElem  
 else:  
 if nowElem not in self.alphabet:  
 self.alphabet.add(nowElem)  
 return nowElem  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
def main():  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 iter1 = Unique(data, ignore\_case=True)  
 for i in iter1:  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Экранная форма с примером выполнения программы:**



**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.

**Текст программы**

import math  
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, key=math.fabs, reverse=True)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda x: math.fabs(x), reverse=True)  
 print(result\_with\_lambda)

**Экранная форма с примером выполнения программы:**



**Задача 5 (файл print\_result.py)**

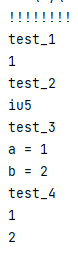
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

**Текст программы**

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 print(str(func).split(" ")[1])  
 result = func(\*args, \*\*kwargs)  
 if isinstance(result, list):  
 for i in result:  
 print(i)  
 elif isinstance(result, dict):  
 for key, val in result.items():  
 print(str(key) + " = " + str(val))  
 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()

**Экранная форма с примером выполнения программы:**



**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).

**Текст программы**

from contextlib import contextmanager  
import time  
  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.start\_time = 0  
  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print("time: " + str(time.time() - self.start\_time))  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 print("time: " + str(time.time() - start\_time))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 time.sleep(1.5)  
   
 with cm\_timer\_2():  
 time.sleep(2.5)

**Экранная форма с примером выполнения программы:**



**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/iu5team/iu5web-fall-2021/tree/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 unique import Unique  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from field import field  
from gen\_random import gen\_random  
  
path = r"C:\My\3 курс\5 семестр\РИП\Лабы\lab3\data\_light.json"  
  
with open(path, encoding="utf8") as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return sorted(Unique(field(arg, 'job-name'), ignore\_case=True))  
  
  
@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: x + " с опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 for job, sal in zip(range(len(arg)), gen\_random(len(arg), 100\_000, 200\_000)):  
 arg[job] += ", зарплата " + str(sal) + " руб."  
 return arg  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Экранная форма с примером выполнения программы:**

