**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 |
| Перова Анна |  | 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 раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

**Задача 1 (файл field.py)**

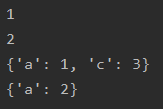
Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

* В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное 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  
 if len(args) == 1:  
 for item in items:  
 result = item.get(args[0])  
 if result is not None:  
 yield result  
 else:  
 for item in items:  
 result = {key: item.get(key) for key in args if item.get(key) is not None}  
 if result != {}:  
 yield result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 lst = [  
 {'a': 1, 'b': 2, 'c': 3},  
 {'a': 2, 'c': None, 'd': -1},  
 {'a': None, 'c': None, 'd': 5}  
 ]  
  
 for i in field(lst, 'a'):  
 print(i)  
 for i in field(lst, 'a', 'c'):  
 print(i)

**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 \_ in range(num\_count):  
 yield random.randint(begin, end)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in gen\_random(5, 0, 2):  
 print(i)

**Экранная форма с примером выполнения программы:**



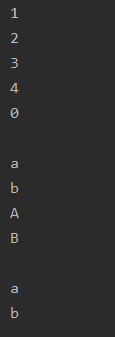
**Задача 3 (файл unique.py)**

* 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:  
 def \_\_init\_\_(self, data, \*\*kwargs):  
 self.data = data  
 self.type = type(data)  
 self.index = 0  
 self.unique\_values = set()  
 self.ignore\_case = kwargs['ignore\_case'] if kwargs.get('ignore\_case') is not None else False  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def check\_elem(self, elem):  
 if self.ignore\_case:  
 elem = elem.lower()  
 if elem not in self.unique\_values:  
 self.unique\_values.add(elem)  
 return True  
 return False  
  
 def \_\_next\_\_(self):  
 if self.type == list:  
 while self.index < len(self.data):  
 elem = self.data[self.index]  
 self.index += 1  
 if self.check\_elem(elem):  
 return elem  
 else:  
 for elem in self.data:  
 if self.check\_elem(elem):  
 return elem  
 raise StopIteration  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data1 = [1, 1, 2, 2, 3, 4, 1, 0]  
 data2 = ['a', 'b', 'A', 'a', 'B', 'b']  
 data3 = (i for i in data2)  
 for i in Unique(data1):  
 print(i)  
 print()  
 for i in Unique(data2):  
 print(i)  
 print()  
 for i in Unique(data3, ignore\_case=True):  
 print(i)

**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**

data = [4, -30, 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)

**Экранная форма с примером выполнения программы:**



**Задача 5 (файл print\_result.py)**

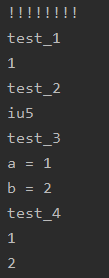
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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):  
 result = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
 if type(result) == list:  
 for elem in result:  
 print(elem)  
 elif type(result) == dict:  
 for (key, val) in result.items():  
 print(f"{key} = {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**

import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
 return 1  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 print('Execution time: {} s'.format(time.time() - self.start\_time))  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield 1  
 print('Execution time: {} s'.format(time.time() - start\_time))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 time.sleep(2)  
  
 with cm\_timer\_2():  
 time.sleep(2)

**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 lab\_python\_fp.field import field  
from lab\_python\_fp.gen\_random import gen\_random  
from lab\_python\_fp.unique import Unique  
from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.cm\_timer import cm\_timer\_1  
  
  
path = "../data\_light.json"  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return sorted([unique\_prof for unique\_prof in Unique([prof for prof in field(arg, 'job-name')], ignore\_case=True)])  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: x.startswith('Программист'), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + " c опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 return [f"{prof}, зарплата {salary} руб." for prof, salary in zip(arg, gen\_random(len(arg), 100\_000, 200\_000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Экранная форма с примером выполнения программы:**

