**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «БКИТ»

Отчет по лабораторной работе №3-4

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35Б |  | преподаватель каф. ИУ5 |
| Нгуен Зуи Лам |  | Нардид А.Н. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2022 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.

1) 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.

2) 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

3) 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 boolпараметр 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.

Пример:

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

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(10, 1, 3)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a,

b.

4) 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.

5) 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.

6) Задача 6 файл “cm\_timmer.py”:

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).

7) 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 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**

**Файлы пакета lab\_python\_fp:**

**Задача 1.**

Field.py

def field(items, \*args):  
 assert len(args) > 0  
 if len(args) == 1:  
 b = [item[i] for item in items for i in item if i == args[0] and item[i] is not None]  
 return b  
 else:  
 b = [{i: item[i]} for item in items for i in item for arg in args if arg == i and item[i] is not None]  
 return b

**Задача 2.**

Gen\_random.py

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

**Задача 3.**

Unique.py

class Unique(object):  
 def \_\_init\_\_(self, items, ignore\_case, \*\*kwargs):  
 self.data = items  
 self.ignore\_case = ignore\_case  
 self.index = 0  
 self.data2 = set()  
  
 def \_\_next\_\_(self):  
 if not self.ignore\_case:  
 for count, el in enumerate(self.data):  
 if type(el) is str:  
 self.data[count] = el.lower()  
  
 while True:  
 if self.index >= len(self.data):  
 raise StopIteration  
 else:  
 current = self.data[self.index]  
 self.index += 1  
 if current not in self.data2:  
 self.data2.add(current)  
 return current  
  
 def \_\_iter\_\_(self):  
 return

**Задача 4.**

Sort.py

def sort(data):  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda a: abs(a), reverse=True)  
 print(result\_with\_lambda)

**Задача 5.**

Print\_result.py

def print\_result(func):  
 def wrapper(lst=[], \*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
  
 if len(lst) == 0:  
 result = func(\*args, \*\*kwargs)  
 else:  
 result = func(lst, \*args, \*\*kwargs)  
  
 if type(result) is dict:  
 for key, el in result.items():  
 print(f'{key} = {el}')  
  
 elif type(result) is list:  
 print('\n'.join(map(str, result)))  
  
 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]

**Задача 6.**

Cm\_timer.py

from time import time  
from contextlib import contextmanager  
  
@contextmanager  
def cm\_timer\_1():  
 start = time()  
 yield None  
 finish = time()  
 print("Время работы: {}".format(finish - start))  
  
class cm\_timer\_2:  
 def \_\_int\_\_(self):  
 self.start = 0  
 self.finish = 0  
  
 def \_\_enter\_\_(self):  
 self.start = time()  
  
 def \_\_exit\_\_(self, ex\_type, ex\_value, ex\_traceback):  
 self.finish = time()  
 print("Время работы: {}".format(self.finish - self.start))

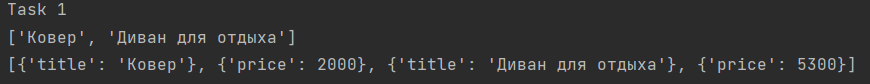
**Задача 7.**

Process\_data.py

from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.gen\_random import gen\_random  
  
@print\_result  
def f1(arg) -> list:  
 return sorted(list(set([i['job-name'] for i in arg])), key=lambda x: x.lower())  
  
@print\_result  
def f2(arg) -> list:  
 return list(filter(lambda s: (s.split())[0].lower() == 'программист', arg))  
  
@print\_result  
def f3(arg) -> list:  
 return list(map(lambda lst: lst + ' с опытом Python', arg))  
  
@print\_result  
def f4(arg) -> list:  
 return list(zip(arg, ['зарплата ' + str(i) + ' руб.' for i in gen\_random(len(arg), 100000, 200000)]))

**Результат выполнения.**

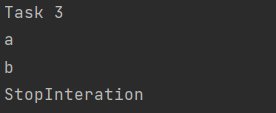
**Задача 1.**



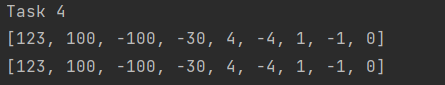
**Задача 2.**



**Задача 3.**



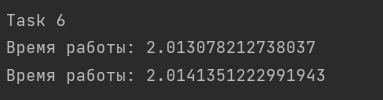
**Задача 4.**



**Задача 5.**



**Задача 6.**



**Задача 7.**

Часть выполнения 7 задачи:

