Московский государственный технический университет имени Н.Э.Баумана

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет РТ Радиотехнический

Кафедра ИУ5 Системы обработки информации и управления

**Отчет по лабораторной работе № 3 по курсу**

**Базовые компоненты интернет-технологий**

#### "Функциональные возможности языка Python"

|  |  |  |
| --- | --- | --- |
| Исполнитель |  |  |
| студент группы РТ5-31б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Титов А.Д. |
|  |  | “\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2021 г. |
|  |  |  |
| Проверил |  |  |
| Доцент кафедры ИУ5 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гапанюк Ю.Е. |
|  |  | “\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2021 г. |

Москва -  2021

**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 раcполагаться 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**.

Пример:

goods = [

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

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

]

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'

field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

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.

**Текст программы**

goods = [

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

{'title': 'Диван для отдыха', 'price': 5000, 'color': 'black'}

]

def field(items, \*args):

assert len(args) > 0

result = []

if len(args) == 1:

for item in items:

if item[args[0]]:

result.append(item[args[0]])

else:

for item in items:

d = {}

for arg in args:

if item[arg]:

d[arg] = item[arg]

result.append(str(d))

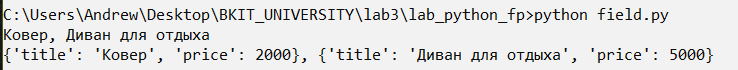
return ', '.join(result)

if \_\_name\_\_ == "\_\_main\_\_":

print(field(goods, 'title'))

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

**Примеры 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.**

Пример:

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

**Текст программы**

import random

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

result = []

for \_ in range(num\_count):

result.append(random.randint(begin, end))

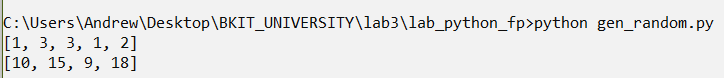
return result

if \_\_name\_\_ == "\_\_main\_\_":

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

print(gen\_random(4, 9, 20))

**Примеры 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.

Пример:

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

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

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

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.

**Текст программы**

class Unique(object):

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

self.items = items # массив элементов

try:

self.ignore\_case = kwargs["ignore\_case"]

except:

self.ignore\_case = False

self.array = []

for item in items:

item = (self.ignore\_case and str(item).lower()) or item # переводим элементы в нижний регистр, если стоит флаг ignore\_case

try: # проверяем наличие элемента в массиве

self.array.index(item)

except:

self.array.append(item)

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

self.index += 1

return self.array[(self.index - 1) % len(self.array)] # возвращаем следующий элемент

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

self.index = 0

return self

if \_\_name\_\_ == "\_\_main\_\_":

a = Unique(["a", "b", "A", "bb", "BB", "b", "c"], ignore\_case = True)

a.\_\_iter\_\_()

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_(), end=', ')

print(a.\_\_next\_\_())

**Примеры 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. 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:

* С использованием lambda-функции.
* Без использования lambda-функции.

**Текст программы**

import operator

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

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

# скорее всего можно проще

# создается кортеж (искомый элемент, его модуль), сортируется список из кортежей по второму элементу кортежа и создается список из первых элементов кортежа

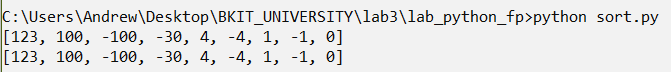
result = list(map(operator.itemgetter(0), sorted([(x, abs(x)) for x in data], key = operator.itemgetter(1), reverse = True)))

print(result)

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

print(result\_with\_lambda)

**Примеры 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(function):

def wrapper():

print(function.\_\_name\_\_)

if type(function()) == type([]):

for item in function():

print(item)

elif type(function()) == type({}):

for key in function().keys():

print("{} = {}".format(key, function()[key]))

else:

print(function())

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\_\_':

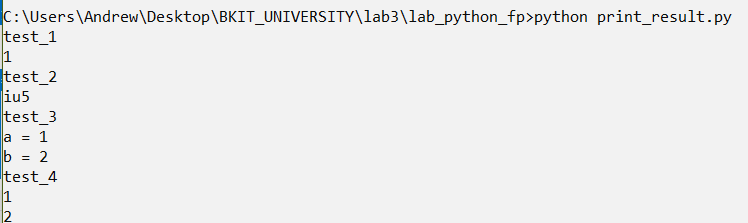
test\_1()

test\_2()

test\_3()

test\_4()

**Примеры 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.**

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).

**Текст программы**

from time import \*

from contextlib import contextmanager

class cm\_timer\_1:

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

pass

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

self.startTime = time()

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

print("time {}".format(time() - self.startTime))

@contextmanager

def cm\_timer\_2():

try:

startTime = time()

yield startTime

finally:

print("time {}".format(time() - startTime))

if \_\_name\_\_ == "\_\_main\_\_":

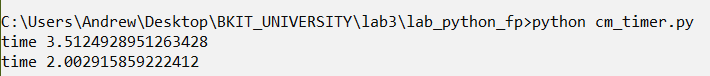
with cm\_timer\_1():

sleep(3.5)

with cm\_timer\_2():

sleep(2)

**Примеры 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 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 cm\_timer import cm\_timer\_1

from unique import Unique

from gen\_random import gen\_random

path = "data\_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

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

data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`

# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку

# В реализации функции f4 может быть до 3 строк

def print\_result(function):

def wrapper(arg):

print(function(arg))

return function(arg)

return wrapper

@print\_result

def f1(arg):

return sorted(Unique([item["job-name"] for item in arg], ignore\_case = True).array)

@print\_result

def f2(arg):

return list(filter(lambda x: x.split()[0] == "программист", arg))

@print\_result

def f3(arg):

return list(map(lambda x: "{} c опытом Python".format(x), arg))

@print\_result

def f4(arg):

zipped = list(zip(arg, gen\_random(len(arg), 100000, 200000)))

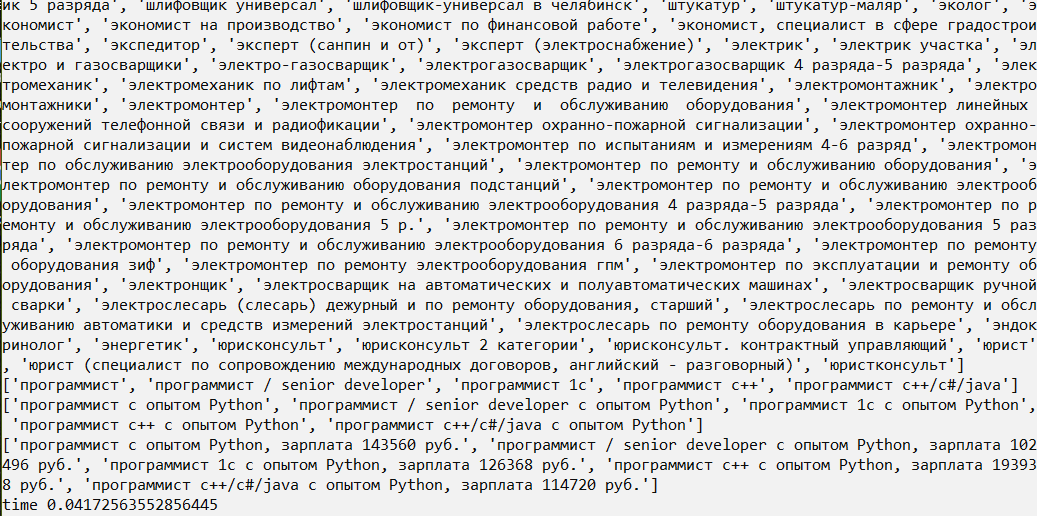
return list(map(lambda x: "{}, зарплата {} руб.".format(x[0], x[1]), zipped))

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

with cm\_timer\_1():

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

**Примеры выполнения программы**

****