**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №5

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Сомов Кирилл |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2023 г.

Задание

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.

Задание 1

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.

Шаблон для реализации генератора:

# Пример:

# goods = [

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

# {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}

def field(items, \*args):

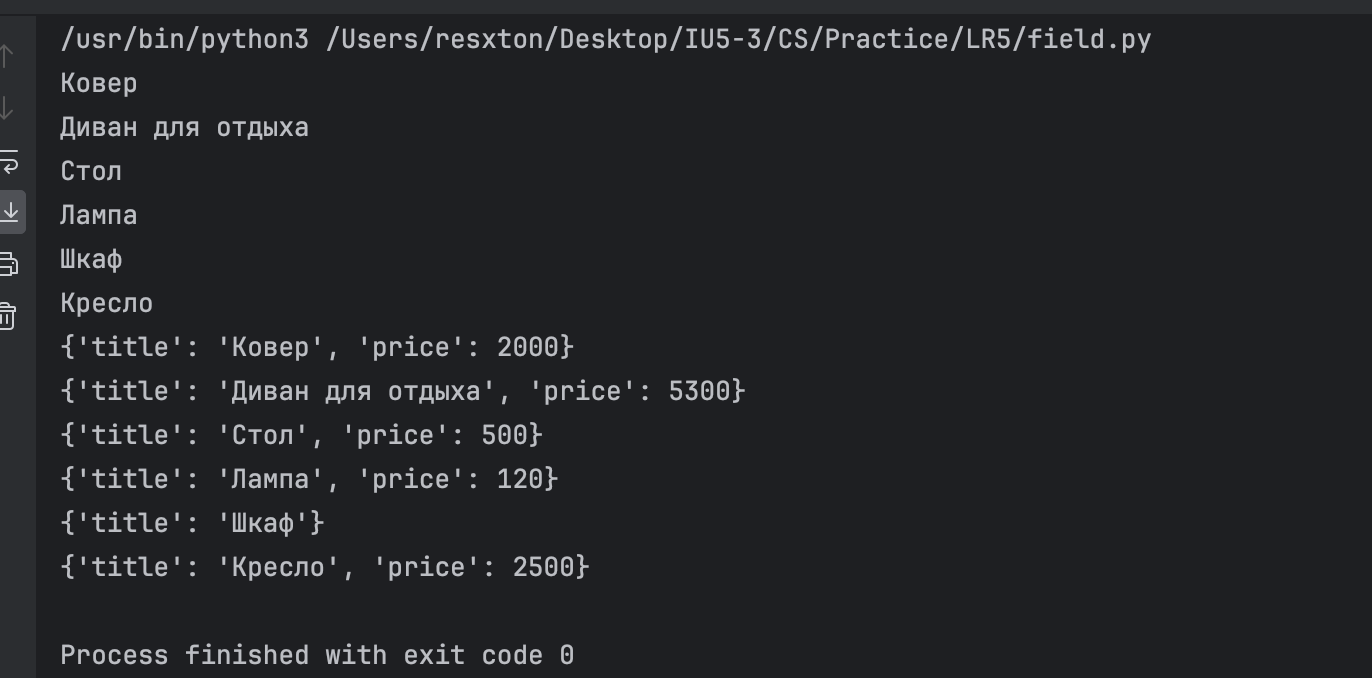
assert len(args) > 0

# Необходимо реализовать генератор

Код программы 1

def field(items, \*args):  
 assert len(args) > 0  
  
 if len(args) == 1:  
 # Если передано только одно поле, выдаем значения этого поля последовательно  
 for item in items:  
 field\_value = item.get(args[0])  
 if field\_value is not None:  
 yield field\_value  
 else:  
 # Если передано несколько полей, выдаем словари, содержащие эти поля  
 for item in items:  
 # comprehesion  
 filtered\_item = {field: item.get(field) for field in args if item.get(field) is not None}  
 if filtered\_item:  
 yield filtered\_item  
  
  
def main():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},  
 {'title': 'Стол', 'price': 500, 'color': 'brown'},  
 {'title': 'Лампа', 'price': 120, 'color': 'white'},  
 {'title': 'Шкаф', 'color': 'red'},  
 {'title': 'Кресло', 'price': 2500, 'color': 'blue'}  
 ]  
  
 for value in field(goods, 'title'):  
 print(value)  
  
 for dictionary in field(goods, 'title', 'price'):  
 print(dictionary)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод 1



Задание 2

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

Шаблон для реализации генератора:

# Пример:

# gen\_random(5, 1, 3) должен выдать выдать 5 случайных чисел

# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

# Hint: типовая реализация занимает 2 строки

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

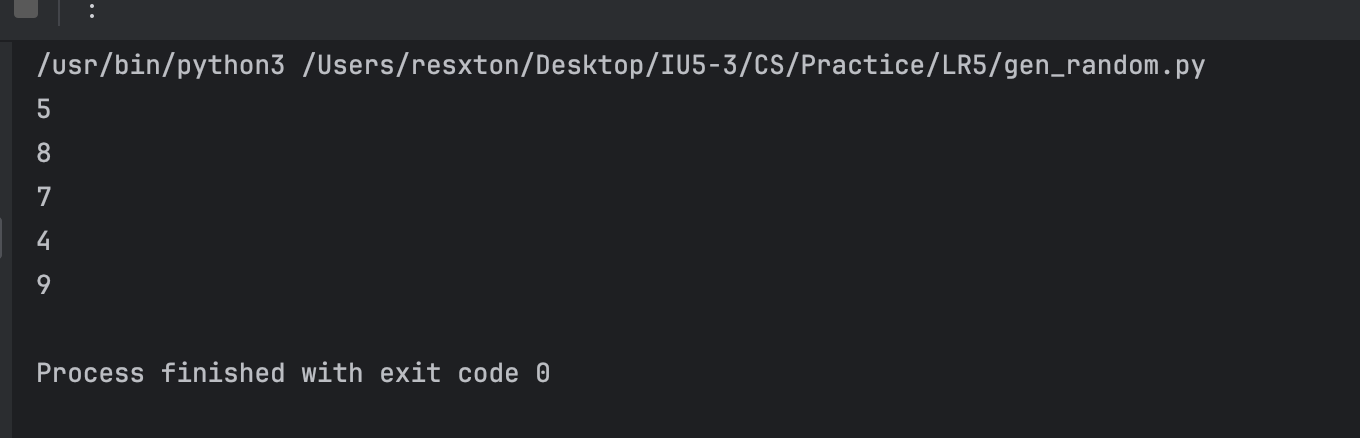
pass

# Необходимо реализовать генератор

Код программы 2

import random  
  
  
def gen\_random(num\_count, begin, end):  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)  
  
  
def main():  
 random\_numbers = gen\_random(5, 1, 10)  
  
 for number in random\_numbers:  
 print(number)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод 2



Задание 3

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(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.

Шаблон для реализации класса-итератора:

# Итератор для удаления дубликатов

class Unique(object):

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

# Нужно реализовать конструктор

# В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore\_case,

# в зависимости от значения которого будут считаться одинаковыми строки в разном регистре

# Например: ignore\_case = True, Aбв и АБВ - разные строки

# ignore\_case = False, Aбв и АБВ - одинаковые строки, одна из которых удалится

# По-умолчанию ignore\_case = False

pass

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

# Нужно реализовать \_\_next\_\_

pass

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

return self

Код программы 3

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.data = iter(items)  
 self.unique\_set = set()  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
  
 def \_\_next\_\_(self):  
 while True:  
 item = next(self.data)  
 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  
  
  
def main():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 unique\_iterator1 = Unique(data1)  
 print(list(unique\_iterator1)) # Вывод: [1, 2]  
  
 from gen\_random import gen\_random  
  
 data2 = gen\_random(10, 1, 3)  
 unique\_iterator2 = Unique(data2)  
 print(list(unique\_iterator2)) # Вывод: комбинация из [0;1] цифр 1, 2, 3  
  
 data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 unique\_iterator3 = Unique(data3)  
 print(list(unique\_iterator3)) # Вывод: ['a', 'A', 'b', 'B']  
  
 unique\_iterator4 = Unique(data3, ignore\_case=True)  
 print(list(unique\_iterator4)) # Вывод: ['a', 'b']  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод 3



Задание 4

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.

Шаблон реализации:

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

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

result = ...

print(result)

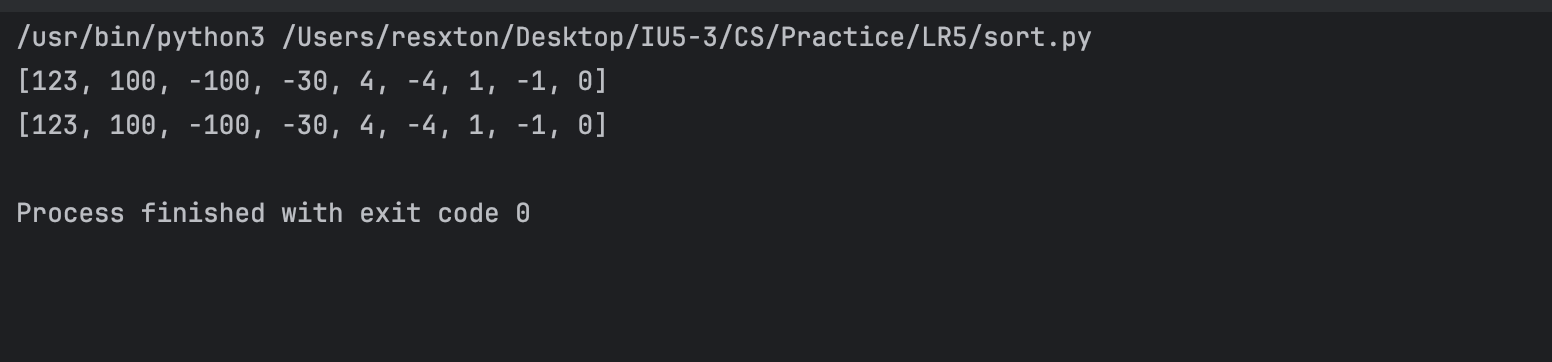
result\_with\_lambda = ...

print(result\_with\_lambda)

Код программы 4

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

Вывод 4



Задание 5

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.

Шаблон реализации:

# Здесь должна быть реализация декоратора

@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()

Результат выполнения:

test\_1

1

test\_2

iu5

test\_3

a = 1

b = 2

test\_4

1

2

Код программы 5

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
  
 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 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]  
  
  
def main():  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
 print('!!!!!!!!')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Вывод 5



Задание 6

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).

Код программы 6

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

Вывод 6



Задание 7

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.

Шаблон реализации:

import json

import sys

# Сделаем другие необходимые импорты

path = None

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

with open(path) 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

@print\_result

def f1(arg):

raise NotImplemented

@print\_result

def f2(arg):

raise NotImplemented

@print\_result

def f3(arg):

raise NotImplemented

@print\_result

def f4(arg):

raise NotImplemented

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

with cm\_timer\_1():

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

Код программы 7

import json  
from cm\_timer import cm\_timer\_1  
from print\_result import print\_result  
from gen\_random import gen\_random  
  
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))  
  
  
@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: f"{x}, с опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg, gen\_random(len(arg), 100000, 200000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Вывод 7

f1

… Профессии

f2

программист

программист / senior developer

программист 1с

программист c#

программист с++

программист с++/с#/java

программист/ junior developer

программист/ технический специалист

программистр-разработчик информационных систем

f3

программист, с опытом Python

программист / senior developer, с опытом Python

программист 1с, с опытом Python

программист c#, с опытом Python

программист с++, с опытом Python

программист с++/с#/java, с опытом Python

программист/ junior developer, с опытом Python

программист/ технический специалист, с опытом Python

программистр-разработчик информационных систем, с опытом Python

f4

программист, с опытом Python, зарплата 197399 руб.

программист / senior developer, с опытом Python, зарплата 145798 руб.

программист 1с, с опытом Python, зарплата 116320 руб.

программист c#, с опытом Python, зарплата 191592 руб.

программист с++, с опытом Python, зарплата 192705 руб.

программист с++/с#/java, с опытом Python, зарплата 137819 руб.

программист/ junior developer, с опытом Python, зарплата 195046 руб.

программист/ технический специалист, с опытом Python, зарплата 164661 руб.

программистр-разработчик информационных систем, с опытом Python, зарплата 178371 руб.

time: 0.00598454475402832