**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 № 3-4

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Бромберг Е.А. |  | Нардид А.Н. |
|  |  |  |

Москва, 2024 г.

**Постановка задачи**

### 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.

### 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'

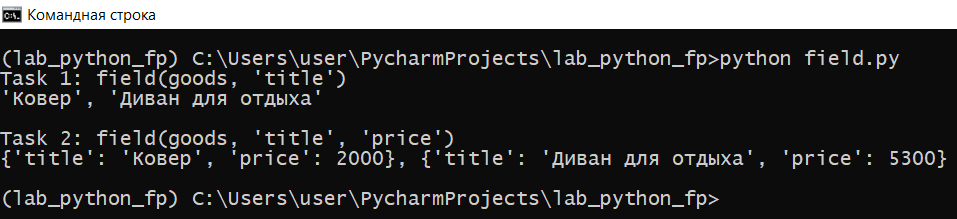
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* 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**

def field(items, \*args):  
 assert len(args) > 0  
 for i in range(len(items)):  
 if len(args) == 1:  
 if items[i][args[0]] != None:  
 yield "'{}'".format(items[i][args[0]])  
 else:  
 flag = True  
 map = dict()  
 for j in range(len(args)):  
 if items[i][args[j]] != None:  
 flag = False  
 map[args[j]] = items[i][args[j]]  
 if not flag:  
 yield map  
  
  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
]  
print("Task 1: field(goods, 'title')")  
test = field(goods, 'title')  
print(\*test, sep=', ')  
print()  
print("Task 2: field(goods, 'title', 'price')")  
test = field(goods, 'title', 'price')  
print(\*test, sep=', ')

**Экранные формы с примерами выполнения программы**



### Задача 2 (файл gen\_random.py)

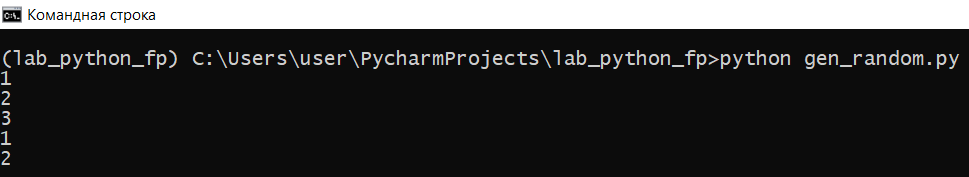
Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

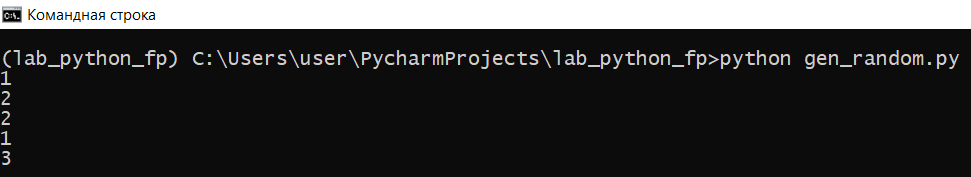
gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

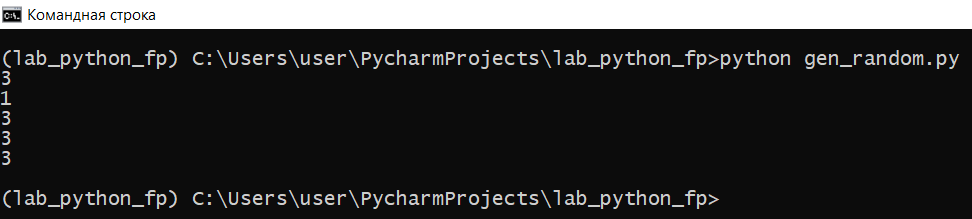
**Текст программы**

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

**Экранные формы с примерами выполнения программы**







### Задача 3 (файл unique.py)

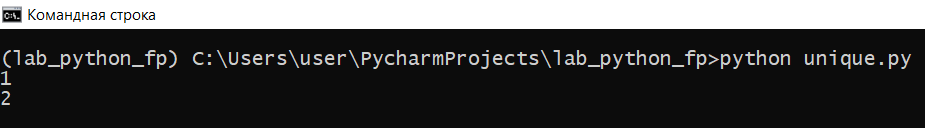
* Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
* Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки 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.

**Тексты программы**

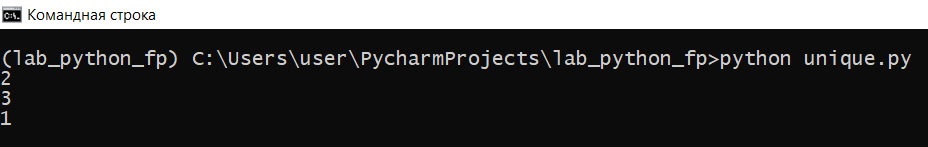
from gen\_random import gen\_random  
  
  
# Итератор для удаления дубликатов  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 # Нужно реализовать конструктор  
 # В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore\_case,  
 # в зависимости от значения 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  
 self.used\_elements = set()  
 self.data = list(items)  
 self.index = 0  
 if len(kwargs) != 0:  
 self.ignore\_case = kwargs['ignore\_case']  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 if self.index >= len(self.data):  
 raise StopIteration  
 else:  
 current = self.data[self.index]  
 self.index = self.index + 1  
 flag = True  
 if type(current) == str and self.ignore\_case == True:  
 for i in self.used\_elements:  
 if type(i) == str and i.lower() == current.lower():  
 flag = False  
 break  
 if current not in self.used\_elements and flag:  
 self.used\_elements.add(current)  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
for i in Unique(data, ignore\_case=True):  
 print(i)

**Экранные формы с примерами выполнения программы**

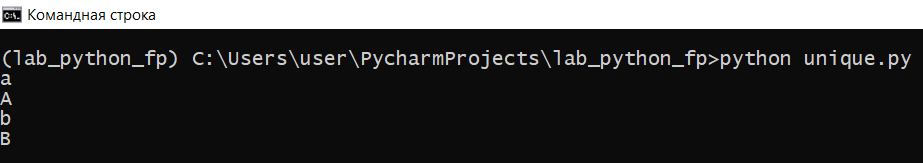
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]



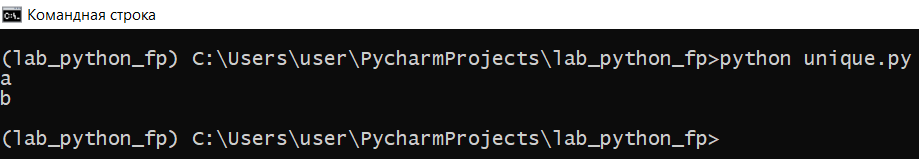
data = gen\_random(10, 1, 3)



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



data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’] + ignore\_case=True



### 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]

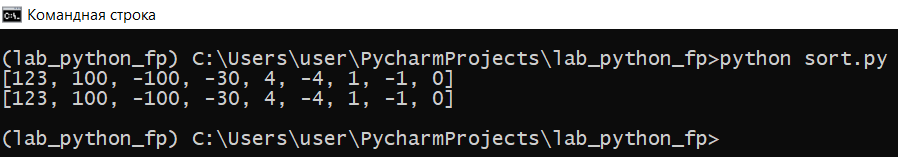
Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

**Текст программы**

data = [4, -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))  
 print(result\_with\_lambda)

**Экранные формы с примерами выполнения программы**



### Задача 5 (файл print\_result.py)

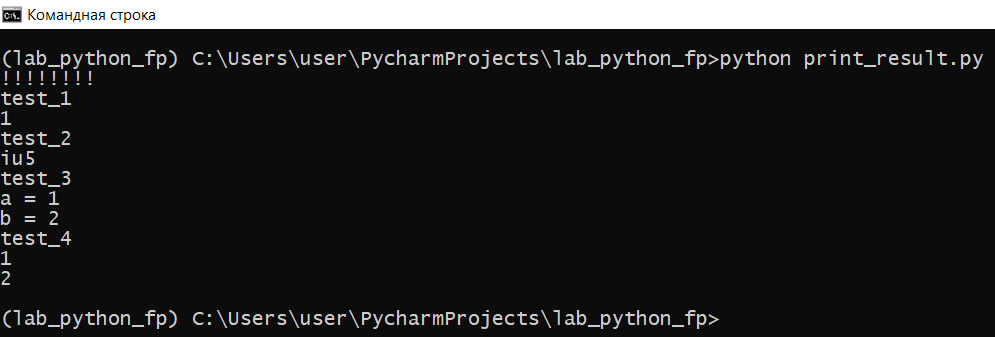
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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**

# Здесь должна быть реализация декоратора  
def print\_result(func\_to\_decorate):  
 def decorated\_func():  
 result = func\_to\_decorate()  
 print(func\_to\_decorate.\_\_name\_\_)  
 if type(result) == list:  
 for i in result:  
 print(i)  
 elif type(result) == dict:  
 for i in result.keys():  
 print('{} = {}'.format(i, result[i]))  
 else:  
 print(result)  
 return result  
 return decorated\_func  
  
@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()

**Экранные формы с примерами 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).

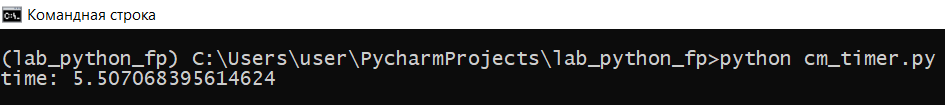
cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

**Текст программы**

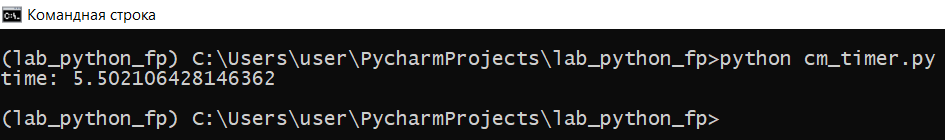
import time  
from contextlib import contextmanager  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.start = 0  
  
 def \_\_enter\_\_(self):  
 self.start = time.time()  
 return self.start  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 if exp\_type is not None:  
 print(exp\_type, exp\_value, traceback)  
 else:  
 print('time: {}'.format(time.time() - self.start))  
  
  
@contextmanager  
def cm\_timer\_2():  
 start = time.time()  
 yield start  
 print('time: {}'.format(time.time() - start))  
  
  
with cm\_timer\_1():  
 time.sleep(5.5)

**Экранные формы с примерами выполнения программы**

cm\_timer\_1



cm\_timer\_2



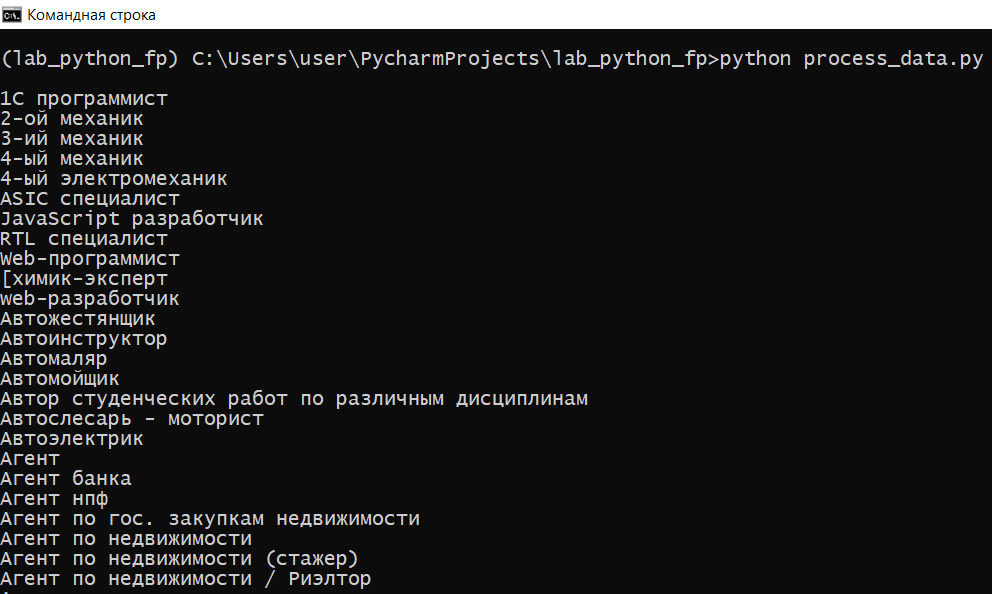
### Задача 7 (файл process\_data.py)

* 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/ugapanyuk/BKIT_2021/blob/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  
import sys  
from unique import Unique  
from gen\_random import gen\_random  
from cm\_timer import cm\_timer\_1  
  
path = './data\_light.json'  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
with open(path, "r", 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(func\_to\_decorate):  
 def decorated\_func(arg):  
 result = func\_to\_decorate(arg)  
 print()  
 if type(result) == list:  
 for i in result:  
 print(i)  
 elif type(result) == dict:  
 for i in result.keys():  
 print('{} = {}'.format(i, result[i]))  
 else:  
 print(result)  
 return result  
 return decorated\_func  
  
  
def field(items, \*args):  
 assert len(args) > 0  
 for i in range(len(items)):  
 if len(args) == 1:  
 if items[i][args[0]] != None:  
 yield "{}".format(items[i][args[0]])  
 else:  
 flag = True  
 map = dict()  
 for j in range(len(args)):  
 if items[i][args[j]] != None:  
 flag = False  
 map[args[j]] = items[i][args[j]]  
 if not flag:  
 yield map  
  
  
@print\_result  
def f1(arg):  
 return sorted(list(Unique(field(arg, "job-name"), ignore\_case=True)))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: (x.startswith('программист') or x.startswith('Программист')), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 return [i + ", зарплата {} руб.".format(j) for i, j in zip(arg, gen\_random(len(arg), 100\_000, 200\_000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Экранные формы с примерами выполнения программы**



…

