Лабораторная работа №3

Рамазанова Диана Ахмедовна

ИБМ3\_34Б

**Функциональные возможности языка Python.**

### **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

* 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.

### 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

### 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.

### 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.

### 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.

### 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).

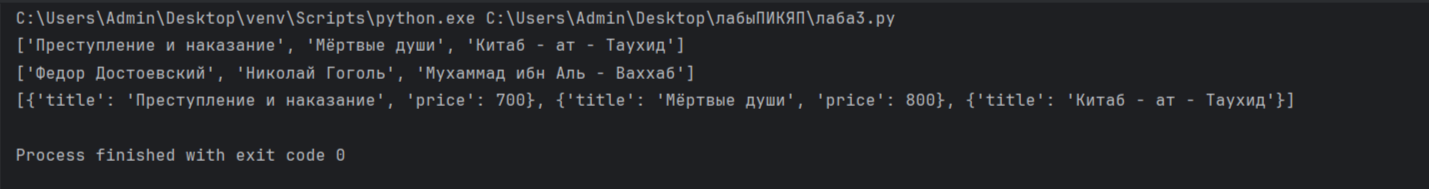
### Задача 7 (файл process\_data.py)

* 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.

Задача 1:

def field(items, \*args):  
 assert len(args) > 0 # Проверка на наличие аргументов  
 if len(args) == 1: # Если передан один аргумент  
 for item in items:  
 value = item.get(args[0])  
 if value is not None:  
 yield value  
 else: # Если передано несколько аргументов  
 for item in items:  
 result = {arg: item.get(arg) for arg in args if item.get(arg) is not None}  
 if result:  
 yield result  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 books = [  
 {'title': 'Преступление и наказание', 'author': 'Федор Достоевский', 'price': 700},  
 {'title': 'Мёртвые души', 'author': 'Николай Гоголь', 'price': 800},  
 {'title': 'Китаб - ат - Таухид', 'author': 'Мухаммад ибн Аль - Ваххаб'},  
 ]  
  
 print(list(field(books, 'title')))  
 print(list(field(books, 'author')))  
 print(list(field(books, 'title', 'price')))

Результат:



Задача 2:

import random

*def* gen\_random(count, min, max):

empty = None

if min > max:

empty = max

max = min

min = empty

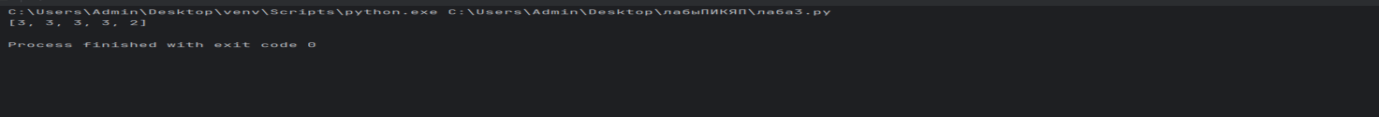
for \_ in range (count):

yield random.randint(min, max)

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

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

Результат:



Задача 3:

class Unique:

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

self.items = items

self.seen = set()

self.ignore\_case = kwargs.get('ignore\_case', False)

if isinstance(items, list):

self.iterator = iter(items)

else:

self.iterator = items

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

while True:

item = next(self.iterator)

normalized\_item = item.lower() if self.ignore\_case and isinstance(item, str) else item

if normalized\_item not in self.seen:

self.seen.add(normalized\_item)

return item

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

return self

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

data = [1, 1, 1, 2, 2, 2, 3, 3]

for i in Unique(data):

print(i)

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

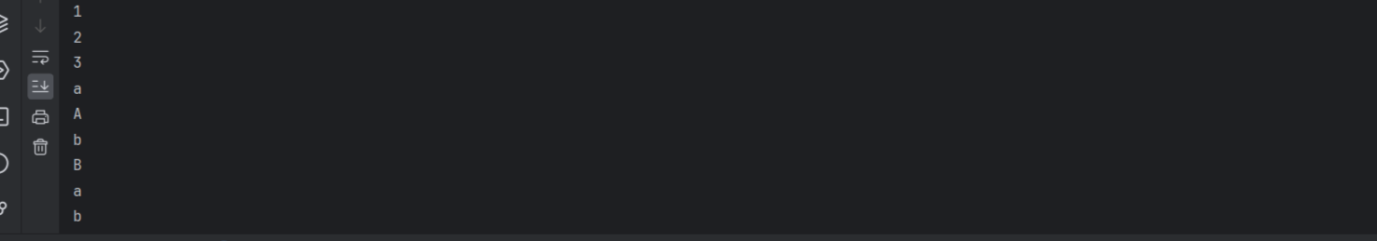
for i in Unique(data):

print(i)

for i in Unique(data, ignore\_case=True):

print(i)

Результат:



Задача 4:

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

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

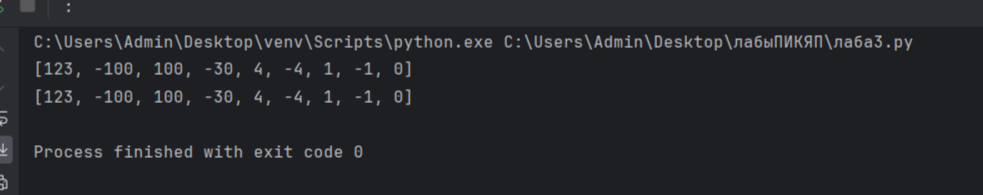
result = sorted(data, key=abs, reverse=True)

print(result)

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

print(result\_with\_lambda)

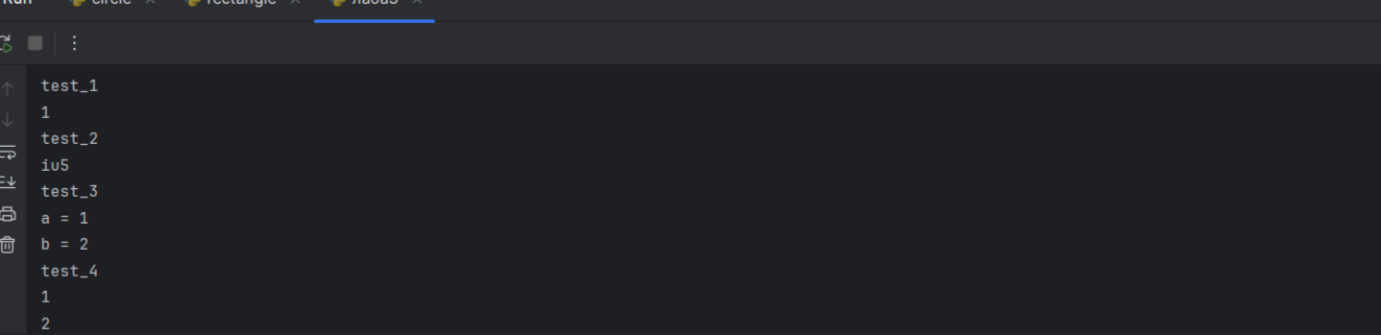
Результат:



Задача 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]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

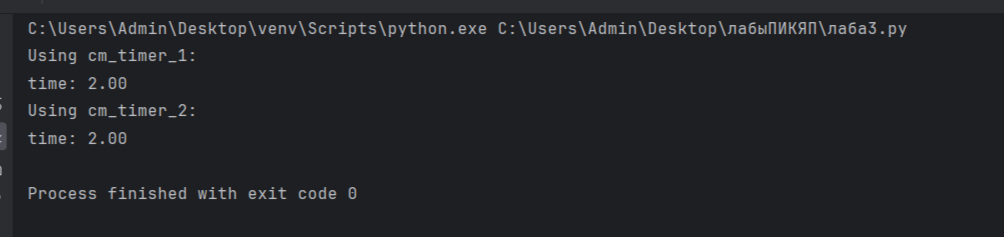
Результат:



Задача 6:

import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
 return self  
  
 def \_\_exit\_\_(self, \*args):  
 elapsed\_time = time.time() - self.start\_time  
 print(f"time: {elapsed\_time:.2f}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 elapsed\_time = time.time() - start\_time  
 print(f"time: {elapsed\_time:.2f}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 from time import sleep  
  
 print("Using cm\_timer\_1:")  
 with cm\_timer\_1():  
 sleep(2)  
 print("Using cm\_timer\_2:")  
 with cm\_timer\_2():  
 sleep(2)

Результат:



Задача 7:

import time  
import random  
from contextlib import contextmanager  
from typing import List, Dict  
  
# Декоратор для печати результата функции  
def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(result)  
 return result  
 return wrapper  
  
# Контекстный менеджер для измерения времени выполнения  
@contextmanager  
def cm\_timer\_1():  
 start\_time = time.time()  
 yield  
 end\_time = time.time()  
 print(f"Время выполнения: {end\_time - start\_time:.4f} секунд")  
  
# Пример данных  
data = [  
 {"profession": "Программист C#", "location": "Москва", "salary": 120000},  
 {"profession": "пРОГРАММИСТ Java", "location": "Санкт-Петербург", "salary": 130000},  
 {"profession": "Тестировщик", "location": "Екатеринбург", "salary": 90000},  
 {"profession": "Программист Python", "location": "Казань", "salary": 150000},  
 {"profession": "разработчик Frontend", "location": "Новосибирск", "salary": 110000},  
 {"profession": "Программист C++", "location": "Мурманск", "salary": 145000},  
]  
  
# Функция f1 - получение уникальных профессий, сортировка без учета регистра  
@print\_result  
def f1(data: List[Dict]) -> List[str]:  
 return sorted(set(item['profession'].lower() for item in data))  
  
# Функция f2 - фильтрация профессий, начинающихся со слова "программист"  
@print\_result  
def f2(professions: List[str]) -> List[str]:  
 return list(filter(lambda x: x.startswith('программист'), professions))  
  
# Функция f3 - добавление строки "с опытом Python"  
@print\_result  
def f3(programmers: List[str]) -> List[str]:  
 return list(map(lambda x: f"{x} с опытом Python", programmers))  
  
# Функция f4 - генерация зарплат для каждой специальности  
@print\_result  
def f4(programmers\_with\_experience: List[str]) -> List[str]:  
 salaries = [random.randint(100000, 200000) for \_ in programmers\_with\_experience]  
 return [f"{profession}, зарплата {salary} руб." for profession, salary in zip(programmers\_with\_experience, salaries)]  
  
# Главная часть - использование функций вместе в контексте менеджера  
with cm\_timer\_1():  
 unique\_professions = f1(data) # f1: Уникальные профессии  
 programmers = f2(unique\_professions) # f2: Фильтрация программирующих профессий  
 programmers\_with\_experience = f3(programmers) # f3: Добавление опыта  
 final\_output = f4(programmers\_with\_experience) # f4: Генерация зарплат

Результат:

