

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”



Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»

Выполнил:
Студент группы ИУ5-31Б
Паронько Д.И.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

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 располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача №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 количество 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': 5300, 'color': 'black'}
]

def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        return [dct.get(args[0]) for dct in items if dct.get(args[0]) is not
None]
    else:
```

```

        return [{key : dct.get(key) for key in args if dct.get(key) is not None}
for dct in items]

if __name__ == "__main__":
    print(field(goods, 'title'))
    print(field(goods, 'title', 'price'))

```

Результат выполнения

- juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp\$ python3 field.py
['Ковер', 'диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'диван для отдыха', 'price': 5300}]
- juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp\$

Задача №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

Текст программы

```

# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки

import random

def gen_random(num_count, begin, end):
    return [random.randint(begin, end) for i in range(num_count)]

if __name__ == "__main__":
    print(gen_random(5, 2, 10))

```

Результат выполнения

- juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp\$ python3 gen_random.py
[10, 5, 4, 5, 5]
- juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp\$

Задача №3

- Необходимо реализовать генератор gen_random(количество, минимум, максимум), 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.

Текст программы

```
from typing import List, Any

class Unique(object):
    def __init__(self, items, **kwargs):
        self._items = iter(items)
        self._ignore_case = kwargs.get('ignore_case', False)
        self._seen = set()

    def __next__(self):
        while True:
            try:
                cur_item = next(self._items)
            except StopIteration:
                raise StopIteration

            key = cur_item

            if self._ignore_case and isinstance(cur_item, str):
                key = cur_item.lower()

            if key not in self._seen:
                self._seen.add(key)
                return cur_item

    def __iter__(self):
        return self

def test():
    data = [1, 4, 3, 2, 2, 1, 10]
    print([item for item in Unique(data)])

if __name__ == "__main__":
    test()
```

Результат выполнения

```
● juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp$ python3 unique.py
[1, 4, 3, 2, 10]
○ juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp$ █
```

Задача №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:

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

Текст программы

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

def abs_sort_key(x):
    return abs(x)

if __name__ == '__main__':
    print(sorted(data, key = abs_sort_key, reverse=True))
    print(sorted(data, key = lambda x: abs(x), reverse=True))
```

Результат выполнения

```
● juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp$ python3 sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
○ juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp$ █
```

Задача №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.

Текст программы

```
from typing import List, Any, Dict

def print_in_column(data):
    for el in data:
        print(el)

def print_result(func):
    def wrapper(*args, **kwargs):
        print("function name:", func.__name__)
        result = func(*args, **kwargs)
        if isinstance(result, List):
            print("result:")
            print_in_column(result)
        elif isinstance(result, Dict):
            print("result:")
            print_in_column([f'{key} = {value}' for key, value in
result.items()])
        else:
            print(f"result: {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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат выполнения

```
● juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp$ python3 print_result.py
function name: test_1
result: 1
function name: test_2
result: iu5
function name: test_3
result:
a = 1
b = 2
function name: test_4
result:
1
2
```

Задача №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`).

Текст программы

```
import time
from contextlib import contextmanager


class cm_timer_1:
    def __enter__(self):
        self.start = time.perf_counter()
        return self
    def __exit__(self, exc_type, exc_val, exc_tb):
        end = time.perf_counter()
        if exc_type is not None:
            print("Ошибка в блоке кода")
            return False

        print(f"time: {end - self.start}")


@contextmanager
def cm_timer_2():
    start = time.perf_counter()

    try:
        yield start
    except Exception as e:
        print("Ошибка в блоке кода")
        raise
```

```

print(f"time: {time.perf_counter() - start}")

def test():
    print("cm_timer_1")
    with cm_timer_1():
        time.sleep(5.5)

    print("\ncm_timer_2")
    with cm_timer_2():
        time.sleep(5.5)

if __name__ == "__main__":
    test()

```

Результат выполнения

- juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/lab_2/lab_python_fp\$ python3 cm_timer.py
 cm_timer_1
 time: 5.501594246998138

 cm_timer_2
 time: 5.5389274949993705
 -

Задача №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.

Текст программы

```

import json

import random

from typing import List, Dict

import lab_python_fp.field as field
import lab_python_fp.unique as unique
import lab_python_fp.gen_random as gen_random
import lab_python_fp.sort as sort
import lab_python_fp.cm_timer as cm_timer
import lab_python_fp.print_result as print_result


with open("data.json", 'r', encoding='utf-8') as file:
    json_data = json.load(file)

@print_result.print_result
def f1(data: List[Dict]) -> List:
    return sorted(unique.Unique(field.field(data, "job-name"), ignore_case=True),
key = lambda s: s.lower())

@print_result.print_result
def f2(data: List) -> List:
    return list(filter(lambda s: "программист" in s.lower(), data))

@print_result.print_result
def f3(data: List) -> List:
    return list(map(lambda job: job + " с опытом Python", data))

@print_result.print_result
def f4(data: List) -> List:
    programmer_salary = list(zip(data, gen_random.gen_random(len(data), 100_000,
200_000)))
    return [f"{spec}, зарплата {salary} р." for spec, salary in
programmer_salary]

```

```
if __name__ == "__main__":
    with cm_timer.cm_timer_1():
        f4(f3(f2(f1(json_data))))
```

Результат выполнения

Т.к. вывод результата выполнения слишком большой, его можно посмотреть в репозитории по ссылке: [Результат Выполнения](#)