

**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  
«Функциональные возможности языка Python»**

Выполнил:  
студент группы ИУ5-32Б  
Канаева Д.Ч.

Подпись и дата:

Проверил:  
преподаватель каф. ИУ5  
Нардид А.Н.

Подпись и дата:

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

## Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fr. Решение каждой задачи должно располагаться 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 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:

```
# 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
    for item in items:
        if len(args) == 1:
            field_name = args[0]
            if field_name in item and item[field_name] is not None:
                yield item[field_name]
        else:
            filtered = {}
            all_none = True
            for arg in args:
                if arg in item and item[arg] is not None:
                    filtered[arg] = item[arg]
            all_none = False
```

```

        if not all_none:
            yield filtered

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

if __name__ == '__main__':
    for x in field(goods, 'title'):
        print(x)

```

**Тестовые результаты программы:**

```

C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>field.py
Ковер
Диван для отдыха

```

## 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:**

```

# 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):
    pass
    for x in range(num_count):
        yield random.randint(begin, end)

if __name__ == '__main__':
    for x in gen_random(5, 1, 3):
        print(x)

```

**Тестовые результаты программы:**

```

C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>gen_random.py
3
2
3
2
3

```

### 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(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.

### Текст программы:

```
# Итератор для удаления дубликатов

class Unique(object):
    def __init__(self, items, **kwargs):
        self.data = items
        self.ignore_case = kwargs.get("ignore_case", False)
        self.seen = set()

    def __next__(self):
        while True:
            value = next(self.data)
            if isinstance(value, str) and self.ignore_case:
                key = value.lower()
            else:
                key = value
            if key not in self.seen:
                self.seen.add(key)
                return value

    def __iter__(self):
        return self

    def lists(self):
        array = []
        for x in self:
            array.append(x)
```

```

        return array

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
#data = [1, 2, 2, 3, 4, 4, 5]

if __name__ == '__main__':
    unique_items = Unique(iter(data))
    for item in unique_items:
        print(item)

```

Тестовые результаты программы:

```

C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>unique.py
a
A
b
B

```

## 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=lambda x: abs(x), reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=abs, reverse=True)
    print(result_with_lambda)

```

Тестовые результаты программы:

```

C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

```

## 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 выводиться в столбик через знак равенства.

Текст программы:

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(f'\nИмя функции: {func.__name__}')
        if isinstance(result, list):
            print('\n'.join(map(str, result)))
        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()
```

## Тестовые результаты программы:

```
C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>print_result.py
!!!!!!!

Имя функции: test_1
1

Имя функции: test_2
iu5

Имя функции: test_3
a = 1
b = 2

Имя функции: test_4
1
2
```

## 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

# Реализация на основе класса
class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(f"time: {time.time() - self.start_time}")

# Реализация с использованием contextlib
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print(f"time: {time.time() - start_time}")
```

```
with cm_timer_1():  
    time.sleep(5.5)  
  
with cm_timer_2():  
    time.sleep(5.5)
```

### Тестовые результаты программы:

```
C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>cm_timer.py  
time: 5.5073325634002686  
time: 5.509819984436035
```

### 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
import sys
import field
import unique
import gen_random
import cm_timer
from print_result import print_result

path = r"C:\Users\123\Desktop\Study\3 sem\PCPL\data_light.json"

with open(path, encoding='utf-8') as f:
    data = json.load(f)

@print_result
def f1(arr):
    return sorted(unique.Unique.lists(unique.Unique(field.field(arr, 'job-
name'), ignore_case=False)), key=lambda x: x.lower())

@print_result
def f2(arr):
    filtered = filter(lambda x: x.startswith('программист') or
x.startswith('Программист'), arr)
    return list(filtered)

@print_result
def f3(arr):
    modified = map(lambda x: x + " с опытом Python", arr)
    return list(modified)

@print_result
def f4(arrs):
    salaries = gen_random.gen_random(len(arrs), 100000, 200000)
    combined = [f"{arr}, зарплата {salary} руб." for arr, salary in zip(arrs,
salaries)]
    return combined

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

## Результат выполнения программы:

```
C:\Users\123\Documents\PCPL_3sem\Lab3\lab_python_fp>process_data.py
time: 5.503216028213501
time: 5.501112222671509
```

```
Имя функции: f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
автомойщик
Автор студенческих работ по различным дисциплинам
автослесарь
Автослесарь
Автослесарь – моторист
Автоэлектрик
```

...

```
энтомолог
Юрисконсульт
юрисконсульт
юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
юрист
Юрист (специалист по сопровождению международных договоров, английский – разговорный)
Юрист волонтер
Юристконсульт

Имя функции: f2
Программист
программист
Программист / Senior Developer
Программист 1С
программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
```

Имя функции: f3

Программист с опытом Python  
программист с опытом Python  
Программист / Senior Developer с опытом Python  
Программист 1C с опытом Python  
программист 1C с опытом Python  
Программист C# с опытом Python  
Программист C++ с опытом Python  
Программист C++/C#/Java с опытом Python  
Программист/ Junior Developer с опытом Python  
Программист/ технический специалист с опытом Python  
Программист-разработчик информационных систем с опытом Python

Имя функции: f4

Программист с опытом Python, зарплата 176298 руб.  
программист с опытом Python, зарплата 153756 руб.  
Программист / Senior Developer с опытом Python, зарплата 189570 руб.  
Программист 1C с опытом Python, зарплата 119418 руб.  
программист 1C с опытом Python, зарплата 123332 руб.  
Программист C# с опытом Python, зарплата 104299 руб.  
Программист C++ с опытом Python, зарплата 163215 руб.  
Программист C++/C#/Java с опытом Python, зарплата 183295 руб.  
Программист/ Junior Developer с опытом Python, зарплата 188003 руб.  
Программист/ технический специалист с опытом Python, зарплата 109710 руб.  
Программист-разработчик информационных систем с опытом Python, зарплата 165255 руб.  
time: 0.03301572799682617