**Московский государственный технический университет им. Н.Э. Баумана**

Факультет «Радиотехнический»
Кафедра РТ5 «Системы обработки информации и управления»

Курс «Объектно-ориентированные возможности языка Python»

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

Выполнил:

студент группы РТ5-31Б:
Эрендженов Д.Б.

Руководитель:

преподаватель каф. ИУ5
Гапанюк Ю.Е.

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.

### 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.

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
```

**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
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
```

**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.
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, Абв и АБВ - разные строки
        #           ignore_case = False, Абв 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
```

**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.

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)
```

**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.

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
```

**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 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))))
```

**Текст программы:**

Cm_timer.py:

```
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)
```

field.py:

```
# 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}
```

```python
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)
```

Gen_random.py:

```python
# 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

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)
```

Print_result.py:

```python
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()
```

process_data.py:

```
import json
import sys
import field
import unique
import gen_random
import cm_timer
from print_result import print_result

path = r"/Users/danir/PycharmProjects/lab(3-4)/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))))
```

sort.py:

```
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)
```

unique.py:

```
# 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)
```

## Результаты выполнения программы:

```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/cm_timer.py
time: 5.505178213119507
time: 5.505054950714111

Process finished with exit code 0
```

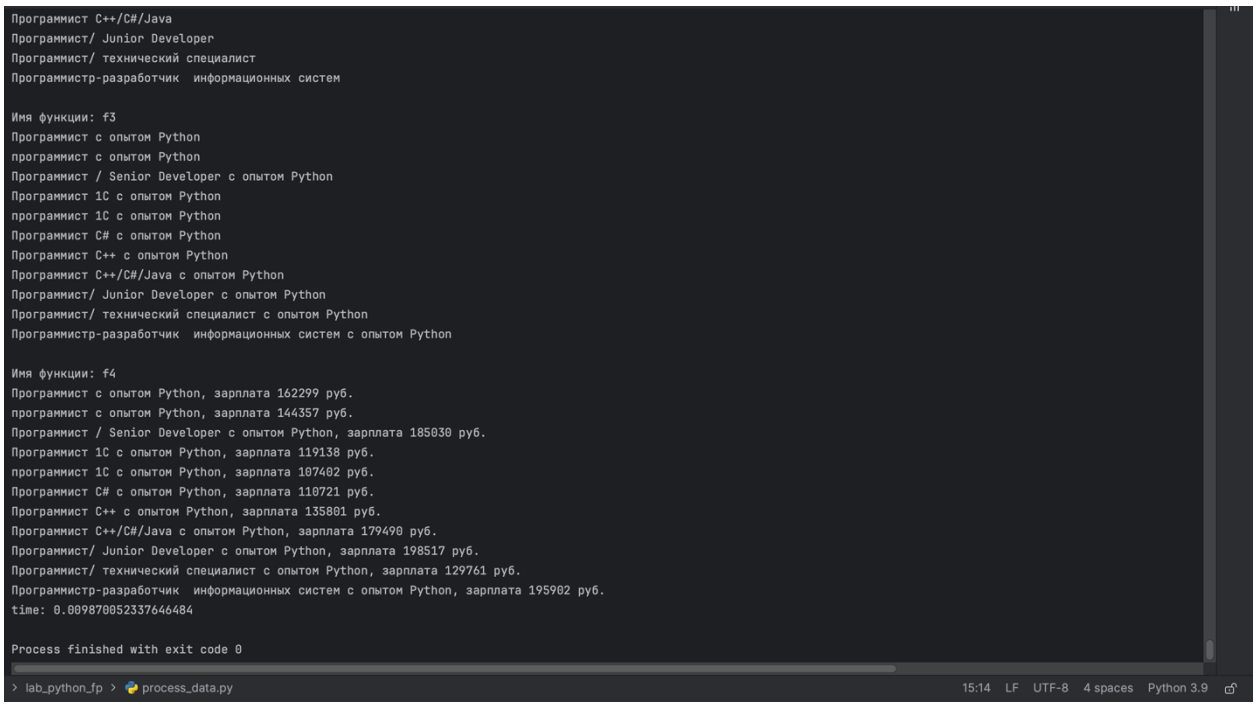
```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/field.py
Ковер
Диван для отдыха

Process finished with exit code 0
```

```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/gen_random.py
2
3
3
1
2

Process finished with exit code 0
```

```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/print_result.py
!!!!!!!!

Имя функции: test_1
1

Имя функции: test_2
rt5

Имя функции: test_3
a = 1
b = 2

Имя функции: test_4
1
2

Process finished with exit code 0
```

```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/process_data.py
time: 5.5051000118255615
time: 5.50508189201355

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
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
агент по гос. закупкам недвижимости
Агент по недвижимости
```

•••




```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

```
/usr/bin/python3 /Users/danir/PycharmProjects/lab(3-4)/lab_python_fp/unique.py
a
A
b
B

Process finished with exit code 0
```