



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»
Курс «Базовые компоненты интернет-технологий»
Функциональные возможности языка Python**

**Выполнил:
Студент группы ИУ5-34Б
Сергеев Никита
Дата и подпись:**

**Преподаватель:
Гапанюк Ю.Е.
Дата и подпись:**

2021 г.

Постановка задачи

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.

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 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(1, 3, 10)
```

`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 выводиться 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 С# 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 С# 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

Файл cm_timer.py

```
import time
import contextlib
class cm_timer_1:

    def __enter__(self):
        self.s_time = time.time()

    def __exit__(self, ex1, ex2, ex3):
        print(time.time() - self.s_time)

@contextlib.contextmanager
def cm_timer_2():
    s_time = time.time()
    yield
    print(time.time() - s_time)
```

Файл field.py

```
def field(data, *args):
    if len(args) == 1:
        for i in data:
            if args[0] in i:
                yield i[args[0]]
    else:
        for i in data:
            tmp = dict()
            for j in args:
                if j in i:
                    tmp[j] = i[j]
            yield tmp
```

Файл gen_random.py

```
from random import randint

def gen_random(n, s, f):
    for x in range(n):
        yield randint(s, f)
```

Файл print_result.py

```
def print_result(obj):
    def decorated(*args, **kwargs):
        print(obj.__name__)
        data = obj(*args, **kwargs)
        if isinstance(data, dict):
            for i in data:
                print(i, ' = ', data[i])
        elif isinstance(data, list):
            for i in data:
                print(i)
        else:
            print(data)
    return data
```



```
return decorated
```

Файл sort.py

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

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

result_with_lambda = sorted(data, key = lambda x: abs(x), reverse=True)

print(result, result_with_lambda, sep='\n')
```

Файл unique.py

```
class Unique:
    def __init__(self, spisok, **kwargs):
        self.data = spisok
        self.used_elements = set()
        self.i = 0
        self.ignore_case = len(kwargs)
    def __iter__(self):
        return self
    def __next__(self):
        while (self.i < len(self.data)):
            if isinstance(self.data[self.i], str):
                if self.ignore_case == 1:
                    s = self.data[self.i].lower()
                else:
                    s = self.data[self.i]
                if s not in self.used_elements:
                    self.used_elements.add(s)
                    return s
            else:
                if self.data[self.i] not in self.used_elements:
                    self.used_elements.add(self.data[self.i])
                    return self.data[self.i]
            self.i += 1
        self.i = 0
        raise StopIteration
```

Файл main.py

```
import json
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1

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

@print_result
def f1(data):
    return sorted(list(Unique([key[value] for key in data
                               for value in key if value.lower() == 'job-
name']))))
```

```

@print_result
def f2(data):
    return list(filter(lambda profession: profession.lower().startswith('программист'),
data))

@print_result
def f3(data):
    return list(map(lambda profession: profession + ' с опытом Python', data))

@print_result
def f4(data):
    return [profession + ', зарплата ' + str(salary)
for profession, salary in zip(data, gen_random(len(data), 100000, 200000))]

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

```

Результат выполнения

```

Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
программист
программист 1C
f3
Программист с опытом Python
Программист / Senior Developer 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
программист с опытом Python
программист 1C с опытом Python
f4
Программист с опытом Python, зарплата 165695
Программист / Senior Developer с опытом Python, зарплата 130750
Программист 1C с опытом Python, зарплата 144022
Программист C# с опытом Python, зарплата 186183
Программист C++ с опытом Python, зарплата 117809
Программист C++/C#/Java с опытом Python, зарплата 132452
Программист/ Junior Developer с опытом Python, зарплата 193160
Программист/ технический специалист с опытом Python, зарплата 147367
Программист-разработчик информационных систем с опытом Python, зарплата 182425
программист с опытом Python, зарплата 186742
программист 1C с опытом Python, зарплата 181653
0.5651750564575195

```