



**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 управление»
Кафедра «Системы обработки информации и управления»**

**Курс «Разработка интернет-приложений»
Отчет
по лабораторной работе №3**

Студент Дубянский А. И., ИУ5Ц-71Б.

(Подпись, дата)

Преподаватель Гапанюк Ю.Е.

(Подпись, дата)

Москва - 2020

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.

- 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 (`field_1.py`):

```
from typing import List # необходим для задания типа для аннотации типов
print("\nДубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3\n")

def field(items: List[dict], *args: str) -> iter:
    """Создает генератор для отбора записей в списке словарей
    items по ключам, указанным в args."""
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            if args[0] in i.keys() and not i[args[0]] is None:
                yield i[args[0]]
    else:
        for i in items:
            temp_dict = {}
            for key in args:
                if key in i.keys() and not i[key] is None:
                    temp_dict[key] = i[key]
            if len(temp_dict) > 0:
                yield temp_dict

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 1500, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
        {'color': 'black'}
    ]
    print(str(list(field(goods, 'title')))[1:-1])
    print(str(list(field(goods, 'title', 'price')))[1:-1])
```

Результат (`field_1.py`):

```
Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

'Ковер', 'Диван для отдыха'
{'title': 'Ковер', 'price': 1500}, {'title': '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.

Текст программы (gen_random_2.py):

```
import random
print("\nДубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3\n")

def gen_random(num_count, begin, end):
    """Генерирует num_count случайных чисел от begin до end, включая их."""
    for i in range(num_count):
        yield random.randrange(begin, end + 1)

if __name__ == '__main__':
    g = gen_random(5, 1, 3)

    for i in gen_random(5, 1, 3):
        print(i)
    print('\n')
```

Результат (gen_random_2.py):

```
Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

2
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.

Текст программы (unique_3.py):

```
from gen_random_2 import gen_random

class Unique:
    """Возвращает итератор, который принимает на вход массив или
    генератор items и итерируется по элементам, пропуская дубликаты.
    Также может принимать булевый параметр ignore_case. Если ignore_case =
    True, то регистр игнорируется (то есть 'a' и 'A' – одно и то же,
    иначе – регистр учитывается (то есть 'a' и 'A' – разные символы.
    По умолчанию ignore_case = False."""
    def __init__(self, items, **kwargs):
        self.used_elements = set() # применяется для хранения уникальных
элементов items
        self.data = list(items)
        self.index = 0 # применяется для отслеживания индекса итерируемого
элемента в items
        if 'ignore_case' in kwargs.keys():
            self.ignore_case = kwargs['ignore_case']
        else:
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            current = self.data[self.index]
            self.index += 1
            if ((self.ignore_case or not isinstance(current, str))
                and current not in self.used_elements):
                self.used_elements.add(current)
                return current
            elif (not self.ignore_case and isinstance(current, str)
                  and current.upper() not in self.used_elements
                  and current.lower() not in self.used_elements):
                self.used_elements.add(current.upper())
                self.used_elements.add(current.lower())
                return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    data_int = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data_rand = gen_random(10, 1, 10) # генерируется 10 случайных чисел от 1
до 10
    data_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print('Уникальные числа: ', str(list(Unique(data_int)))[1:-1])
```

```
print('Уникальные случайные числа: ', str(list(Unique(data_rand)))[1:-1])
print('Уникальные строки без игнорирования регистра по умолчанию: ',
str(list(Unique(data_str)))[1:-1])
print('Уникальные строки с игнорированием регистра: ',
str(list(Unique(data_str, ignore_case=True)))[1:-1])
print('Уникальные строки без игнорирования регистра: ',
str(list(Unique(data_str, ignore_case=False)))[1:-1])
```

Результат (unique_3.py):

Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

Уникальные числа: 1, 2

Уникальные случайные числа: 7, 8, 10, 4, 5, 1

Уникальные строки без игнорирования регистра по умолчанию: 'a', 'b'

Уникальные строки с игнорированием регистра: 'a', 'A', 'b', 'B'

Уникальные строки без игнорирования регистра: 'a', '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.

Необходимо решить задачу двумя способами:

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

Текст программы (sort_4.py):

```
print("\nДубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3\n")
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)
    print('Без использования lambda-функции: ', result)
    result_with_lambda = sorted(data, key = lambda x: x if x >= 0 else -x,
reverse=True)
    print('С использованием lambda-функции: ', result with lambda)
```

Результат (sort_4.py):

```
Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

Без использования lambda-функции: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
С использованием lambda-функции: [123, 100, -100, -30, 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 (print_result_5.py):

```
print("\nДубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3\n")

def print_result(func):
    def decorated_func(*args): # args - для 7-го пункта
        print(func.__name__)
        return_value = func(*args)
        if isinstance(return_value, list):
            for value in return_value:
                print(str(value))
        elif isinstance(return_value, dict):
            for key in return_value.keys():
                print(str(key) + ' = ' + str(return_value[key]))
        else:
            print(return_value)
        return return_value
    return decorated_func

@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 (print_result_5.py):

Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

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.

cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами.

Текст программы (cm_timer_6.py):

```
import time
from contextlib import contextmanager
print("\nДубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3\n")

class cm_timer_1:
    """Контекстный менеджер, который считает время работы блока кода и
    выводит его на экран,
    основанный на классе"""
    def __enter__(self):
        self.start_time = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(cm_timer_1.__name__, time.time() - self.start_time)

@contextmanager
def cm_timer_2():
    """Контекстный менеджер, который считает время работы блока кода и
    выводит его на экран,
    основанный на библиотечной функции"""
    start_time = time.time()
    yield
    print(cm_timer_2.__name__, time.time() - start_time)

if __name__ == '__main__':
    with cm_timer_1():
        print('\n1-й способ')
        time.sleep(5.5)

    with cm_timer_2():
        print('\n2-й способ')
        time.sleep(5.5)
```

Результат (cm_timer_6.py):

```
Дубянский Антон Игоревич, ИУ5Ц-71Б, Лаб №3

1-й способ
cm_timer_1 5.500710964202881

2-й способ
cm_timer_2 5.50054931640625
```

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 (process_data_7.py):

```
import json
from print_result_5 import print_result
from unique_3 import Unique
from field_1 import field
from gen_random_2 import gen_random
from cm_timer_6 import cm_timer_1

#path = '/C:/Users/MSI/PycharmProjects/Lab-3/data_light.json'
path = "../data/data_light.json"

#with open(path) as f:
#    data = json.load(f)

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

@print_result
def f1(arg):
    return sorted(Unique(field(arg, 'job-name'))))

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

@print_result
def f3(arg):
```

```

        return list(map(lambda x: x + ' с опытом Python', arg))

@print_result
def f4(arg):
    gen_salary = list(gen_random(len(arg), 100000, 200000))
    work_and_salary = list(zip(arg, gen_salary))
    return list(map(lambda x: x[0] + ', зарплата ' + str(x[1]) + ' руб',
work_and_salary))

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

```

Результат (process_data_7.py):

F1:

```

f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь – моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
Агент по привлечению юридических лиц

```

F2:

```
f2
Программист
Программист / Senior Developer
Программист 1C
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программист-разработчик информационных систем
```

F3:

```
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
```

F4:

```
f4
Программист с опытом Python, зарплата 111144 руб
Программист / Senior Developer с опытом Python, зарплата 199330 руб
Программист 1C с опытом Python, зарплата 192879 руб
Программист C# с опытом Python, зарплата 149040 руб
Программист C++ с опытом Python, зарплата 184792 руб
Программист C++/C#/Java с опытом Python, зарплата 142370 руб
Программист/ Junior Developer с опытом Python, зарплата 171672 руб
Программист/ технический специалист с опытом Python, зарплата 127564 руб
Программист-разработчик информационных систем с опытом Python, зарплата 177472 руб
cm_timer_1 0.09873628616333008
```