

**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 лабораторным работам №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-32Б
Сайфутдинов Р.И.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Описание задания

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

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.

Файл `field.py`

```
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):
    if len(args) == 1:
        for item in items:
            if args[0] in item and item[args[0]] is not None:
                yield item[args[0]]
    else:
        for item in items:
            result = {arg: item[arg] for arg in args if arg in item and item[arg]
is not None}
            if result:
                yield result
```

```
for item in field(goods, 'title', 'price'):
    print(item)
```

```
PS C:\Users\user\Desktop\lab3_4> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe
eld.py
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
PS C:\Users\user\Desktop\lab3_4> █
```

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

Файл `gen_random.py`

```
import random

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

print(gen_random(5, 1, 3))
```

```
PS C:\Users\user\Desktop\lab3_4> & C:/Users/user/AppData
n_random.py
[3, 3, 1, 1, 2]
```

Задача 3

- 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.

Файл `unique.py`

```
from gen_random import gen_random

class Unique:
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.items = list(items)
        self.index = 0
        self.unique_items = []

    def _get_key(self, item):
        if self.ignore_case and isinstance(item, str):
```

```

        return item.lower()
    return item

    def __next__(self):
        while self.index < len(self.items):
            current_item = self.items[self.index]
            key = self._get_key(current_item)
            if key not in self.unique_items:
                self.unique_items.append(key)
                self.index += 1
                return current_item
            else:
                self.index += 1
        raise StopIteration

    def __iter__(self):
        return self

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
for i in Unique(data):
    print(i)
print()
data_random = gen_random(10,1,3)
for i in Unique(data_random):
    print(i)
print()
data_str = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
for i in Unique(data_str):
    print(i)
print()
for i in Unique(data_str, ignore_case=True):
    print(i)

```

PS C:\Users\user\Desktop\lab3_4> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe .\unique.py

[1, 1, 2, 2, 2]

1

2

3

1

2

a

A

b

B

a

b

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.

Файл sort.py

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

if __name__ == '__main__':
    result = sorted(data, key=abs, reverse=True)
    print(result)

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)
PS C:\Users\user\Desktop\lab3_4> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe .\sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

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 выводиться в столбик через знак равенства.

Файл print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(result, list):
            for item in result:
                print(item)
        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()

```

```

PS C:\Users\user\Desktop\lab3_4> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c
int_result.py
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS C:\Users\user\Desktop\lab3_4>

```

Задача 6

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 (на основе класса и с использованием библиотеки `contextlib`).

Файл `cm_timer.py`

```

import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):

```

```

        self.end_time = time.time()
        cool_time = self.end_time - self.start_time
        print(f"time: {cool_time}")

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    end_time = time.time()
    cool_time = end_time - start_time
    print(f"time: {cool_time}")

```

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.

Файл `process_data.py`

```

import json
from unique import Unique
from cm_timer import cm_timer_1
from field import field
from print_result import print_result
from gen_random import gen_random

path = "code/lab3-4_code/data_light.json"

with open(path) as f:

```

```
data = json.load(f)

def programmer_filter(data):
    return list(filter(lambda x: x.startswith("Программист"), data))

def unique_things(data):
    return list(Unique(field(data, "job-name")))

def add_python_exp(data):
    return list(map(lambda x: f"{x}, с опытом Python", data))

def add_salary(data):
    salaries = gen_random(2000, 100000, 200000)
    return [f"{job}, зарплата {salary} руб." for job, salary in zip(data, salaries)]

@print_result
def f1(arg):
    return unique_things(arg)

@print_result
def f2(arg):
    return programmer_filter(arg)

@print_result
def f3(arg):
    return add_python_exp(arg)

@print_result
def f4(arg):
    return add_salary(arg)

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


Программист C++/C#/Java с опытом Python
Программист 1C с опытом Python
Программист-разработчик информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python, зарплата 117224 руб.
Программист C++/C#/Java с опытом Python, зарплата 196903 руб.
Программист 1C с опытом Python, зарплата 130847 руб.
Программист-разработчик информационных систем с опытом Python, зарплата 117240 руб.
Программист C++ с опытом Python, зарплата 157738 руб.
Программист/ Junior Developer с опытом Python, зарплата 167587 руб.
Программист / Senior Developer с опытом Python, зарплата 123653 руб.
Программист/ технический специалист с опытом Python, зарплата 101665 руб.
Программист C# с опытом Python, зарплата 116638 руб.
time: 0.0062541961669921875