



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»

ДИСЦИПЛИНА:

«БКИТ»

Лабораторная работа № 3

Студент Бабаян А.А. ИУ5Ц-52Б

(И.О. Фамилия) (Группа)

(Подпись, дата)

Преподаватель Гапанюк Ю.Е.

(И.О. Фамилия)

(Подпись, дата)

1. Общее 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.

2. Задача 1

Описание:

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.

Код:

```
def field(items, *args):
    assert len(args) > 0

    for item in items:
        dict = {arg: item.get(arg) for arg in args if item.get(arg)}

        if len(dict) == 0: continue

        if len(args) == 1:
            yield dict[args[0]]
        else:
            yield dict

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': 'Стул', 'color': 'black', 'price': None}
]

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
```

Скрины:

```
C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>field.py
['Ковер', 'Диван для отдыха', 'Стул']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}, {'title': 'Стул'}]
```

3. Задача 2

Описание:

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`

Код:

```
from random import randint

def gen_random(num_count, begin, end):
    return (randint(begin, end) for _ in range(num_count))

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

Скриншоты:

```
C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>gen_random.py
[2, 1, 2, 3, 2]
```

4. Задача 3

Описание:

- 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.

Код:

```
from gen_random import gen_random

class Unique(object):

    def __init__(self, items, **kwargs):

        self.items = items

        if kwargs:
            seen, result = set(), []
            for item in self.items:
                if type(item) == str:
                    if str(item.lower()) not in seen:
                        seen.add(item.lower())
                        result.append(item)
                else:
                    if item not in seen:
                        seen.add(item)
                        result.append(item)
            else:
                result = list(set(self.items))

        self.items = result

    def __next__(self):
        self.items += 1
        return (self.items)

    def __iter__(self):
        return (el for el in self.items)

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
print(data1)
'''for i in Unique(data, ignore_case=True):
    print(i)'''
un1 = Unique(data1)
for i1 in un1:
    print(i1, end=' ')
print('\n')

data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
print(data2)
un2 = Unique(data2)
for i2 in un2:
    print(i2, end=' ')
print('\n', end='')

un3 = Unique(data2, ignore_case=True)
for i2 in un3:
    print(i2, end=' ')
print('\n')
```

Скрины:

```
C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>unique.py
[3, 1, 2, 2, 2]
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2

['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
b A a B
a b
```

5. Задача 4

Описание:

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.

Код:

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

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

    result_with_lambda = sorted(data, reverse=True, key=lambda el: abs(el))
    print(result_with_lambda)
```

Скриншоты:

```
C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

6. Задача 5

Описание:

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.

Код:

```
def print_result(func_to_decorate):

    def decorated_func(*args):
        print(func_to_decorate.__name__)
        result = func_to_decorate(*args)
```

```

        if type(result) is list:
            for i in result:
                print(i)
        elif type(result) is dict:
            for i in result:
                print(i, result.get(i), sep=' = ')
        else:
            print(result)

    return result
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]

def main():
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()

```

Скрины:

```

test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
!!!!!!!

```

7. Задача 6

Описание:

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.

Код:

```
from time import sleep, time
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.start = 0
        self.stop = 0
        self.res = 0

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop = time()
        self.res = self.stop - self.start

        print(f"time: {self.res:0.5f} ")

@contextmanager
def cm_timer_2(*args, **kwds):
    start = time()

    yield

    print(f"time: {time() - start:0.5f} ")

with cm_timer_1():
    sleep(1.5)
with cm_timer_2():
    sleep(2.0)
```

Скрины:

```
C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>cm_timer.py
time: 1.50133
time: 2.00747
```

Задача 7

Описание:

- 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.

Код:

```
import json
import sys

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

path = 'data_light.json'

with open(path, encoding='utf8') as f:
    data = json.load(f)

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

@print_result
def f2(arg):
    return list(filter(lambda job: job.startswith('Программист'), arg))

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

@print_result
def f4(arg):
    arg = list(arg)
    salary = gen_random(len(arg), 100000, 200000)
```



```

    return [job + f', зарплата {salary} руб' for salary, job in zip(salary,
arg)]

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

```

Скриншоты:

```

C:\Users\79626\PycharmProjects\Laboratornaya3\lab_python_fp>process_data.py
['Ковер', 'Диван для отдыха', 'Стул']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}, {'title': 'Стул'}]
[3, 3, 1, 2, 3]
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2

['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
В А а б
а б

time: 1.50436
time: 2.01504
f1
<unique.Unique object at 0x0000024522AF9E40>
f2
Программист
Программист C++/C#/Java
Программист 1С
Программист-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#

```

```

f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1С 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 115711 руб
Программист C++/C#/Java с опытом Python, зарплата 193047 руб
Программист 1С с опытом Python, зарплата 109663 руб
Программист-разработчик информационных систем с опытом Python, зарплата 132404 руб
Программист C++ с опытом Python, зарплата 105200 руб
Программист/ Junior Developer с опытом Python, зарплата 117193 руб
Программист / Senior Developer с опытом Python, зарплата 136964 руб
Программист/ технический специалист с опытом Python, зарплата 165265 руб
Программист C# с опытом Python, зарплата 128986 руб
time: 0.04760

```