

**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»  
Лабораторная работа №3-4

Выполнил:

Студент ИУ5-34Б

Флоринский В. А.

Подпись и дата:

Проверил:

Преподаватель каф. ИУ5

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023 г.

## Задание 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.

## Текст программы.

### field.py:

```
def field(items, *spec, key=0):
    assert len(spec) > 0, "Параметры обязательно указывать"
    if len(spec) == 1:
        ans = []
        for i in range(len(items)-1):
            if items[i].get(spec[0]) != None : ans.append(items[i].get(spec[0]))
            if items[i+1].get(spec[0]) != None : ans.append(items[i+1].get(spec[0]))
        return ans
    else:
        for it in items:
            tdict = {}
            for sp in spec:
                if it.get(sp) != None : tdict[sp] = it.get(sp)
            if len(tdict.keys()) != 0 : return(tdict)

if __name__ == "__main__":
    cars = [
        {'name' : 'bmw', 'price' : 10000, 'max_speed' : 300, 'hpower' : 650},
        {'name' : 'audi', 'price' : 9000, 'max_speed' : 250, 'hpower' : 400,
        'color' : 'red'},
        {'name' : 'mercedes', 'price' : 15000, 'max_speed' : 270, 'hpower' :
        4000, 'color' : 'green'}
    ]

    print(field(cars, 'name', 'price'))
```

## Примеры работы.

```
Sem\LR3\lab_python_fp\field.py"
{'name': 'bmw', 'price': 10000}
PS C:\Users\vladi\OneDrive\Рабочий
```

## Задание 2.

### 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`

## Текст программы.

### gen\_random.py:

```
from random import randint

def gen_random(count_, min_, max_):
    arr = [randint(min_, max_) for i in range(count_)]
    return arr

if __name__ == "__main__":
    arr = gen_random(5, 1, 3)
    print(arr)
```

## Примеры работы.

```
Sem\LR3\lab_python_fp\gen_random.py
[2, 2, 2, 2, 1]
PS C:\Users\vladi\OneDrive
```

## Задание 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.

## Текст программы.

### Unique.py:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = items
        self.curr = 0
        if len(list(kwargs.keys())) == 0 or list(kwargs.values()) == [False]:
            self.ign = False #a != A
            self.set_arr = list(set(self.items))
        elif list(kwargs.values()) == [True]:
            self.ign = True #a == A
            self.set_arr = []
```

```

        for i in self.items:
            if (type(i) is str) and (i.upper() not in self.set_arr and
i.lower() not in self.set_arr):
                self.set_arr.append(i)
            elif i not in self.set_arr and (type(i) is not str):
self.set_arr.append(i)

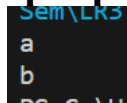
    def __next__(self):
        if self.curr < len(self.set_arr):
            res = self.set_arr[self.curr]
            self.curr += 1
            return res
        raise StopIteration

    def __iter__(self):
        return self

if __name__ == '__main__':
    arr = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    #arr = [1,2,2,3,3,1]
    it = Unique(arr, ignore_case = True)
    for el in it:
        print(el)

```

## Примеры работы.



## Задание 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.

## Текст программы.

sort.py:

```

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

if __name__ == '__main__':
    result_with_lambda = sorted(list(map(lambda i: abs(i), data)), reverse=True)
    print(result_with_lambda)

    result = sorted(list(map(abs, data)), reverse=True)
    print(result)

```

## Примеры работы.

```
Sem\LR3\lab_python_fp\sort.py"
[123, 100, 100, 30, 4, 4, 1, 1, 0]
[123, 100, 100, 30, 4, 4, 1, 1, 0]
PS C:\Users\vladi\OneDrive\Рабочий стол
```

## Задание 5.

### 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.py:

```
def print_result(func):
    def wrapper(*args, **kwargs):
        print(func.__name__)
        temp = func(*args, **kwargs)
        if type(temp) is list:
            for el in temp: print(el)
        elif type(temp) is dict:
            for key,value in temp.items(): print(f"{key} = {value}")
        else: print(temp)
        return temp
    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()

```

## Результат работы.

```

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

```

## Задание 6.

### 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 выводиться `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`).

## Текст программы.

### cm\_timer.py:

```

from contextlib import contextmanager
import time

class Cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        delta_time = time.time() - self.start_time
        print(f"time: {round(delta_time, 3)}")

@contextmanager
def Cm_timer_2():
    start_time = time.time()
    try:
        yield
    finally:

```

```

        delta_time = time.time() - start_time
        print(f"time: {round(delta_time, 3)}")

if __name__ == '__main__':
    with Cm_timer_1():
        time.sleep(5.5)

    with Cm_timer_2():
        time.sleep(5.5)

```

## Результат работы.

```

Sem\LR3\lab_python_fp\cm_timer.py"
time: 5.501
time: 5.501
PS C:\Users\vladi\OneDrive\Рабочий

```

## Задание 7.

### 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.py:

```

import json
from random import randint
from unique import Unique
from print_result import print_result
from field import field
from gen_random import gen_random
from cm_timer import Cm_timer_1

```

```

path = "C:\\Users\\vladi\\OneDrive\\Рабочий
стол\\LR3thSem\\LR3\\lab_python_fp\\data_light.json"

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

@print_result
def f1(arg):
    return sorted([el for el in Unique(field(arg, 'job-name'), ignore_case =
True)])

@print_result
def f2(arg):
    return list(filter(lambda x: x.split()[0] == "программист", arg))

@print_result
def f3(arg):
    return list(map(lambda x: f"{x} с опытом Python", arg))

@print_result
def f4(arg):
    return [f"{man}, зарплата {sal} руб." for man, sal in zip(arg,
gen_random(len(arg), 100000, 200000))]

if __name__ == "__main__":
    with Cm_timer_1():
        f4(f3(f2(f1(data))))

```

## Результат работы.

### Результат функции f1():

```

Специалист по проведению историко-культурной экспертизы объектов культурного наследия
Специалист по продажам
Специалист по продажам (интернет, ТВ, телефония)
Специалист по промышленной безопасности оборудования, работающего под давлением
Специалист по работе с клиентами
Специалист по работе с крупными клиентами
Специалист по работе с персоналом
Специалист по ремонту полиграфического оборудования
Специалист по связям с общественностью волонтер
Специалист по сертификатам и документообороту
Специалист по сертификации переоборудованных автотранспортах средств
Специалист по социальной работе

```

Выводит все профессии без повторений



Результаты остальных функций:

```
юрист  
f2  
программист  
программист 1С  
f3  
программист с опытом Python  
программист 1С с опытом Python  
f4  
программист с опытом Python, зарплата 154914 руб.  
программист 1С с опытом Python, зарплата 100847 руб.  
time: 0.768
```