# Задачи на функциональное программирование 2

**Задача 1.**
Решите задачу размещения N королев на шахматной доске NxN, чтобы ни одна королева не угрожала другой с использованием генератора. Каждое обращение к генератору должно давать позицию на доске следующей королевы.

**Задача 2.**
Напишите на основе использования генераторов решение задачи тура шахматного Коня (поиск маршрута, который ведет Коня на каждый квадрат шахматной доски NxN без посещения какого-либо квадрата дважды). Каждое обращение к генератору должно давать на доске новую позицию коня.

**Замечание: Достаточно выполнить одну из первых двух задач.**

**Задача 3.**
Напишите функцию высшего порядка, которая получает в качестве аргумента две функции первого порядка:

- Функцию для преобразования текста сообщения в верхний регистр.
- Функцию для преобразования текста сообщения в нижний регистр.

**Пример вывода:**
РЕГИСТР ПРЕОБРАЗОВАН ФУНКЦИЕЙ, ПОЛУЧЕННОЙ В КАЧЕСТВЕ АРГУМЕНТА
регистр преобразован функцией, полученной в качестве аргумента

**Задача 4.**
Напишите функцию высшего порядка, которая может принимать функции **float()**, **hex()**, **bin()**, **str()** и содержит вложенную функцию первого порядка, которая конвертирует полученное от пользователя целое число **n** в соответствии с полученными функциями.
**Пример вывода для n = 25:**

```
Преобразуем полученное число 25 в типы:
float => 25.0
bin => 0b11001
hex => 0x19
str => 25
```

**Задача 5.**
Напишите функцию высшего порядка, которая:

- Определяет, состоит ли полученный от пользователя список из четного или нечетного количества чисел.
- Возвращает функцию умножения элементов списка (в которой не используется **math.prod()**), если количество чисел **четное**.
- Возвращает функцию суммирования (в которой не используется встроенная функция **sum()**), если количество чисел **нечетное**.

**Пример ввода 1:**

```
8 9 3 5 1 3 8 2 9
```

**Вывод 1:**

```
Количество чисел нечетное, результат: 48
```

**Пример ввода 2:**

```
7 3 2 8 9 1 2 3 4 6
```

**Вывод 2:**

```
Количество чисел четное, результат: 435456
```

**Задача 6.**
Напишите собственный аналог функции **filter()**. Для отбора данных **my_filter()** должна, как и встроенная **filter()**, использовать **функцию-предикат**. Функция-предикат возвращает True или False в зависимости от критерия – в нашем случае это факт совпадения первой и последней букв слова в строке, полученной от пользователя.

**Пример ввода:**

крюк арбуз торт абрикос кулак барабан рупор господин томат мадам

**Вывод:**

крюк торт кулак рупор томат мадам

**Задача 7.**
Напишите собственный вариант функции-агрегатора **reduce()**. Функция при вызове должна получать:

1. Функцию первого порядка для проведения операции умножения или сложения.
2. Список чисел от пользователя.
3. Начальное значение – **0** для операции суммирования, **1** для операции умножения.

**Пример ввода:**

```
5 7 8 3 2 5 8 12 3 5 4 8 9
```

**Примеры вызова:**

```
print(my_reduce(add, my_list, 0))
print(my_reduce(mult, my_list, 1))
```

**Вывод:**

```
79
3483648000
```

**Задача 8.**
Вы знаете встроенную функцию **zip()** для параллельной итерации двух наборов данных. Напишите функцию высшего порядка, которая возвращает функции для:

1. Группировки параллельных элементов списков с помощью самописной **my_zip()**.
2. Конкатенации элементов, сгруппированных **my_zip()**.
3. Сложения элементов, сгруппированных **my_zip()**.

***Примечание:*** следует учесть, что получаемые от пользователя списки могут быть разной длины. Как и встроенная **zip()**, **my_zip()** должна ограничивать размер возвращаемого списка длиной **более короткого** набора данных.

**Пример ввода:**

```
5 8 9 8 3 12 3 5 5 4 0 9 6 1 23 6 12 30
4 5 6 2 3 2 5 9 4 12 9 3
```

**Вывод:**

```
(5, 4) (8, 5) (9, 6) (8, 2) (3, 3) (12, 2) (3, 5) (5, 9) (5, 4) (4, 12) (0,
9) (9, 3)
54 85 96 82 33 122 35 59 54 412 09 93
9 13 15 10 6 14 8 14 9 16 9 12
```

**Задача 9.**
Напишите программу, которая:

- получает от пользователя список слов и букву на отдельных строках;
- с помощью самописной функции **my_filter()** определяет, какие слова начинаются с полученной буквы;
- выводит индексы и слова отфильтрованного списка с помощью самописной функции **my_enumerate()**.

**Пример ввода:**

```
абрикос бюро газета банк коробка стол бобр ноутбук блокнот баланс абажур
б
```

**Вывод:**

```
1-е слово нового списка - бюро
```

```
2-е слово нового списка - банк
3-е слово нового списка - бобр
4-е слово нового списка - блокнот
5-е слово нового списка - баланс
```

**Задача 10.**

Напишите функцию высшего порядка, которая принимает две одно-аргументные функции первого порядка и возвращает новую функцию. Эта функция принимает аргумент **x** и применяет к нему полученные функции в следующем порядке:

```
function1(function2(x))
```

К примеру, если передать в функцию высшего порядка эти функции:

```
def add(x):
    return x + 10

def multiply(x):
    return x * 5
```

И вызвать функцию так:

```
print(super_function(add, float)('16'))
print(super_function(tuple, multiply)((3, 4, 5)))
print(super_function(str, multiply)('55'))
print(super_function(list, multiply)((1, 2, 3)))
```

Результат будет выглядеть следующим образом:

```
26.0
(3, 4, 5, 3, 4, 5, 3, 4, 5, 3, 4, 5, 3, 4, 5)
5555555555
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

**Задача 11.**

Напишите декоратор, который будет подсчитывать количество **позиционных** и **именованных** аргументов, переданных в функцию первого порядка.

**Пример функции и вызова 1:**

```
@decorator_func
def names_and_age(age1, age2, age3, name1, name2, name3):
    return f'У меня есть три сестры: {name1}, ей {age1} лет; {name2},
            ей {age2} лет; {name3} - ей {age3} лет\n'
print(names_and_age(12, 15, 13, name1='Света', name2='Маша', name3='Ира'))
```

**Вывод 1:**

```
Функция получила позиционных аргументов: 3, именованных аргументов: 3
У меня есть три сестры: Света, ей 12 лет; Маша, ей 15 лет; Ира - ей 13 лет
```

**Пример функции и вызова 2:**

```
@decorator_func
def position_and_salary(sal1, sal2, sal3, sal4, pos1, pos2, pos3, pos4):
    return f'{pos1} получает {sal1} тыс, {pos2} получает {sal2} тыс,
            {pos3} получает {sal3} тыс, {pos4} получает {sal4} тыс\n'
print(position_and_salary(320, 150, 230, 170, pos1='разработчик',
pos2='тестировщик', pos3='девопс', pos4='сисадмин'))
```

**Вывод 2:**

```
Функция получила позиционных аргументов: 4, именованных аргументов: 4
разработчик получает 320 тыс, тестировщик получает 150 тыс, девопс получает 230 тыс, сисадмин
получает 170 тыс
```

**Задача 12.**

Напишите декоратор, который будет измерять производительность функций, создающих список с помощью этих методов:

- range()
- списковое включение
- append()
- конкатенация

Среди показателей должны быть:

1. Время работы функции.
2. Текущее потребление памяти.
3. Пиковое потребление памяти.

**Пример вызова:**

```
print(make_list_with_range())
print(make_list_comprehension())
print(make_list_with_append())
print(make_list_concatenation())
```

**Вывод:**

```
Название функции: make_list_with_range
Использованный метод: range()
Текущее потребление памяти: 0.290164 мб
Пик использования памяти: 2.289118 мб
Операция заняла: 0.112532 секунд
Функция make_list_with_range завершила работу
-----------------------------------------------
Название функции: make_list_comprehension
Использованный метод: list comprehension
Текущее потребление памяти: 0.000930 мб
Пик использования памяти: 1.947573 мб
Операция заняла: 0.085460 секунд
Функция make_list_comprehension завершила работу
-----------------------------------------------
Название функции: make_list_with_append
Использованный метод: append()
Текущее потребление памяти: 0.000582 мб
Пик использования памяти: 1.947229 мб
Операция заняла: 0.100597 секунд
Функция make_list_with_append завершила работу
-----------------------------------------------
```

**Задача 13.**

Эмуляция гонки трёх автомобилей.
Позиции автомобилей вычисляются через дискретные интервалы времени.
В каждый момент времени машина случайным образом либо двигается вперёд, либо нет.
Через каждый следующий интервал времени программа выводит пройденный автомобилями путь. Через пять промежутков времени гонка заканчивается.
Нефункциональное решение задачи может выглядеть так:

```
from random import random

time = 5
car_positions = [0, 0, 0]

while time:
    time -= 1   # decrease time
    print ('_____________________________')
    for i in range(len(car_positions)):
```

```
        if random() > 0.3:     # move car
            car_positions[i] += 1
        print ('Start|', '-' * car_positions[i])   # draw car
```

Переработайте программу, определив функции:

- move_cars – вычисление позиции автомобиля,
- draw_car – печать позиции автомобиля
- run_step_of_race – вычисление позиций автомобилей в следующий момент времени
- draw – вывод позиций всех автомобилей,
- race – запуск гонки автомобилей.

С помощью этих функций перепишите программу в функциональном стиле.
Все моделирование гонки должно выполняться композиций функций.
Цикл моделирования нужно заменить на рекурсию.
В вашем коде не должно быть расшаренных глобальных переменных time и car_positions, все взаимодействия должны происходить через передачу параметров в функции, переменные не должны меняться внутри функций, все значения возвращаются из функций через return.

**Задача 14.**
Следующий код программы изменяет неправильное значения в словаре для ключа 'name', неправильную страну происхождения, написание значений для ключа 'name', а также булевские значения ключа.

```
data = [{'name': 'wonderful sunset on the volga', 'country': 'UK',
       'active_state': False},
         {'name': 'fantastic sunrise in .crimea. ', 'country': 'Germany',
       'active_state': False},
         {'name': 'good rain in Vladivostok.', 'country': 'Spain',
       'active_state': True}]

def transform_data(data):
    for band in data:
        band['country'] = 'Russia'
        band['name'] = band['name'].replace('.', '')
        band['name'] = band['name'].title()
        band['active_state'] = not band['active_state']

transform_data(data)
print(data)
```

Разработайте версию программы на основе использования конвейера, где функция pipeline() будет перебирать элементы группы по одному, и передавать их функциям преобразования, вроде set_new_country(). После применения очередной функции ко всем элементам группы, pipeline() должна делать из них список и передавать следующей функции.

**Задача 15.**
Решите предыдущую задачу с использованием короутинов, которые передают обработку данных по цепочке.