

# Шпаргалка. Прога КЕГЭ. ШКОЛКОВО.

## Задание №2

```
# 1 СПОСОБ
print('x y z w f')
for x in range(2):
    for y in range(2):
        for z in range(2):
            for w in range(2):
                if (логическая функция == 0 или 1):
                    print(x, y, z, w)
# 1) Получаем набор х, у, z, w для которых
# функция ложна или истинна
# в зависимости от условия задачи
# 2) Аккуратно сопоставляем с таблицей
# в условии

# 2 СПОСОБ
from itertools import product
print('x y z w')
for x, y, z, w in product((0, 1), repeat=4):
    # repeat - кол-во переменных
    if логическая функция == 0 или 1:
        print(x, y, z, w)
```

## Задание №8

Прототип: "Сколько 4-буквенных слов сможет составить N из букв А, Б, В, Г, если при этом А не должна стоять на первом месте и должно быть ровно две буквы Б в слове?"

### ВЛОЖЕННЫМИ ЦИКЛАМИ

```
letters = 'АБВГ' #сюда записываем буквы
s = set() #создаем множество, дабы не запутаться со счетчиком
for q in letters:
    for w in letters:
        for e in letters:
            for r in letters: #создаем столько циклов, сколько букв в слове
                word = q+w+e+r
                if (word[0] != 'А' and
                    word.count('Б') == 2):
                    #прописываем все условия
                    s.add(word)
print(len(s))
```

### ЧЕРЕЗ ITERTOOLS

```
from itertools import product
ans = 0
comb = set(product('АБВГ', repeat = 4))
for i in comb:
    s = "join(i)"
    if s[0] != 'А' and s.count('Б') == 2:
        ans += 1
print(ans)
```

Прототип: "Все 4-буквенные слова, составленные из букв Б, Ш, А, В, записаны в алфавитном порядке. Вот начало списка:

1. АААА  
2. АААБ  
3. АААШ  
4. АААВ  
5. ААБА  
Укажите номер слова "ББББ"

```
letters = 'АБШВ' #записываем буквы в том порядке, в каком они стоят в нумерации
k = 0 #создаем счетчик с нумерацией с нуля
for q in letters:
    for w in letters:
        for e in letters:
            for r in letters: #создаем столько циклов, сколько букв в слове
                word = q+w+e+r
                k += 1 #добавляем в счетчик +1,
                когда записываем новое слово
                if word == 'ББББ':
                    print(k)
```

## Задание №6

```
# Запускаем перебор значений,
# которое вводится в программу
for t in range(-1000, 1000):
    # промежуток перебора может
    # быть другим
    x = t # если в коде вводится
    # переменная x, то
    # присваиваем её t
    # сюда вставляем код программы,
    # где вводится переменная x
    # ...
    if (n == 60): # если по коду
        выводится n и по условию
        вывелось 60
        print(t)
```

## Задание №14

```
s = *сюда выражение*
x = int(s)
r = *номер СС до 10*
result = ""
while x > 0:
    result = str(x % r) + result
    x //= r

print(result.count('1')) # количество
# символов '1'
result = list(result) # находим сумму и
# произведение всех
цифр в числе
summ = 0
mul = 1
for elem in result:
    summ += int(elem)
    mul *= int(elem)
print(summ, mul)
```

```
# Универсальная функция перевода
# в любые СС
def convert_base(num, to_base=10,
                 from_base=10):
    if isinstance(num, str): # проверяем,
        # является ли
        num строкой
    n = int(num, from_base)
    else:
        n = int(num)
    alphabet = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    if n < to_base:
        return alphabet[n]
    return convert_base(n // to_base,
                        to_base) + alphabet[n % to_base]
```

#Встроенная функция перевода из
любой СС в десятичную
n = int(a, 3) #перевод из троичной
в десятичную.
Важное условие: а
должно быть строкой

## Задание №15

### НЕРАВЕНСТВА

```
def f(x, y, A):
    return *функция из условия*

for A in range(1, 200):
    flag = True # предполагаем, что с этим
    # все хорошо
    for x in range(1, 200):
        for y in range(1, 200):
            if not f(x, y, A): # в условии нужно
                # истинно, поэтому
                # если
                # при этом A - ложно, выходим
                # с флагом False
                flag = False
                break
            if not flag:
                break
        if flag:
            ans = A # максимальный ответ
            # будет присвоен на
            # последней операции
print(ans)
```

## Задание №12

### ИСПОЛНИТЕЛЬ РЕДАКТОР

```
s = *заданное выражение*
while s.find(s1) != -1:
    # while - пока
    # s.find(s1) - находит индекс первого
    # вхождения строки s1 в s
    # если не нашёл - возвращает -1
    # вместо s1 можно писать нужные символы
    # в кавычках, как ниже
    while s.find('5') != -1:
        s = s.replace('5', '77', 1)
    # s.replace(s1, s2, 1) заменяет первое
    # вхождение s1 в s на s2
    print(sum([int(s[i]) for i in range(len(s))]))
    # строка выше печатает сумму цифр
    # полученного числа
```

### ИСПОЛНИТЕЛЬ ЧЕРТЕЖНИК

Условие: чертёжнику был дан для исполнения следующий алгоритм:  
НАЧАЛО  
сместиться на (-30, -110)  
ПОВТОРИ N РАЗ  
сместиться на (a,b)  
сместиться на (76,-93)  
КОНЕЦ ПОВТОРИ  
сместиться на (0,5)  
КОНЕЦ

Вопрос: чему равно максимальное значение N при котором найдутся такие значения чисел a и b что после выполнения программы  
Чертёжник возвратится в исходную точку?

### Решение

Чертёжник остановился в точке (0, 0), значит конечные координаты равны этим числам.

$$x = -30 + N \cdot (a + 76) = 0$$

$$y = -110 + N \cdot (b - 93) + 5 = 0$$

Перенесём все числа в левые части:

$$N \cdot (a + 76) = 30 = 2 \cdot 3 \cdot 5$$

$$N \cdot (b - 93) = 105 = 3 \cdot 5 \cdot 7$$

Для выбранных N должны подобраться целые значения в скобках. Значит N должно быть делителем чисел 30 и 105. Максимальное такое число - 15.

### ПОРАЗРЯДНАЯ КОНЬЮНКЦИЯ

```
def f(x, a):
    return *функция*

for a in range(0, 300):
    p = True # предполагаем всё хорошо
    for x in range(0, 300):
        if f(x, a) == False: # если не хорошо -
            # противоречит условию
            задачи:
            # выходим флагом False
            p = False
            break
    if p == True:
        print(a)
```

# Задание №15 (продолжение)

## ЧИСЛОВОЙ ОТРЕЗОК

### #1 СПОСОБ

```
def inn(x, A):
    return A[0] <= x <= A[1] # функция принадлежности числа к отрезку
    # предполагаем, что A - это массив,
    # где 0-ой элемент - начало отрезка
    # a 1-ый элемент - конец отрезка

def f(x, A):
    P = [a1, a2]
    Q = [a2, b2]
    K = [a3, b3]
    return *функция*

borders = [0, 0]
minim = 100000
k = 5 # воспользуемся более точным приближением при подборе
for a in range(0, 80 * k):
    for b in range(a, 80 * k):
        A = [a / k, b / k]
        good = True # предполагаем, что при этом A всё хорошо
        for x in range(0, 100 * k):
            if not f(x/k, A): # оказалось не хорошо, по условию нужно истинно,
                # а получили ложь, выходим с флагом False
                good = False
                break
        if good: # всё хорошо, отрезок такой походит
            if A[1] - A[0] <= minim:
                minim = A[1] - A[0] # сохраним длину
                borders = A.copy() # сохраним сам отрезок
print(minim)
print(borders)
```

### #2 СПОСОБ

```
# Задаем отрезки
p = range(10, 29 + 1)
q = range(13, 18 + 1)

# Для максимального отрезка: создаем множество со значениями от 1 до 1000
a = set(range(1, 1000))
# Для минимального отрезка: создаем пустое множество
a = set()

for x in range(1, 1000):
    if логическая функция == 0 или 1:
        # Для максимального отрезка: выкидываем такие иксы из отрезка
        a.remove(x)
        # Для минимального отрезка: добавляем такие иксы в отрезок
        a.add(x)

    # Количество элементов в отрезке
    print(sorted(a))
    # Максимальная длина отрезка
    print(sorted(a)[-1] - sorted(a)[0])

    # 3 способ (в действительных числах)
    # Задаем отрезки
    p = [i / 10 for i in range(50, 300 + 1)]
    q = [i / 10 for i in range(140, 240 + 1)]

    # Для максимального отрезка: создаем множество со значениями от 1 до 1000
    a = set([i / 10 for i in range(10000)])
    # Для минимального отрезка: создаем пустое множество
    a = set()
```

```
for x in [i / 10 for i in range(10000)]:
    if логическая функция == 0 или 1:
        # Для максимального отрезка: выкидываем такие иксы из отрезка
        a.remove(x)
        # Для минимального отрезка: добавляем такие иксы в отрезок
        a.add(x)

    # Количество элементов в отрезке
    print(sorted(a))
    # Максимальная длина отрезка
    print(sorted(a)[-1] - sorted(a)[0])
```

# Задание №19-21

## ОДНА КУЧА

```
from functools import lru_cache
def moves(heap):
    return heap + 1, heap * 2

@lru_cache(None)
def game(heap):
    if heap >= 31:
        return 0
    steps = [game(x) for x in moves(heap)]
    # Если хотя бы один ход приводит нас к победе
    if any(x % 2 == 0 for x in steps):
        # Возвращаем этот ход
        return 1 + min(x for x in steps if x % 2 == 0)
    # Максимально оттягиваем свое поражение
    return max(steps) + 1

for s in range(1, 31):
    print(s, game(s), [game(x) for x in moves(s)])
```

## ДВЕ КУЧИ

```
from functools import lru_cache
def moves(heap):
    a, b = heap
    return (a + 1, b), (a, b + 1), (a * 2, b), (a, b * 2)

@lru_cache(None)
def game(heap):
    if sum(heap) >= 47:
        return 0
    steps = [game(x) for x in moves(heap)]
    # Если хотя бы один ход приводит нас к победе
    if any(x % 2 == 0 for x in steps):
        # Возвращаем этот ход
        return 1 + min(x for x in steps if x % 2 == 0)
    # Максимально оттягиваем свое поражение
    return max(steps) + 1

for s in range(1, 43):
    print(s, game((4, s)), [game(x) for x in moves((4, s))])
```

```
from functools import lru_cache
def moves(heap):
    # Создаем массив, в который будем добавлять ходы которые можно совершить
    m = []
    if heap >= 1:
        m += [heap - 1]
    if heap >= 3:
        m += [heap - 3]
    return m

@lru_cache(None)
def game(heap):
    if heap <= 7:
        return 0
    steps = [game(x) for x in moves(heap)]
    # Если хотя бы один ход приводит нас к победе
    if any(x % 2 == 0 for x in steps):
        # Возвращаем этот ход
        return 1 + min(x for x in steps if x % 2 == 0)
    # Максимально оттягиваем свое поражение
    return max(steps) + 1

for s in range(100, 7, -1):
    print(s, game(s), [game(x) for x in moves(s)])
```

# Задание №22

```
# Запускаем перебор значений, которое вводится в программу
for t in range(-1000, 1000): # промежуток перебора
    x = t # если в коде вводится переменная x, то
          # присваиваем её t
    # сюда вставляем код программы, где вводится переменная x
    #
    if (n == 60 and k == 70): # если по коду выводится
        n и по условию вывелоось
        60 и 70
    print(t)
```

# Задание №23

## СТАНДАРТНЫЙ ПРОТОТИП

```
"Исполнитель N преобразует число на экране. У исполнителя есть две команды:
1. Умножить на 3
2. Прибавить 1

Сколько существует программ, для которых при числе 1 результат будет 28, при этом траектория вычислений содержит число 5 и не содержит число 17?

a = [0] * 84 #создаем массив на все ходы
a[1] = 1
for i in range(1, 5): #идем до "связующего узла", через
    который должны пройти все прогр.
    a[i * 2] += a[i]
    a[i * 3] += a[i]
for i in range(5, 84): #обнуляем все остатки путей, которые
    прошли не через 5
    a[i] = 0
for i in range(5, 28): #продолжаем от "связующего узла"
    if i == 17: #роверяем, что число не равно 17
        a[i] = 0
        a[i + 1] += a[i]
        a[i + 3] += a[i]
print(a[28])
```

## ИДЕЯ ДИНАМИКИ

# Задание №23 (прод.)

```
"Исполнитель N преобразует число на экране. У исполнителя есть две команды:
1. Умножить на 3
2. Прибавить 1

Сколько существует программ, для которых при числе 1 результат будет 28, при этом траектория вычислений содержит число 5 и не содержит число 17?

def f1(st, fn, flag, flag_number, exit_number):
    if st == fn and flag:
        return 1
    if st > fn or st == exit_number:
        return 0
    if st == flag_number: # поднимаем флаг,
        # если дошли до требуемого значения
        flag = True
        x = f1(st + 1, fn, flag, flag_number, exit_number)
        y = f1(st * 3, fn, flag, flag_number, exit_number)
        return x + y
    print(f1(1, 28, False, 5, 17))
```

## ИДЕЯ РЕКУРСИИ

### "СОДЕРЖИТ СТОЛЬКО-ТО КОМАНД"

```
"Исполнитель Щелчок преобразует число на экране. У исполнителя есть три команды:
1. Прибавить 1
2. Прибавить саму число
3. Прибавить само число

Сколько существует программ, для которых при исходном числе 1 результатом является число 28, при этом программа содержит 9 команд, траектория обязательно проходит через число 11 и проходит через число 13?"
```

### ДИНАМИКА

```
a = [0] * 10 for i in range(100): #создаем двумерный массив в первой ячейке будем
a[1][0] = 1 #хранить программы, во второй - команды
for i in range(1, 13):
    for j in range(9):
        if i == 11:
            a[i][j + 1] += a[i][j]
            a[i][j + 1] += a[i][j]
            a[i][j + 1] += a[i][j]
        for i in range(14, 100):
            for j in range(9):
                a[i][j + 9] = 0
                for i in range(13, 28):
                    for j in range(9):
                        a[i][j + 9] += a[i][j]
                        a[i][j + 9] += a[i][j]
                        a[i][j + 9] += a[i][j]
                        print(a[28][9])
```

### РЕКУРСИЯ

```
def f1(st, fn, flag, comm):
    if st > fn:
        return 0
    if st == 11:
        return 1
    if st == 13:
        flag = True
        if comm < 0:
            return 0
        if st > fn and flag and comm == 0:
            return 1
        return f1(st+3,fn,flag,comm-1)+f1(st+2,fn,flag,comm-1)
    print(f1(1, 28, False, 9))
```

# Задание №24

```
f = open('24.txt') # откроем файл
# Если задача на одну строку
s = f.readline() # считает одну строку
# Если задача на несколько строк, то
# проще их поместить в массив
a = f.readlines() # считываем несколько строк и кладем их в массив

len(s) # длина строки
s.count('AR') # число подстрок 'AR'
            # внутри строки s
s.find('AR') # поиск подстроки 'AR'
            # слева, вычисляет индекс первого вхождения, иначе -1
s.rfind('AR') # поиск подстроки 'AR'
            # справа, вычисляет индекс последнего вхождения, иначе -1
s.replace('AR', 'MO') # замена подстроки 'AR' на подстроку 'MO'
            # одно первое вхождение слова
```

```
# прототип поиска максимальной длинной цепочки одинаковых символов
s = f.readline()
cur_len = 1
max_len = 0
for i in range(len(s)):
    if s[i] == s[i + 1]:
        cur_len += 1
        max_len = max(max_len, cur_len)
    else:
        cur_len = 1
print(max_len)
```

Не забывайте, что иногда можно идти нестандартными путями в задаче, например, воспользоваться помощью .split() и считать разбитые подцепочки.

Файл состоит из символов А, В, С, Д, Е. Найдите количество символов между двумя самыми отдаленными буквами А, между которыми находятся только согласные буквы.

```
n = open('file.txt').readline()
a = n.split('A')
maxim = 0
Flag = True
for i in a: #идем по массиву
    for x in i: #теперь по строке
        if x not in 'BCD':
            Flag = False
        if Flag:
            maxim = max(maxim, len(i))
            Flag = True
print(maxim)
```

Важно помнить, что если вы используете .split() для задачи, где нужно подсчитывать максимальную длину, не содержащую что-то, рассмотрите боковые символы от разбитой строчки.

## НАЙТИ И ВЫВЕСТИ САМЫЙ ЧАСТО ПОВТОРЯЮЩИЙСЯ СИМВОЛ

Пример: "Найдите самый часто повторяющийся символ между символами A и R (в данном порядке). Если таких символов несколько, следует взять тот, что раньше по алфавиту."

### ЧЕРЕЗ МАССИВ С АЛФАВИТОМ

```
n = open("file.txt").readline()
letters = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
array = [0] * 26
for i in range(1, len(n)-1):
    if n[i-1] == 'A' and n[i+1] == 'R':
        array[letters.index(n[i])] += 1
for i in range(len(array)):
    if array[i] == max(array):
        print(letters[i])
        break
```

### ЧЕРЕЗ ORD, CHR

```
n = open("file.txt").readline()
array = [0] * 200
for i in range(1, len(n)-1):
    if n[i-1] == 'A' and n[i+1] == 'R':
        array[ord(n[i])] += 1
print(array)
for i in range(len(array)):
    if array[i] == max(array):
        print(chr(i))
        break
```

## 24 ПРОДОЛЖЕНИЕ

## Задание №25

```
# Функция нахождения
количество делителей числа
def count_div(n): # Счетчик всех
                  # делителей
    k = 2 # 1 и само число -
          # делители числа всегда
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            k += 1
            if n // i != i:
                k += 1
    return k
```

```
# Функция нахождения
отсортированного массива
нетривиальных делителей
def dividers(n):
```

```
    a = []
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            a.append(i)
            if n // i != i:
                a.append(n // i)
    a.sort()
    return a
```

```
# Функция проверки числа
на простоту
```

```
def is_prime(n):
    if n == 1: return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

```
# Функция нахождения
минимального делителя (после 1)
def min_divider(n):
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return i
```

## Задание №27

### ПОСЛЕДОВАТЕЛЬНОСТИ (ПРЕФ. СУММЫ)

Найти цепочку с макс. суммой, кратной 42

```
f = open('27.txt')
n = int(f.readline())
s = 0
ans = -10000000
mps = [10000000] * 42 #минимальные преф.
                     #суммы элементов,
                     #кратных всем числам
                     #до 42 вкл.
mps[0] = 0 #т.к. мин. преф. сумма,
            #кратная 42, является нулем
for i in range(n):
    s += int(f.readline()) #считаем текущ. сумму
    ans = max(ans, s - mps[s % 42]) #динамически
                                     #перезаписываем ответ
    mps[s % 42] = min(mps[s % 42], s) #кладем
                                         #минимальную преф.
                                         #сумму в нужную ячейку
print(ans)
```

### ДВИЖЕНИЕ ПО ОКРУЖНОСТИ

```
f = open('27.txt')
n = int(f.readline())
a = [int(i) for i in f]
s = [0] * n
current_sum = 0 #задаем первую полуокружность
for i in range(n//2):
    current_sum += a[i]
s[0] = current_sum
for i in range(1, n): #задаем следующие
                      #полуокружности
    s[i] = s[i - 1] - a[i - 1] + a[(i - 1 + n // 2) % n]
cost = 0
for i in range(n):
    cost += (min(i, n - i) * a[i]) #считаем стоимость
                                      #доставки по кругу
minim = cost
for i in range(1, n):
    cost = cost - s[i] + s[(i + n // 2) % n] #подсчитываем
                                                #стоимость доставки,
                                                #передвигая точки
                                                #старта
    if minim > cost:
        minim = cost
        ans = i
print(ans+1) #добавляем +1 в ответ, т.к.
             #индексация с нуля
```

### ОЧЕРЕДИ

#Количество пар, находящихся на расстоянии 5,  
произведение которых четно.

```
f = open('27.txt')
n = int(f.readline())
line = []
for i in range(4):
    line.append(int(f.readline()))
ans = 0
even = 0
odd = 0
for i in range(4, n):
    x = int(f.readline())
    if x % 2 == 0:
        ans += odd + even
    else:
        ans += even
#бросаем самого левого элемента на свалку
if line[0] % 2 == 0:
    even += 1
else:
    odd += 1
#сдвигаем очередь
for j in range(3):
    line[j] = line[j+1]
line[3] = x
print(ans)
```

# Шпаргалка. Теория. Информатика. Школково.

## Системы счисления

10cc	2cc	8cc	16cc
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

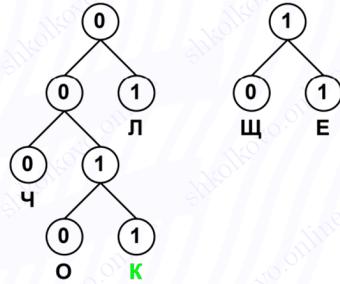
## Задание №4

Информатика. Щелчок. Условие Фано. 4 номер. Шпаргалка.

Прямое условие Фано означает, что ни одно кодовое слово не является началом другого кодового слова.

Обратное условие Фано означает, что ни одно кодовое слово не является концом другого кодового слова.

Чтобы соблюсти прямое условие Фано достаточно нарисовать двоичное дерево (смотри ниже):



Где 000 - кодовое слово у буквы Ч, 01 - у буквы Л, и так далее... (у обратного у Фано это будут 000 - у Ч, 10 - у Л, и т.д. Кодовые слова пишутся с конца ветки)

## Алгебра логики

### Варианты обозначений

Отрицание	not	!	$\neg$	$\bar{x}$
Конъюнкция	and	&	$\wedge$	
Дизъюнкция	or		$\vee$	
Импликация	$\rightarrow$	$\Rightarrow$		
Эквивалентность	$\equiv$	$\Leftrightarrow$		

Таблица основных функций

x	y	$x \vee 0$	$x \wedge 0$	$x \vee 1$	$x \wedge 1$	$\bar{x}$	$x \wedge y$	$x \vee y$	$x \equiv y$	$x \rightarrow y$	$\bar{x} \vee y$
0	0	0	0	1	0	1	0	0	1	1	1
0	1	0	0	1	0	1	0	1	0	1	1
1	0	1	0	1	1	0	0	1	0	0	0
1	1	1	0	1	1	0	1	1	1	1	1

### Законы

Двойного отрицания	$\bar{\bar{x}} = x$	
Переместительный	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
Сочетательный	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Распределительный	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
Идемпотентности	$x \vee x = x$	$x \wedge x = x$
Исключение третьего	$x \vee \bar{x} = 1$	$x \wedge \bar{x} = 0$
Поглощения	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$
де Моргана	$\neg(x \vee y) = \bar{x} \wedge \bar{y}$	$\neg(x \wedge y) = \bar{x} \vee \bar{y}$

## Задание №7

Биты, байты, килобайты...

8 бит = 1 байт

1024 байт =  $2^{10}$  байт =  $2^{13}$  бит = 1 Кбайт

$2^{10}$  Кбайт =  $2^{23}$  бит = 1 Мбайт

Все формулы для 7 задачи

Под объёмом здесь и далее подразумевается минимальное количество бит, необходимое для хранения объекта.

Объём изображения  $I = k \cdot i$ , где  $k$  – размер изображения,  $i$  – глубина кодирования.

$2^i$  – максимальное количество цветов в палитре с глубиной кодирования  $i$ .

Если изображений несколько, то суммарный объём  $S = N \cdot I$ , где  $N$  - количество изображений.

Объём музыкального файла  $I = f \cdot r \cdot k \cdot t$ , где  $f$  – частота дискретизации (в Гц),  $r$  – разрешение (глубина кодирования),  $k$  – количество каналов,  $t$  – время звучания (в секундах).

Объём видеодорожки  $I_{image} = I \cdot FPS \cdot t$ , где  $I$  – объём одного кадра (изображения),  $FPS$  - количество кадров в секунду,  $t$  - время (в секундах).

Объём видео  $W = I_{image} + I_{audio}$ , где  $I_{image}$  - объём видеодорожки, а  $I_{audio}$  - объём аудиодорожки.

## Задание №8

Правило умножения: если элемент  $a$  можно выбрать  $n$  способами, а элемент  $b$  -  $m$  способами (независимо от  $a$ ), то все пары  $a$  и  $b$  можно выбрать  $n \cdot m$  способами.

Правило сложения: если элемент  $a$  можно выбрать  $n$  способами, а элемент  $b$  -  $m$  способами (независимо от  $a$ ), то все пары  $a$  или  $b$  можно выбрать  $n + m$  способами.

$n!$  - количество способов переставить  $n$  объектов без повторений.

Например, количество способов переставить 3 элемента без повторений: 123, 132, 213, 231, 312, 321 или:  $3 \cdot 2 \cdot 1 = 3!$

$C_n^k = \frac{n!}{(n-k)!k!}$  - количество способов выбрать  $k$  объектов из  $n$  с учётом порядка.

$A_n^k = \frac{n!}{(n-k)!}$  - количество способов выбрать  $k$  объектов из  $n$  без учёта порядка

## Задание №9

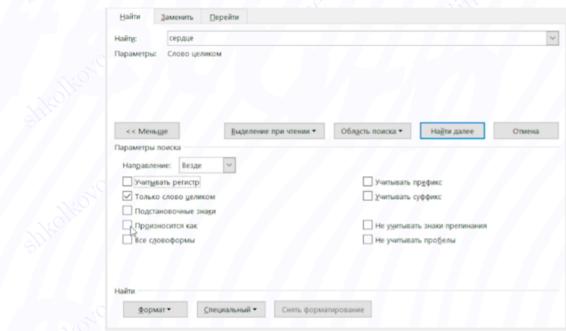
Действие функции	Синтаксис
Суммирует числа в заданном диапазоне	СУММ(Х1:Х2)
Количество непустых ячеек	СЧЁТ(Х1:Х2)
Подсчет сердного арифметического значения	СРЕДНЕЕ(Х1:Х2)
Возвращает максимальное значение в заданном диапазоне	МАКС(Х1:Х2)
Возвращает минимальное значение в заданном диапазоне	МИН(Х1:Х2)
Проверка условия	ЕСЛИ (лог_выражение; значение_если_истина; значение_если_ложь)
Количество непустых ячеек в указанном диапазоне, удовлетворяющих заданному условию	СЧЁТЕСЛИ (Х1:Х2; лог_выражение)
Сумма значений в указанном диапазоне, удовлетворяющих заданному условию	СУММЕСЛИ (Х1:Х2; критерий)
Часто встречаемое число	МОДА(Х1:Х2)
Корень из числа	КОРЕНЬ(Х1)

## Задание №10

Сноска — это текст, помещаемый в самом низу страницы, отдельно от основного. Регистр у букв — это вид букв с точки зрения размера, то есть заглавная или строчная.

Формы слова — это варианты одного и того же слова, выражающие различные грамматические значения (например, значение числа и падежа у существительного, времени, числа и лица у глагола и т. д.).

Чаще всего вам при открытии "Расширенного поиска" нужно будет выставлять такие параметры, где вместо слова 'сердце' нужно подставить слово из задания.



Область поиска, чаще всего, основной документ (то есть без учёта сносков).

## Задание №11

### Все формулы для 11 задачи

Чтобы закодировать алфавит из  $n$  символов потребуется минимум  $I = \log_2 n$  (округлённое вверх до целого) бит. Это объём одного символа.

Объём  $n$  символов  $S = I \cdot n$ , где  $I$  - объём одного символа,  $n$  - количество символов.

Чтобы узнать, какой объём у слова, надо сначала узнать объём одной буквы (символа), а после умножить на количество символов в слове. При условии, что используется посимвольное кодирование.

Чтобы закодировать число меньше или равное  $n$  необходимо  $I = \log_2 n$  (округлённое вверх до целого) бит. При условии, что число кодируется как число, а не, например, посимвольно, где каждый символ - цифра.