1. Написать функцию Эйлера $\varphi(n)$.

```
def euler_func(n):
    '''
    Функция Эйлера.
    Возвращает количество чисел в ряду 1, 2, 3, ..., n-1, взаимно простых с n
    Если n - простое, то count = n-1
    '''
    count = 1
    for x in range(2, n):
        count += 1 if gcd(x, n) == 1 else 0
    return count
```

2. Утв. Если p – простое число, то $\varphi(p) = p - 1$.
Док-во. Из определения функции Эйлера действительно следует, что если p – простое число, то все числа, меньшие p, взаимно просты с ним, а их ровно p-1 штук.

3. Пусть p и q два различных простых числа. Тогда $\varphi(pq) = (p-1)(q-1)$.

Доказательство. В ряду $1, 2, \ldots, pq - 1$ не взаимнопростыми с pq будут числа

$$p, 2p, 3p, \cdots, (q-1)\,p$$

и

$$q, 2q, 3q, \cdots, (p-1)\,q.$$

Всего таких чисел будет $(q-1)+(p-1)$. Следовательно, количество чисел, взаимнопростых с pq, будет $pq - 1 - (p-1) - (q-1) = pq - q - p + 1 = (p-1)(q-1)$. □

4. Написать функцию *z_nz_group(n)*.

```
def z_nz_group(n):
    '''
    Множество классов вычетов по модулю n
    '''
    res = []
    for x in range(1, n):
        if gcd(x, n) == 1:
            res.append(x)
    return res
```

5. Проверить, что $Z_9^* = \{1, 2, 4, 5, 7, 8\} = < 2 > = < 5 >$, $Z_7^* = \{1, 2, 3, 4, 5, 6\} = < 3 >$, $Z_8^* = \{1,3, 5, 7\}$ – не циклическая.

Напишем функцию для определения циклической группы

```python
def cyclic_group(n):
    z_nz = z_nz_group(n)
    for x in z_nz:
        gr = []
        for i in range(len(z_nz)):
            gr.append(x**i % n)

        gr.sort()

        if gr == z_nz:
            return x

    return None
```

Результат:

```
z_7z: [1, 2, 3, 4, 5, 6] = 3
z_9z: [1, 2, 4, 5, 7, 8] = 2
z_8z: [1, 3, 5, 7] = None
```

6. Посчитать порядок чисел в Z_9^*.

g	1	2	4	5	7	8
$\mathrm{ord}(g)$	1	6	3	6	3	2

7. Написать функцию *multiplicative_order(g,n)*.

```python
def multiplicative_order(g, n):
    '''
    Порядок числа g в группе G по модулю n - это такое число,
    что g**m = 1(mod n), m >= 1
    '''
    m = 1
    while g**m % n != 1:
        m += 1
    return m
```

8. Проверить утверждение, что порядок любого элемента $a \in Z_n^*$. является делителем $\varphi(n)$, для *n* = 10.

```
x = 1, ord(x) = 1
x = 3, ord(x) = 4
x = 7, ord(x) = 4
x = 9, ord(x) = 2
```

Действительно, $\varphi(10) = 4$, а 4 делится на 1, 2 и 4.

9. Написать функцию *primitive_roots(n)*.

```
def primitive_roots(n):
    '''
    Возвращает первообразные корни.
    Если порядок элемента g равен euler_func(n), то g - первообразный корень по модулю n
    '''
    primitive_roots = []
    z_nz = z_nz_group(n)
    for g in z_nz:
        is_primitive_root = multiplicative_order(g, n) == euler_func(n)
        if is_primitive_root:
            primitive_roots.append(g)
    return primitive_roots
```

10. Проверить утверждение, что если p – простое и в Z_p^*.ровно $\varphi(p-1)$ первообразных корней, для *p* = 19.

```
p = 19, phi(p-1) = 6, primitive roots = [2, 3, 10, 13, 14, 15]
```

11. Проверить утверждение, что если *k* делит *p-1*, то существует $\varphi(k)$ элементов порядка *k*, для *p* = 19.

```
p = 19, phi(k=9) = 6,
mult_order = [1, 18, 18, 9, 9, 9, 3, 6, 9, 18, 3, 6, 18, 18, 18, 9, 9, 2]
```

12. Проверить с помощью функции pow() малую теорему Ферма $a^{p-1}\, mod\, p = 1$.

```
>>> 2**12 % 13 == (2**2)**2 * ((2**2)**2)**2 % 13
True
>>> (2**2)**2 * ((2**2)**2)**2 % 13 == (3 * 9) % 13
True
>>> (3 * 9) % 13 == 1
True
```

```
>>> 10**10 % 11 == 10**2 * ((10**2)**2)**2 % 11 == 1
True
```

```
>>> 2**15485862 % 15485863 == 1
True
```

13. Проверить малую теорему Ферма при p < a.

```
>>> 12**10 % 11 == 1
True
```

14. Док-во малой теоремы Ферма.
    Для любого простого числа p и целого числа k, где p и k – взаимно простые, произведения k и чисел 1, 2, 3, …, p-1 при делении по модулю на p в остатке дают те же самые числа 1, 2, 3, …, p-1.
    Тогда для любого числа a, остатки от деления чисел будут:
    a, 2a, 3a, …, (p-1)a, тогда a*2a*3a*…*(p-1)a = 1*2*3*…*(p-1) (mod p), и отсюда следует
    a^{p-1} * (p-1)! = (p-1)! (mod p), сокращая на (p-1)! Получаем требуемое утверждение a^{p-1} = 1 (mod p).

15. Проверить малую теорему Ферма в более общей формулировке: $a^p = a \ (mod\ p)$ с помощью функции pow().

```
>>> 12**13 % 13 == 12
True
```

16. Док-во обобщения малой теоремы Ферма.
17. Найти обратное по умножению значение для *a* = 7814 в Z_{17449}^* расширенным алгоритмом Евклида и с помощью малой теоремы Ферма $a^{-1} = a^{p-2} \ (mod\ p)$.

```
EEA = 1284,
FLT = 1284
```

18. Найти не простое число, с которым выполняется соотношение $a^{p-1}\ mod\ p = 1$.
    В качестве *a* можно взять любое четное число, а в качестве *p = a + 1*, при этом, чтобы выполнялось требование: p – не простое число.
    То есть, получается $a^a = 1 \ (mod\ (a+1))$, где *a+1* – не простое.
    Например:

```
>>> 1024**1024 % 1025 == 1
True
```

    1025 число не простое, оно точно имеет делители 5 и 25.
    И при этом 1024 и 1025 взаимно простые.

19. Проверить теорему Эйлера: $a^{\varphi(b)} = 1 \ (mod\ b)$, *a* и *b* – взаимно простые числа.

```
5**phi(12) mod 12 =  1
```

20. Док-во теоремы Эйлера.

21. Написать функцию *pow_right_left().*

```
def pow_right_left(a, x, p):
    '''
    Возведение в степень справа-налево
    Вход: a, x, p - целые числа
    Выход: y = a**x mod p
    '''
    y = 1
    s = a
    xb = bin(x)[2:]            # xb = (x_t, x_t-1, ..., x_0) binary
    for i in range(len(xb)):
        if int(xb[i]) == 1:
            y = (y * s) % p
        s = (s**2) % p
    return y
```

22. Написать функцию *pow_left_right().*

```
def pow_left_right(a, x, p):
    '''
    Возведение в степень слева-направо
    Вход: a, x, p - целые числа
    Выход: y = a**x mod p
    '''
    y = 1
    xb = bin(x)[2:]
    for i in range(len(xb)-1, -1, -1):
        y = (y**2) % p
        if int(xb[i]) == 1:
            y = (y * a) % p
    return y
```

23. Написать функцию *sieveEratosthen().*

```
def sieveEratosthen(n):
    '''
    Решето Эратосфена
    '''
    numbers = list(range(2, n+1))
    for number in numbers:
        if number > (n+1)**(1/2):
            break
        if number != 0:
            for c in range(2*number, n+1, number):
                numbers[c-2] = 0
    return [x for x in numbers if x != 0]
```

```
n = 120:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
[31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
[73, 79, 83, 89, 97, 101, 103, 107, 109, 113]
```

24. Задачи.

2.6. Используя свойства функции Эйлера, вычислить $\varphi(53)$, $\varphi(21)$, $\varphi(159)$.

```
phi(53) = 52
phi(21) = 12
phi(159) = 104
```

2.7. Используя теорему Ферма, вычислить $3^{13} \bmod 13$, $5^{22} \bmod 11$, $3^{17} \bmod 5$.

```
3**13 mod 13 = 3
5**22 mod 11 = 3
3**17 mod 5 = 3
```

2.8. Используя теорему Эйлера, вычислить $3^{9} \bmod 20$, $2^{14} \bmod 21$, $2^{107} \bmod 159$.

```
3**9 mod 20 = 3
2**14 mod 21 = 4
2**107 mod 159 = 8
```

2.13. Найти все допустимые варианты выбора параметра g в системе Диффи–Хеллмана при $p = 11$.

При p = 11, g может принимать значения 2, 6, 7, 8.

2.14. Вычислить секретные ключи Y_A, Y_B и общий ключ Z_{AB} для системы Диффи–Хеллмана с параметрами:

а. $p = 23$, $g = 5$, $X_A = 5$, $X_B = 7$,

$Y_A = g^{X_A} \, mod \, p = 5^5 \, mod \, 23 = 20$

$Y_B = g^{X_B} \, mod \, p = 5^7 \, mod \, 23 = 17$

$Z_{AB} = Y_B^{X_A} \, mod \, p = 17^5 \, mod \, 23 = 21$

2.15. Для шифра Шамира с заданными параметрами p, c_A, c_B найти недостающие параметры и описать процесс передачи сообщения m от A к B:

a. $p = 19$, $c_A = 5$, $c_B = 7$, $m = 4$,

$gcd(4,19) = 1, gcd(6,19) = 1.$

Посчитаем d_A и d_B так, что $c_A d_A\ mod(p-1) = 1$ и $c_B d_B\ mod(p-1) = 1$:

$d_A = 4$ и $d_B = 11$

А вычисляет число $x_1 = m^{c_A}\ mod\ p = 4^5\ mod\ 19 = 17$ и пересылает В.

В вычисляет число $x_2 = x_1^{c_B}\ mod\ p = 17^7\ mod\ 19 = 5$ и пересылает А.

А вычисляет число $x_3 = x_2^{d_A}\ mod\ p = 5^4\ mod\ 19 = 17$ и передает В.

В вычисляет число $x_4 = x_3^{d_B}\ mod\ p = 17^{11}\ mod\ 19 = 4$ и получает передаваемое сообщение.

2.16. Для шифра Эль-Гамаля с заданными параметрами p, g, c_B, k найти недостающие параметры и описать процесс передачи сообщения m пользователю B:

a. $p = 19$, $g = 2$, $c_B = 5$, $k = 7$, $m = 5$,

В вычисляет $d_B = g^{c_B}\ mod\ p = 2^5\ mod\ 19 = 13$

А формирует пару (r, e) и передает В.

$r = g^k\ mod\ p = 2^7\ mod\ 19 = 14, e = m * d_B^k\ mod\ p = 5 * 13^7\ mod\ 19 = 12$

В вычисляет:

$m' = e * r^{p-1-c_B}\ mod\ p = 14 * 12^{19-1-5}\ mod\ 19 = 5$

1. Сгенерировать нечетное число и проверить его функцией *rabin_miller()*.

```
n = 613
q = 153
k = 2
trials = 0, a = 498, v = 612
trials = 1, a = 310, v = 35
trials = 2, a = 458, v = 1
trials = 3, a = 133, v = 1
trials = 4, a = 541, v = 578
613 is prime with probability = 0.9990234375
```

Для n = 221 и t = 4:

```
n = 221
q = 55
k = 2
trials = 0, a = 21, v = 200
trials = 0, a = 47, v = 174
trials = 0, a = 174, v = 47
trials = 0, a = 200, v = 21
trials = 1, a = 21, v = 200
trials = 1, a = 47, v = 174
trials = 1, a = 174, v = 47
trials = 1, a = 200, v = 21
trials = 2, a = 21, v = 200
trials = 2, a = 47, v = 174
trials = 2, a = 174, v = 47
trials = 2, a = 200, v = 21
trials = 3, a = 21, v = 200
trials = 3, a = 47, v = 174
trials = 3, a = 174, v = 47
trials = 3, a = 200, v = 21
221 is prime with probability = 0.99609375
```

Хотя на самом деле:

```
n = 221
q = 55
k = 2
trials = 0, a = 21, v = 200
trials = 1, a = 85, v = 119
221 is composite
```

Найти 10 простых чисел в диапазоне [13000, 14000] функцией *is_prime().*

```
13001
13003
13007
13009
13033
13037
13043
13049
13063
13093
```

Проверить, что числа 1000000000061 и 1000000000063 простые.

```
1000000000061 (True, 0.9990234375)
1000000000063 (True, 0.9990234375)
```

2. Написать значения $\pi(x)$ и $x/\ln(x)$ для $x = 10^5$.

```
x = 100 pi(x) = 25 x/ln(x) = 22
x = 1000 pi(x) = 168 x/ln(x) = 145
x = 10000 pi(x) = 1229 x/ln(x) = 1086
x = 100000 pi(x) = 9592 x/ln(x) = 8686
```

3. Написать функцию *generate_large_prime(bitfield_width).*

```python
def generate_large_prime(bitfield_width):
    '''
    Возвращает простое число, в двоичной СС которой содержится bitfield_width бит
    '''
    candidate = 0
    while True:
        candidate = getrandbits(bitfield_width)
        candidate += 1 if not candidate & 1 else 0  # искусственно делаем нечетное число
        candidate |= (1 << bitfield_width - 1)  # два старших
        candidate |= (2 << bitfield_width - 3)  # бита равны 1
        if is_prime(candidate):
            break
    return candidate
```

```
1941449784413 = 11100010000000111011011001011110001011101 содержит 41 бит
6920091953489503 = 11000100101011100100101110110101100110110011001011111 содержит 53 бит
1005150799300876617704543597644587401 = 11000001100101011100001010110101111100011101110101010000000
110100111101000111100101101000011000 1001 содержит 120 бит
```

4. Написать функцию *get_blocks_from_data()*.

```python
def get_blocks_from_data(data, block_size):
    block_ints = []
    for blockStart in range(0, len(data), block_size):
        block_int = 0
        for i in range(blockStart, min(blockStart + block_size, len(data))):
            block_int += data[i] * (256 ** (i % block_size))
        block_ints.append(block_int)
    return block_ints
```

```python
print(get_blocks_from_text('Hello world!', 12))
print(get_blocks_from_data([72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 33], 12))
```

Windows Powershell

```
[1033441003260674863331426632]
[1033441003260674863331426632]
```

5. Написать функцию *get_data_from_blocks()*.

```python
def get_data_from_blocks(block_ints, message_length, block_size):
    message = []
    for block_int in block_ints:
        block_message = []
        for i in range(block_size - 1, -1, -1):
            if len(message) + i < message_length:
                ascii_number = block_int // (256 ** i)
                block_int %= 256 ** i
                block_message.insert(0, ascii_number)
        message.extend(block_message)
    return message
```

```python
print(get_text_from_blocks([1033441003260674863331426632], 12, 12))
print(get_data_from_blocks([1033441003260674863331426632], 12, 12))
```

Windows Powershell

```
Hello world!
[72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 33]
```

6. Зашифровать свое ФИО.

```
data = read_write_file.read_data_1byte('name.txt')
ints = get_blocks_from_data(data, 3)
print('ints =', ints)
blocks = get_data_from_blocks(ints, len(data), 3)
read_write_file.write_data_1byte('res_name.txt', blocks)
```

7. Написать функции *elgamal_encrypt(),elgamal_decrypt().*

```
def elgamal_encrypt(pub_key, g, p, m):
    '''
    pub_key = public key
    g = generator
    p = prime
    message = number < p
    '''
    k = randint(1, p-1)
    c1 = pow_(g, k, p)
    c2 = (m * pow_(pub_key, k, p)) % p

    return c1, c2

def elgamal_decrypt(pri_key, p, c1, c2):
    '''
    pri_key = private key
    p = prime
    (c1 ,c2) = ciphertext
    '''
    return (get_inverse_ltf(pow_(c1, pri_key, p), p) * c2) % p
```

Известно: $p = 2^{31} - 1$, g=7, pub_key= 833287206, (c1,c2) = (1457850878, 2110264777).

Найти: m

m = 1535607309.

Зашифровать свое ФИО:

```
name = 'Fedotov Alexander Alexandrovich'
m = get_blocks_from_text(name, len(name))[0]
keys = generate_keys()
p, g, y = keys['public_key']
x       = keys['private_key']

r, e = elgamal_encrypt(y, g, p, m)
print('Message:', m)
print('Public key:', p, g, y)
print('Private key:', x)
print('Cipher:', r, e)

print(elgamal_decrypt(x, p, r, e))
```

Message:

m = 18443821089238127661795593422828031285394092330040283941651807546664998637 4

Public key:

p = 87961037278023961165805299068784239872629837284207848824753538184398307011989

g = 83923678229832266673697378811758277222770016849898225104778457608038173754 13

y = 20810124915121381041265143820679706485337843940842399309475316095341608187419

Private key:

x = 82195069053570735358608701805178729379421525961055031649599075571308769026097

Cipher:

r = 58347684978701003542952516189653078767366202354826008932793129126383322789203

e = 73525424255885287409318134712053447786948476665405513015910574754472147516 40

Decrypt message:

m' = 18443821089238127661795593422828031285394092330040283941651807546664998637 4