1. 🔢 **Importando bibliotecas**

```
import os

from math import gcd
```

- import os: importa uma biblioteca que permite interagir com o sistema operacional, como arquivos e pastas.
- from math import gcd: importa da biblioteca math a função gcd, que calcula o **máximo divisor comum** entre dois números.

---

2. 🔐 **Codificação e decodificação de mensagens**

Essas funções transformam texto em números e vice-versa.

```
def encode_message(message):
    message = message.upper()
    alphabet = {chr(i + 65): i + 2 for i in range(26)}
    alphabet[" "] = 28
    return [alphabet[char] for char in message if char in alphabet]
```

- Converte a mensagem para **letras maiúsculas** com upper().
- Cria um **dicionário** chamado alphabet onde:
    - 'A' vira 2, 'B' vira 3, até 'Z' virar 27.
    - O espaço " " vira 28.
- Transforma a mensagem em **números** usando esse dicionário.

```
def decode_message(codes):
    reverse_alphabet = {i + 2: chr(i + 65) for i in range(26)}
    reverse_alphabet[28] = " "
    return "".join(reverse_alphabet.get(code, "?") for code in codes)
```

- Faz o **inverso** da função acima: transforma os **números** de volta em **letras**.
- Se o código não estiver no dicionário, coloca “?”.

---

3. **□ Funções matemáticas usadas no RSA**

Essas funções ajudam nos cálculos da criptografia RSA.

```
def mod_exp(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp >> 1
        base = (base * base) % mod
    return result
```

- Faz **exponenciação modular rápida**, ou seja: calcula (base^exp) % mod de forma eficiente.
- Isso é essencial no RSA para encriptar e desencriptar com números grandes.

```
def mod_inverse(e, phi):
    for d in range(2, phi):
        if (e * d) % phi == 1:
            return d
    raise Exception("Não foi possível encontrar inverso modular.")
```

- Procura um número d que satisfaça: (e * d) % phi == 1.
- Esse d é a **chave privada** na criptografia RSA.
- Se não encontrar, dá erro.

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

- Verifica se um número n é **primo** (usado para gerar chaves).
- Um número é primo se **só é divisível por 1 e por ele mesmo**.

---

4. 🔑 Gerar chaves RSA

```
def gerar_chave_publica():
```

Entrada dos primos p e q

```
    p = int(input("Digite um número primo p: "))
    q = int(input("Digite um número primo q: "))
```

- O usuário digita dois números primos.

Verificação se são primos

```
    if not is_prime(p) or not is_prime(q):
        print("Erro: p e q devem ser números primos.")
        return
```

- Se p ou q **não forem primos**, o programa para.

Cálculo de n e phi

```
    n = p * q
    phi = (p - 1) * (q - 1)
```

- n: parte da chave pública.
- phi: usado para calcular a chave privada.

Escolha do e

```
    e = int(input(f"Digite o expoente e (relativamente primo a {phi}): "))
    if gcd(e, phi) != 1:
        print("Erro: e não é relativamente primo a (p - 1)(q - 1).")
        return
```

- e é outro número que o usuário digita.
- Ele precisa ser **relativamente primo** a phi (o gcd deve ser 1).

Checagem do tamanho de n

```
if n <= 28:
    print(f"Erro: n = {n} é muito pequeno. Deve ser maior que 28.")
    return
```

- n precisa ser **maior que o valor máximo da tabela de codificação** (28).

Salvando as chaves

```
with open("chave_publica.txt", "w") as f:
    f.write(f"{e},{n}")
```

- Salva a **chave pública** num arquivo.

```
d = mod_inverse(e, phi)
with open("chave_privada.txt", "w") as f:
    f.write(f"{d},{n}")
```

- Calcula o d (chave privada) e salva num arquivo.

---

5. 🔒 Encriptar a mensagem

```
def encriptar():
```

Entrada da mensagem e da chave

```
    mensagem = input("Digite a mensagem (A-Z e espaços): ")
    try:
        e, n = map(int, input("Digite a chave pública (formato: e,n): ").split(","))
```

- O usuário digita a mensagem e a chave pública (valores e e n).

Codifica a mensagem em números

```
    codigos = encode_message(mensagem)
    if any(c >= n for c in codigos):
        print(f"Erro: Algum caractere da mensagem tem valor >= n ({n}). Use um n maior.")
        return
```

- Verifica se algum código da mensagem é maior que n, o que não pode acontecer.

Encriptação usando RSA

```
criptografada = [mod_exp(c, e, n) for c in codigos]
```

- Criptografa cada número usando: (c^e) % n.

Salva mensagem encriptada

```
with open("mensagem_encriptada.txt", "w") as f:
    f.write(",".join(map(str, criptografada)))
```

---

6. 🔓 Desencriptar a mensagem

```
def desencriptar():
```

Entrada da chave privada

```
d, n = map(int, input("Digite a chave privada (formato: d,n): ").split(","))
```

Lê a mensagem encriptada

```
with open("mensagem_encriptada.txt", "r") as f:
    criptografada = list(map(int, f.read().strip().split(",")))
```

Decriptação

```
decodificada = [mod_exp(c, d, n) for c in criptografada]
mensagem = decode_message(decodificada)
```

- Aplica (c^d) % n para cada número e depois transforma de volta para texto.

Salva mensagem original

```
with open("mensagem_desencriptada.txt", "w") as f:
    f.write(mensagem)
```

---

7. □ Menu principal

```
def menu():
    while True:
        print("\nEscolha uma opção:")
        print("1 - Gerar chave pública e privada")
        print("2 - Encriptar")
        print("3 - Desencriptar")
        print("4 - Sair")
```

- Mostra as opções para o usuário.

```
opcao = input("Opção: ")
```

- Lê a opção escolhida e executa a função correspondente:

```
if opcao == "1":
    gerar_chave_publica()
elif opcao == "2":
    encriptar()
elif opcao == "3":
    desencriptar()
elif opcao == "4":
    print("Saindo...")
    break
```

---

8. 🚀 Começo do programa

```
if __name__ == "__main__":
    menu()
```

- Diz ao Python: “Se esse arquivo for executado, comece chamando a função menu()”.