Sure! Let's go through this RSA encryption and decryption Python code **line by line**:

---

1. Install the library (done in terminal/command prompt):

```
pip install pycryptodome
```

This command installs the `pycryptodome` library, which provides cryptographic functions, including the `inverse` function from `Crypto.Util.number`.

---

2. Import necessary libraries

```
from sympy import randprime, gcd
```

- `randprime`: generates a random prime number in a given range.
- `gcd`: calculates the greatest common divisor of two numbers.

```
from Crypto.Util.number import inverse
```

- `inverse`: computes the modular inverse. Essential in RSA for generating the private key.

```
import random
```

- Standard Python library to generate random numbers.

---

3. Function to generate RSA keys

```
def generate_keys(bit_length=512):
```

- This function generates public and private RSA keys.
- `bit_length=512` means each prime number will be 512 bits long (1024-bit RSA key).

```
    p = randprime(2**(bit_length-1), 2**bit_length)
    q = randprime(2**(bit_length-1), 2**bit_length)
```

- Generates two **random prime numbers**, `p` and `q`, each roughly 512 bits long.

```
    while p == q:
        q = randprime(2**(bit_length-1), 2**bit_length)
```

- Ensures `p` and `q` are **not equal** (RSA requires two distinct primes).

```
n = p * q
```

- `n` is part of the **public and private keys**.
- It is the **modulus** used in encryption and decryption.

```
phi_n = (p - 1) * (q - 1)
```

- Euler's totient function φ(n), needed to calculate the **modular inverse**.
- Represents the number of integers less than `n` that are coprime to `n`.

```
e = random.randrange(2, phi_n)
while gcd(e, phi_n) != 1:
    e = random.randrange(2, phi_n)
```

- Chooses `e` such that `1 < e < φ(n)` and `e` is **coprime** to `φ(n)` (i.e., GCD = 1).
- This will be the **public exponent** used to encrypt messages.

```
d = inverse(e, phi_n)
```

- `d` is the **private exponent**, calculated as the **modular inverse** of `e` mod `φ(n)`.
- Used to **decrypt** messages.

```
return ((e, n), (d, n))
```

- Returns the **public key** `(e, n)` and **private key** `(d, n)` as tuples.

---

4. Encryption function

```
def encrypt(plaintext, public_key):
    e, n = public_key
```

- Takes the plaintext message and the **public key**.

```
plaintext_int = int.from_bytes(plaintext.encode(), byteorder='big')
```

- Converts the string into an integer using **UTF-8 encoding**.

```
ciphertext = pow(plaintext_int, e, n)
```

- Encrypts the integer using **modular exponentiation**: `ciphertext = plaintext_int^e mod n`.

```
    return ciphertext
```

- Returns the encrypted number.

---

5. Decryption function

```
def decrypt(ciphertext, private_key):
    d, n = private_key
```

- Takes the encrypted message and the **private key**.

```
    plaintext_int = pow(ciphertext, d, n)
```

- Decrypts the ciphertext using **modular exponentiation**: `plaintext = ciphertext^d mod n`.

```
    plaintext = plaintext_int.to_bytes((plaintext_int.bit_length() + 7) // 8,
byteorder='big').decode()
```

- Converts the integer back to bytes, then decodes it to get the original string.

```
    return plaintext
```

- Returns the original message.

---

6. Main function

```
def main():
```

- The main function that demonstrates key generation, encryption, and decryption.

```
    public_key, private_key = generate_keys()
```

- Generates the RSA key pair.

```
    print("Public Key (e, n):", public_key)
    print("Private Key (d, n):", private_key)
```

- Displays the keys.

```
    message = "Hello RSA!"
```

```
    print("\nOriginal Message:", message)
```

- The plaintext message to encrypt.

```
    ciphertext = encrypt(message, public_key)
    print("\nCiphertext:", ciphertext)
```

- Encrypts the message and prints the result.

```
    decrypted_message = decrypt(ciphertext, private_key)
    print("\nDecrypted Message:", decrypted_message)
```

- Decrypts the ciphertext and shows the original message.

---

7. Run the main function

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

- This ensures that `main()` runs only when the script is executed directly (not when imported).

Would you like a diagram or flowchart to help visualize the process?