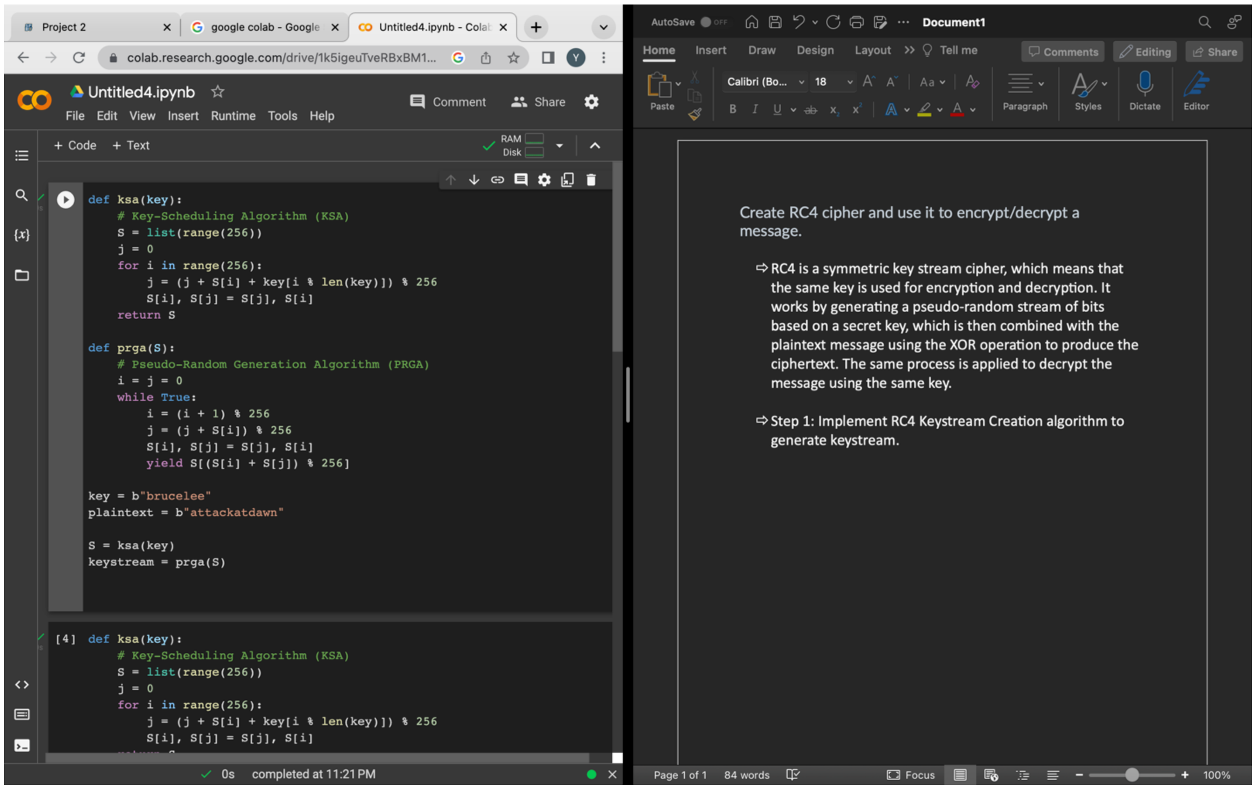
Create RC4 cipher and use it to encrypt/decrypt a message.

⇨ RC4 is a symmetric key stream cipher, which means that the same key is used for encryption and decryption. It works by generating a pseudo-random stream of bits based on a secret key, which is then combined with the plaintext message using the XOR operation to produce the ciphertext. The same process is applied to decrypt the message using the same key.

⇨ Step 1: Implement RC4 Keystream Creation algorithm to generate keystream.



```
def ksa(key):
    # Key-Scheduling Algorithm (KSA)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    return S

def prga(S):
    # Pseudo-Random Generation Algorithm (PRGA)
    i = j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        yield S[(S[i] + S[j]) % 256]

key = b"brucelee"
plaintext = b"attackatdawn"

S = ksa(key)
keystream = prga(S)
```

```
[4] def ksa(key):
        # Key-Scheduling Algorithm (KSA)
        S = list(range(256))
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) % 256
            S[i], S[j] = S[j], S[i]
```

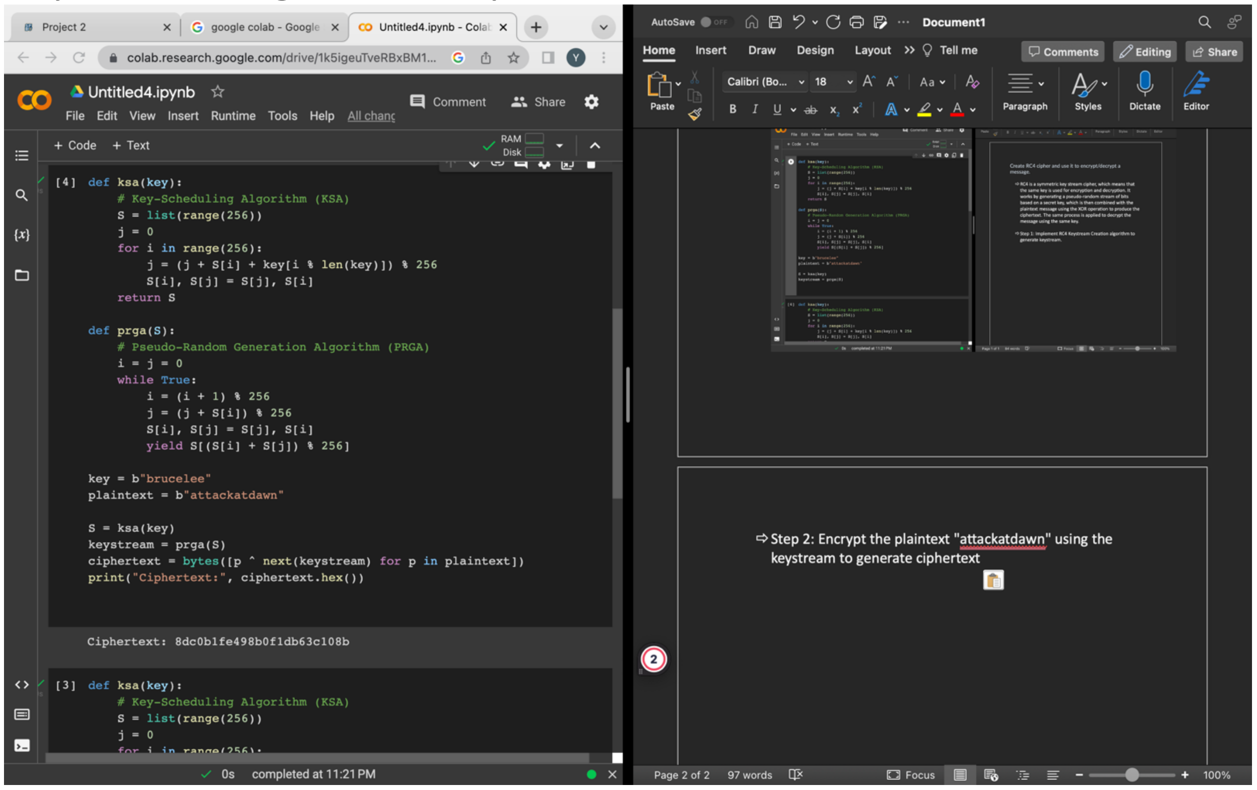
```
0s   completed at 11:21 PM
```

Create RC4 cipher and use it to encrypt/decrypt a message.

⇨ RC4 is a symmetric key stream cipher, which means that the same key is used for encryption and decryption. It works by generating a pseudo-random stream of bits based on a secret key, which is then combined with the plaintext message using the XOR operation to produce the ciphertext. The same process is applied to decrypt the message using the same key.

⇨ Step 1: Implement RC4 Keystream Creation algorithm to generate keystream.

⇨ Step 2: Encrypt the plaintext "attackatdawn" using the keystream to generate ciphertext



```
[4] def ksa(key):
        # Key-Scheduling Algorithm (KSA)
        S = list(range(256))
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) % 256
            S[i], S[j] = S[j], S[i]
        return S

    def prga(S):
        # Pseudo-Random Generation Algorithm (PRGA)
        i = j = 0
        while True:
            i = (i + 1) % 256
            j = (j + S[i]) % 256
            S[i], S[j] = S[j], S[i]
            yield S[(S[i] + S[j]) % 256]

    key = b"brucelee"
    plaintext = b"attackatdawn"

    S = ksa(key)
    keystream = prga(S)
    ciphertext = bytes([p ^ next(keystream) for p in plaintext])
    print("Ciphertext:", ciphertext.hex())
```

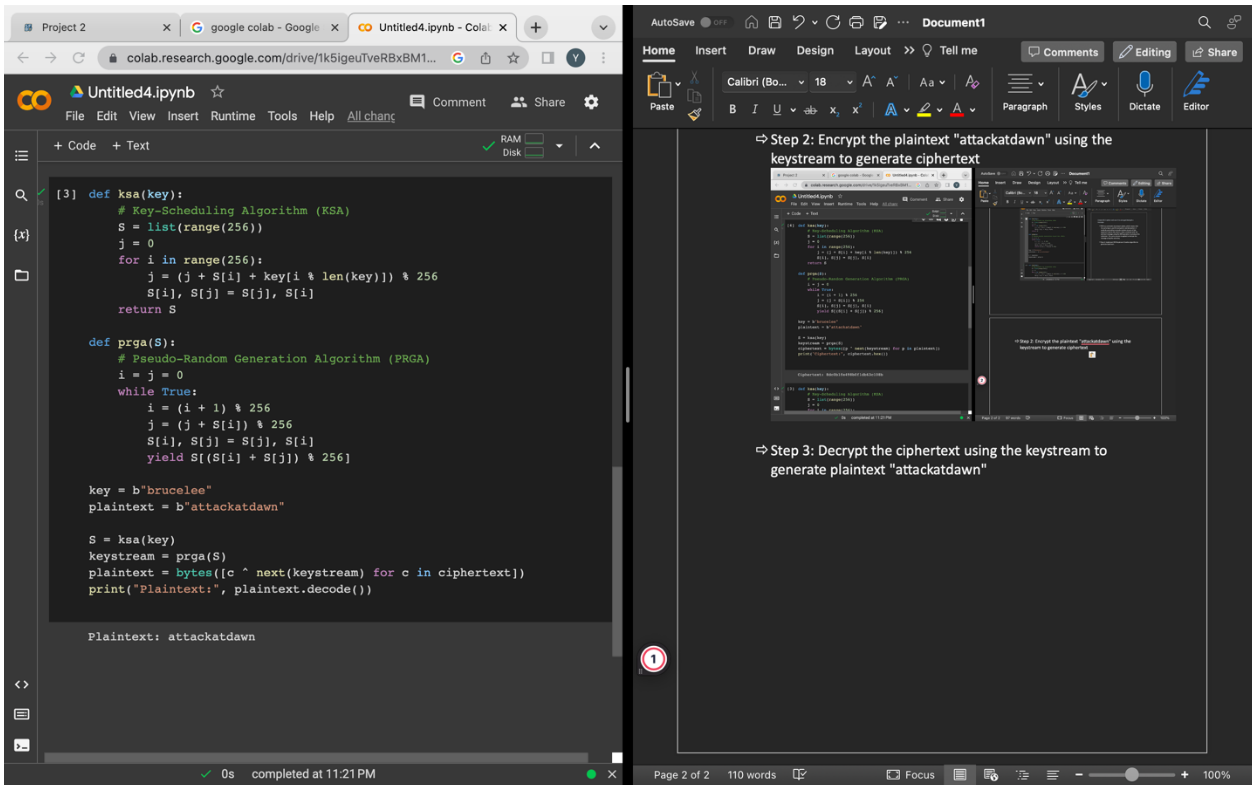
```
Ciphertext: 8dc0b1fe498b0f1db63c108b
```

```
[3] def ksa(key):
        # Key-Scheduling Algorithm (KSA)
        S = list(range(256))
        j = 0
```

0s completed at 11:21 PM

⇨ Step 2: Encrypt the plaintext "attackatdawn" using the keystream to generate ciphertext

⇨ Step 3: Decrypt the ciphertext using the keystream to generate plaintext "attackatdawn"



```
[3] def ksa(key):
        # Key-Scheduling Algorithm (KSA)
        S = list(range(256))
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) % 256
            S[i], S[j] = S[j], S[i]
        return S

    def prga(S):
        # Pseudo-Random Generation Algorithm (PRGA)
        i = j = 0
        while True:
            i = (i + 1) % 256
            j = (j + S[i]) % 256
            S[i], S[j] = S[j], S[i]
            yield S[(S[i] + S[j]) % 256]

    key = b"brucelee"
    plaintext = b"attackatdawn"

    S = ksa(key)
    keystream = prga(S)
    plaintext = bytes([c ^ next(keystream) for c in ciphertext])
    print("Plaintext:", plaintext.decode())
```

```
Plaintext: attackatdawn
```

0s completed at 11:21 PM
AutoSave OFF Document1
Home Insert Draw Design Layout Tell me
Comments Editing Share
Paste Calibri (Bo... 18 Paragraph Styles Dictate Editor
Page 2 of 2 110 words Focus 100%

⇨ Step 2: Encrypt the plaintext "attackatdawn" using the keystream to generate ciphertext

⇨ Step 3: Decrypt the ciphertext using the keystream to generate plaintext "attackatdawn"