```python
def caesar_cipher(text, shift):
    out = ""
    for c in text:
        if c.isalpha():
            base = ord('A') if c.isupper() else ord('a')
            out += chr((ord(c) - base + shift) % 26 + base)
        else:
            out += c
    return out

def atbash_cipher(text):
    out = ""
    for c in text:
        if c.isalpha():
            base = ord('A') if c.isupper() else ord('a')
            out += chr(25 - (ord(c) - base) + base)
        else:
            out += c
    return out

def rot13_cipher(text):
    return caesar_cipher(text, 13)

msg = "Hello World"

print("Caesar (+3):", caesar_cipher(msg, 3))
print("Atbash:", atbash_cipher(msg))
print("ROT13:", rot13_cipher(msg))
```

```
Caesar (+3): Khoor Zruog
Atbash: Svool Dliow
ROT13: Uryyb Jbeyq
```

```python
def rail_fence_cipher(text, rails):

    fence = [''] * rails
    rail, step = 0, 1
    for c in text:
        fence[rail] += c
        rail += step
        if rail == 0 or rail == rails - 1:
            step *= -1
    return ''.join(fence)

# Example usage
msg = "HELLO WORLD"
print("3 rails:", rail_fence_cipher(msg, 3))
print("4 rails:", rail_fence_cipher(msg, 4))
```

```
3 rails: HOREL OLLWD
4 rails: HOLELWRD LO
```

Diffie Hellman

```
p = int(input("Prime p: "))
g = int(input("Primitive root g: "))
a = int(input("Alice key: "))
b = int(input("Bob key: "))

A = g**a % p
B = g**b % p
print("Alice secret:", B**a % p)
print("Bob secret:  ", A**b % p)
key = pow(g, a * b, p)
print("Out:", key)
```

```
Prime p: 23
Primitive root g: 5
Alice key: 6
Bob key: 15
Alice secret: 2
Bob secret:   2
```

RSA

```
p = int(input("Prime p: "))
q = int(input("Prime q: "))
m = int(input("Message: "))

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

e = 13
while phi % e == 0:
    e += 2

d = 1
while (e * d) % phi != 1:
    d += 1

c = pow(m, e, n)
print("Out:", c)

decry = pow(c, d, n)
print("decrypter text ",decry)
```

```
Prime p: 13
Prime q: 11
Message: 13
Out: 52
decrypter text  13
```

SHA 256 and MD5

```python
import hashlib
text = input("Enter text: ")

md5_hash = hashlib.md5(text.encode()).hexdigest()
print("MD5:", md5_hash)

sha_hash = hashlib.sha256(text.encode()).hexdigest()
print("SHA-256:", sha_hash)
```

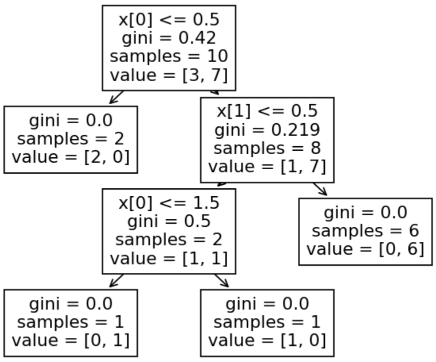
```
Enter text: abcd
MD5: e2fc714c4727ee9395f324cd2e7f331f
SHA-256: 88d4266fd4e6338d13b845fcf289579d209c897823b9217da3e161936f031589
```

```python
from Crypto.Cipher import Blowfish
from Crypto.Util.Padding import pad, unpad

key = b'mysecretkey'  # Key must be bytes
data = b'HelloWorld'  # Data must be bytes

cipher = Blowfish.new(key, Blowfish.MODE_ECB)

ciphertext = cipher.encrypt(pad(data, 8))
print("Encrypted:", ciphertext.hex())

decrypted = unpad(cipher.decrypt(ciphertext), 8)
print("Decrypted:", decrypted.decode())
```

```
Encrypted: 48346cf3a2d6422793c5d690bb458a38
Decrypted: HelloWorld
```

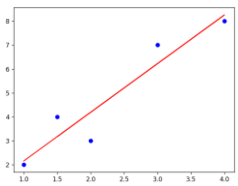
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

X = [
    [0, 0], [1, 1], [1, 0], [0, 1],
    [2, 1], [2, 0], [3, 3], [3, 2],
    [4, 4], [4, 3]
]
y = [0, 1, 1, 0, 1, 0, 1, 1, 1, 1]

clf = DecisionTreeClassifier()

clf.fit(X, y)

predictions = clf.predict([[1, 0], [3, 3], [0, 0]])
print("Predictions:", predictions)

tree.plot_tree(clf)
plt.show()
```

```
Predictions: [1 1 0]
```



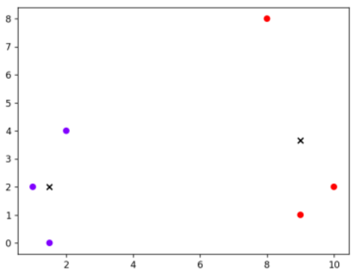
```python
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[1],[2],[3],[4],[1.5]])
y = np.array([2,3,7,8,4])
model = LinearRegression().fit(X, y)

print("Prediction for 5:", model.predict([[5]])[0])

plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.show()
```

```
Prediction for 5: 10.293103448275861
```



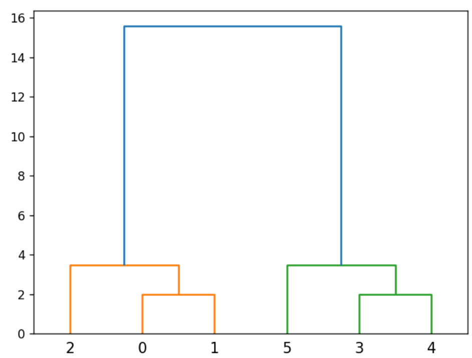
```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[1,2],[2,4],[1.5,0],
              [10,2],[8,8],[9,1]])

kmeans = KMeans(n_clusters=2, n_init=10).fit(X)

print("Cluster Labels:", kmeans.labels_)
print("Cluster Centers:\n", kmeans.cluster_centers_)

plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            c='black', marker='x')
plt.show()
```

```
Cluster Labels: [0 0 0 1 1 1]
Cluster Centers:
 [[1.5        2.        ]
 [9.         3.66666667]]
```




```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

X = np.array([[1,2],[1,4],[1,0],
              [10,2],[10,4],[10,0]])

linked = linkage(X, method='ward')

print("Linkage Matrix:\n", linked)

dendrogram(linked)
plt.show()
```

```
Linkage Matrix:
 [[ 0.          1.          2.          2.        ]
 [ 3.          4.          2.          2.        ]
 [ 2.          6.          3.46410162  3.        ]
 [ 5.          7.          3.46410162  3.        ]
 [ 8.          9.         15.58845727  6.        ]]
```



```python
from itertools import combinations

T = [
    ['A','B','C','D'],
    ['A','C','D'],
    ['B','C','E'],
    ['A','B','C','E'],
    ['B','C','D'],
    ['A','B','C'],
    ['A','C','E'],
    ['B','C','D','E']
]

min_sup = 0.5

def support(itemset):
    count = 0
    for transaction in T:
        if set(itemset).issubset(transaction):
            count += 1
    return count / len(T)

items = []
for transaction in T:
    for item in transaction:
        if item not in items:
            items.append(item)
items.sort()

freq = []

for c in combinations(items, 1):
    sup = support(list(c))
    if sup >= min_sup:
        freq.append((list(c), sup))

for c in combinations(items, 2):
    sup = support(list(c))
    if sup >= min_sup:
        freq.append((list(c), sup))

for c in combinations(items, 3):
    sup = support(list(c))
    if sup >= min_sup:
        freq.append((list(c), sup))

print("Frequent Itemsets with Support:")
for itemset, sup in freq:
    print(itemset, "->",sup)
```

```
Frequent Itemsets with Support:
['A'] -> 0.625
['B'] -> 0.75
['C'] -> 1.0
['D'] -> 0.5
['E'] -> 0.5
['A', 'C'] -> 0.625
['B', 'C'] -> 0.75
['C', 'D'] -> 0.5
['C', 'E'] -> 0.5
```