

Practical 3\practical.py

```

1 #####
2                                     #Practical 3A
3 #####
4 def EEA(a: int ,b: int) → tuple:
5     r1: int = a
6     r2: int = b
7     t1: int = 0
8     t2: int = 1
9     s1: int = 1
10    s2: int = 0
11    print('-'*70)
12    print(f'|{'q':^6}|',end='')
13    print(f'|{'r1':^6}|',end='')
14    print(f'|{'r2':^6}|',end='')
15    print(f'|{'r':^6}|',end='')
16    print(f'|{'t1':^6}|',end='')
17    print(f'|{'t2':^6}|',end='')
18    print(f'|{'t':^6}|',end='')
19    print(f'|{'s1':^6}|',end='')
20    print(f'|{'s2':^6}|',end='')
21    print(f'|{'s':^6}|')
22    print('-'*70)
23
24    while(r2 > 0):
25        q: int = r1 // r2
26        print(f'|{'q':^6}|',end='')
27        print(f'|{'r1':^6}|',end='')
28        print(f'|{'r2':^6}|',end='')
29        r: int = r1 - q * r2
30        r1, r2 = r2, r
31        print(f'|{'r':^6}|',end='')
32
33        print(f'|{'t1':^6}|',end='')
34        print(f'|{'t2':^6}|',end='')
35        t: int = t1 - q * t2
36        t1, t2 = t2, t
37        print(f'|{'t':^6}|',end='')
38
39        print(f'|{'s1':^6}|',end='')
40        print(f'|{'s2':^6}|',end='')
41        s: int = s1 - q * s2
42        s1, s2 = s2, s
43        print(f'|{'s':^6}|')
44    print(f'|{'':^6}|',end='')
45    print(f'|{'r1':^6}|',end='')
46    print(f'|{'r2':^6}|',end='')
47    print(f'|{'':^6}|',end='')
48    print(f'|{'t1':^6}|',end='')
49    print(f'|{'t2':^6}|',end='')
50    print(f'|{'':^6}|',end='')
51    print(f'|{'s1':^6}|',end='')
52    print(f'|{'s2':^6}|',end='')
53    print(f'|{'':^6}|')
54
55    print('-'*70)
56
57    return (r1, t1, s1)

```

```

58
59 def main() → None:
60     print(f'{'start':-^40}')
61     a: int = int(input("Enter A: "))
62     b: int = int(input("Enter B: "))
63
64     result = EEA(a=a,b=b)
65     print(f"GCD({a},{b}) = {result[0]}")
66     print("coefficients of Bezout's")
67     print("t:",result[1])
68     print("s:",result[2])
69     print(f'{'end':-^40}')
70
71
72
73 if __name__ == '__main__':
74     main()
75 #####
76                                     #Practical 3B
77 #####
78 def EEA(a: int, b: int) → int:
79     r1: int = a
80     r2: int = b
81     t1: int = 0
82     t2: int = 1
83     while r2 > 0:
84         q: int = r1 // r2
85         r: int = r1 - q * r2
86         r1, r2 = r2, r
87         t: int = t1 - q * t2
88         t1, t2 = t2, t
89     return t1
90
91
92 def multiplicativeCipher(inputText: str, key: int):
93     alphabet: str = "abcdefghijklmnopqrstuvwxyz"
94     result: str = ""
95     if inputText.islower():
96         for ch in inputText:
97             result += alphabet[(alphabet.find(ch) * key) % 26].upper()
98         print("Encrypted:", result)
99     else:
100         t = EEA(26, key)
101         while t < 0:
102             t += 26
103         for ch in inputText.lower():
104             result += alphabet[(alphabet.find(ch) * t) % 26].lower()
105         print("Decrypted:", result)
106
107 def AffineCipher(inputText: str, k1: int, k2: int):
108     alphabet: str = "abcdefghijklmnopqrstuvwxyz"
109     result: str = ""
110     if inputText.islower():
111         for ch in inputText:
112             result += alphabet[((alphabet.find(ch) * k1) + k2) % 26]
113         print("Encrypted:", result.upper())
114     else:
115         inputText = inputText.lower()
116         k1 = EEA(26, k1)

```

```

117         k2 = 26 - k2
118         while k2 < 0:
119             k2 += 26
120         while k1 < 0:
121             k1 += 26
122         print(f'{k1=}')
123         print(f'{k2=}')
124         for ch in inputText:
125             result += alphabet[((alphabet.find(ch) + k2) * k1) % 26]
126         print("Decrypted:", result.lower())
127
128
129 def main() → None:
130     print(f'{'start':-^40}')
```

131

```

132     choice: str = input('(M) for Multiplicative Cipher\n(A) for Affine Cipher\nEnter Your
Choice: ').lower()
133     match choice:
134         case 'm':
135             print('-'*40)
136             inputText: str = input("Enter Your Text: ")
137             key: int = int(input("Enter Your Key: "))
138             print('-'*40)
139             multiplicativeCipher(inputText, key)
140         case 'a':
141             print('-'*40)
142             inputText: str = input("Enter Your Text: ")
143             k1: int = int(input("Enter Key1: "))
144             k2: int = int(input("Enter Key2: "))
145             print('-'*40)
146             AffineCipher(inputText, k1, k2)
147
148     print(f'{'end':-^40}')
```

149

```

150
151 if __name__ == "__main__":
152     main()
153 #####
154             #Practical 3C
155 #####
156 import random
157
158 def EEA(a: int, b: int) → int:
159     r1: int = a
160     r2: int = b
161     t1: int = 0
162     t2: int = 1
163     while r2 > 0:
164         q: int = r1 // r2
165         r: int = r1 - q * r2
166         r1, r2 = r2, r
167         t: int = t1 - q * t2
168         t1, t2 = t2, t
169     return t1
170
171 def hillCipher(text: str, matrix: List[int]):
172     size: int = 2
173     alphabet: str = 'abcdefghijklmnopqrstuvwxyz'
174     flag: bool = False if text.islower() else True
```

```

175
176     for i, v in enumerate(matrix):
177         while v < 0:
178             v += 26
179             matrix[i] = v
180
181     if flag:
182         text = text.lower()
183         a, b, c, d = matrix
184         det = ((a*d) - (b*c))
185         while( det < 0):
186             det += 26
187         detinv = EEA(26, det)
188         while(detinv < 0):
189             detinv += 26
190         adjoint = [d, -b, -c, a]
191         for i, v in enumerate(adjoint):
192             while v < 0:
193                 v += 26
194             adjoint[i] = v
195         for i, v in enumerate(adjoint):
196             matrix[i] = v * detinv
197
198
199     while(len(text) % size):
200         text += random.choice("".join(ch for ch in alphabet if ch not in [*text]))
201
202     textlist: list[str] = [text[i:i+size] for i in range(0,len(text),size)]
203
204     result: str = ""
205     for txt in textlist:
206         plist: list[int] = list()
207         for ch in txt:
208             plist.append(alphabet.find(ch))
209         x, y = plist
210         a, b, c, d = matrix
211         clist: list[int] = [(a*x + c*y)%26, (b*x + d*y)%26]
212         for v in clist:
213             result += alphabet[v]
214
215     if flag:
216         print("Decrypted:", result.lower())
217     else:
218         print("Encrypted:", result.upper())
219
220
221 def main() → None:
222     print(f'{'start':-^40}')
223     matrix: list[int] = list(map(int, input("Enter Your Matrix a, b, c, d: ").strip().split("
224     ))))
225     inputText: str = input("Enter Your String: ")
226     print('-'*40)
227     hillCipher(inputText, matrix)
228     print(f'{'end':-^40}')
229
230 if __name__ == '__main__':
    main()

```