

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 if __name__ == '__main__':
72     main()
73 #####
74             #Practical 3B
75 #####
76 def EEA(a: int, b: int) → int:
77     r1: int = a
78     r2: int = b
79     t1: int = 0
80     t2: int = 1
81     while r2 > 0:
82         q: int = r1 // r2
83         r: int = r1 - q * r2
84         r1, r2 = r2, r
85         t: int = t1 - q * t2
86         t1, t2 = t2, t
87     return t1
88
89
90 def multiplicativeCipher(inputText: str, key: int):
91     alphabet: str = "abcdefghijklmnopqrstuvwxyz"
92     result: str = ""
93     if inputText.islower():
94         for ch in inputText:
95             result += alphabet[(alphabet.find(ch) * key) % 26].upper()
96         print("Encrypted:", result)
97     else:
98         t = EEA(26, key)
99         while t < 0:
100             t += 26
101         for ch in inputText.lower():
102             result += alphabet[(alphabet.find(ch) * t) % 26].lower()
103         print("Decrypted:", result)
104
105 def AffineCipher(inputText: str, k1: int, k2: int):
106     alphabet: str = "abcdefghijklmnopqrstuvwxyz"

```

```

107     result: str = ""
108     if inputText.islower():
109         for ch in inputText:
110             result += alphabet[((alphabet.find(ch) * k1) + k2) % 26].upper()
111         print("Encrypted:", result)
112     else:
113         t = EEA(26, k1)
114         while t < 0:
115             t += 26
116         for ch in inputText.lower():
117             index: int = alphabet.find(ch) - k2
118             while( index < 0):
119                 index += 26
120             result += alphabet[(index * t) % 26].lower()
121         print("Decrypted:", result)
122
123
124 def main() → None:
125     print(f'{'start':-^40}')
126
127     choice: str = input(''
128     Enter
129     (M) for Multiplicative Cipher
130     (A) for Affine Cipher
131     Enter Your Choice: '').lower()
132     match choice:
133         case 'm':
134             print('-'*40)
135             inputText: str = input("Enter Your Text: ")
136             key: int = int(input("Enter Your Key: "))
137             print('-'*40)
138             multiplicativeCipher(inputText, key)
139         case 'a':
140             print('-'*40)
141             inputText: str = input("Enter Your Text: ")
142             k1: int = int(input("Enter Key1: "))
143             k2: int = int(input("Enter Key2: "))
144             print('-'*40)
145             AffineCipher(inputText, k1, k2)
146
147     print(f'{'end':-^40}')
148
149
150 if __name__ == "__main__":
151     main()
152     #####
153     #Practical 3C
154     #####
155 import random
156
157 def EEA(a: int, b: int) → int:
158     r1: int = a
159     r2: int = b
160     t1: int = 0

```

```

161     t2: int = 1
162     while r2 > 0:
163         q: int = r1 // r2
164         r: int = r1 - q * r2
165         r1, r2 = r2, r
166         t: int = t1 - q * t2
167         t1, t2 = t2, t
168     return t1
169
170 def hillCipher(text: str, matrix: list[int]):
171     size: int = 2
172     alphabet: str = 'abcdefghijklmnopqrstuvwxyz'
173     flag: bool = False if text.islower() else True
174
175     for i, v in enumerate(matrix):
176         while v < 0:
177             v += 26
178         matrix[i] = v
179
180     if flag:
181         text = text.lower()
182         a, b, c, d = matrix
183         det = ((a*d) - (b*c))
184         while( det < 0):
185             det += 26
186         detinv = EEA(26, det)
187         while(detinv < 0):
188             detinv += 26
189         adjoint = [d, -b, -c, a]
190         for i, v in enumerate(adjoint):
191             while v < 0:
192                 v += 26
193             adjoint[i] = v
194         for i, v in enumerate(adjoint):
195             matrix[i] = v * detinv
196
197
198     while(len(text) % size):
199         text += random.choice("".join(ch for ch in alphabet if ch not in
200 [*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     if flag:
215         print("Decrypted:", result.lower())
216     else:
217         print("Encrypted:", result.upper())
218
219
220 def main() → None:
221     print(f'{'start':-^40}')
222     matrix: list[int] = list(map(int, input("Enter Your Matrix a, b, c, d:
223 ").strip().split(" ")))
224     inputText: str = input("Enter Your String: ")
225     print('-'*40)
226     hillCipher(inputText, matrix)
227     print(f'{'end':-^40}')
228
229 if __name__ == '__main__':
230     main()
```