

# Programming in the Past

---

Robert Perrone

robert.perrone1@marist.edu

March 27, 2021

## 1 FORTRAN

### 1.1 CONSULTING LOG

Expected hours needed: 10

<i>Date</i>	<i>Hours Spent</i>	<i>Tasks / Accomplishments / Issues / Thoughts</i>
03/02/2021	3	I began to get some of the basics down here and actually got rid of some errors that would not let me print. I think the hardest part is reading these ambiguous and long error messages.
03/08/2021	3	Made some progress with functions. It's still giving me trouble but I switched over to a different web based IDE that has some better error messaging because I found ideone.com to be confusing in error messaging with too much on the screen.
03/22/2021	1	I messed around with some things and got frustrated so I moved to start COBOL in order to not waste time dwelling on Fortran.
03/23/2021	2	I finally got the syntax down for what I was doing wrong in the function sections with some new found understanding for what was going on and what some key-words meant. Once the encrypt function was working, decrypt fell into place and then solve came shortly after. The most frustrating thing at the very end was why there were extraneous spaces in the middle of the "Caesar 25-0: ..." part in solve. I could not figure out how to reformat it, but it works and that's what matters.

### 1.2 COMMENTARY

Fortran was very frustrating for me mainly because of the syntax. I found it confusing and not a whole lot of useful documentation that worked in this version. The coding ground website was also more helpful but only for fortran because the error messages were easier to follow, but still confusing compared to something like C or Java. Overall it was not terrible, but not enjoyable.

## 1.3 SOURCE CODE

```
1 !Encrypt
2 function encrypt(testStr, shift) result(encryptedStr)
3     implicit none
4
5     !local variables
6     character(len = 30), intent(in) :: testStr
7     integer, intent(in) :: shift
8
9     !character, dimension(30) :: encryptedChrArr
10    character :: letter
11    integer :: asciiVal, e
12
13    character(len = 30) :: encryptedStr
14
15    encryptedStr = trim(testStr)
16    eloop: do e = 1, len(encryptedStr)
17        !if character from A to Z
18        letter = encryptedStr(e:e)
19        asciiVal = ichar(letter)
20        if((asciiVal >= 65) .and. (asciiVal <= 90)) then
21            if((asciiVal == 65) .and. (shift < 0)) then
22                encryptedStr(e:e) = char(asciiVal + 26 + shift)
23            else if((asciiVal == 90) .and. (shift > 0)) then
24                encryptedStr(e:e) = char(asciiVal - 26 + shift)
25            else
26                encryptedStr(e:e) = char(asciiVal + shift)
27            end if
28        !if character from a to z
29        else if((asciiVal >= 97) .and. (asciiVal <= 122)) then
30            if((asciiVal == 97) .and. (shift < 0)) then
31                encryptedStr(e:e) = char(asciiVal + 26 + shift)
32            else if((asciiVal == 122) .and. (shift > 0)) then
33                encryptedStr(e:e) = char(asciiVal - 26 + shift)
34            else
35                encryptedStr(e:e) = char(asciiVal + shift)
36            end if
37        else
38            encryptedStr(e:e) = letter
39        end if
40    end do eloop
41 end function encrypt
42
43 !Decrypt
44 function decrypt(testStr, shift) result(decryptedStr)
45     implicit none
46
47     !local variables
48     character(len = 30), intent(in) :: testStr
49     integer, intent(in) :: shift
50
51     character :: letter
52     integer :: asciiVal, d
53
54     character(len = 30) :: decryptedStr
55
56     decryptedStr = trim(testStr)
57     dloop: do d = 1, len(decryptedStr)
58         !if character from A to Z
59         letter = decryptedStr(d:d)
60         asciiVal = ichar(letter)
61         if((asciiVal >= 65) .and. (asciiVal <= 90)) then
62             if((asciiVal == 65) .and. (-shift < 0)) then
63                 decryptedStr(d:d) = char(asciiVal + 26 - shift)
```

```

64         else if((asciiVal == 90 ) .and. (-shift > 0)) then
65             decryptedStr(d:d) = char(asciiVal - 26 - shift)
66         else
67             decryptedStr(d:d) = char(asciiVal - shift)
68         end if
69         !if character from a to z
70     else if((asciiVal >= 97) .and. (asciiVal <= 122)) then
71         if((asciiVal == 97) .and. (-shift < 0)) then
72             decryptedStr(d:d) = char(asciiVal + 26 - shift)
73         else if((asciiVal == 122) .and. (-shift > 0)) then
74             decryptedStr(d:d) = char(asciiVal - 26 - shift)
75         else
76             decryptedStr(d:d) = char(asciiVal - shift)
77         end if
78     else
79         decryptedStr(d:d) = letter
80     end if
81 end do dloop
82 end function decrypt
83
84 !Solve
85 function solve(testStr, maxShift) result(solvedStr)
86     implicit none
87
88     !local variables
89     character(len = 30), intent(in) :: testStr
90     integer, intent(in) :: maxShift
91
92     !character, dimension(30) :: solvedChrArr
93     character :: letter
94     integer :: shift, caesarNum
95     integer :: asciiVal, j, s
96
97     character(len = 30) :: solvedStr
98
99     shift = -1
100    solvedStr = trim(testStr)
101    PRINT *, char(9), 'Caesar 26: ', solvedStr
102    jloop: do j = 0, maxShift-1
103        sloop: do s = 1, len(solvedStr)
104            !if character from A to Z
105            letter = solvedStr(s:s)
106            asciiVal = ichar(letter)
107            if((asciiVal >= 65) .and. (asciiVal <= 90)) then
108                if((asciiVal == 65) .and. (shift < 0)) then
109                    solvedStr(s:s) = char(asciiVal + 26 + shift)
110                else if((asciiVal == 90) .and. (shift > 0)) then
111                    solvedStr(s:s) = char(asciiVal - 26 + shift)
112                else
113                    solvedStr(s:s) = char(asciiVal + shift)
114                end if
115            !if character from a to z
116            else if((asciiVal >= 97) .and. (asciiVal <= 122)) then
117                if((asciiVal == 97) .and. (shift < 0)) then
118                    solvedStr(s:s) = char(asciiVal + 26 + shift)
119                else if((asciiVal == 122) .and. (shift > 0)) then
120                    solvedStr(s:s) = char(asciiVal - 26 + shift)
121                else
122                    solvedStr(s:s) = char(asciiVal + shift)
123                end if
124            else
125                solvedStr(s:s) = letter
126            end if
127        end do sloop
128        caesarNum = maxShift-j-1

```

```

129     print *, char(9), 'Caesar ', caesarNum, ': ', solvedStr
130 end do jloop
131 end function solve
132
133 program MAIN
134 !makes sure that variables "i,j,k,l,m,n" are not default to integer
135     implicit none
136
137 !Declare variables
138     character(len = 30) :: encrypt !return type
139     character(len = 30) :: decrypt !return type
140     character(len = 30) :: solve !return type
141
142     character(len = 30) :: testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7, testStr8
143     character(len = 30), dimension(10) :: testStrArr(8) !one dimensional array
144     character(len = 30) :: tempStr, encryptedTempStr, decryptedTempStr, solvedTempStr
145
146     integer :: shiftAmount
147     integer :: maxShiftAmount
148     integer :: i
149
150 !Initialize variables
151     testStr1 = 'IBM'
152     testStr2 = 'Hello World'
153     testStr3 = 'This is a test'
154     testStr4 = 'Hi my name is Robbie'
155     testStr5 = 'WandaVision'
156     testStr6 = 'Abed'
157     testStr7 = 'The Mandalorian'
158     testStr8 = 'Wow I learned Fortran I think'
159
160     shiftAmount = -1
161     maxShiftAmount = 26
162
163
164     testStrArr = (/ testStr1,testStr2,testStr3,testStr4,testStr5,testStr6,testStr7,testStr8 /)
165
166     PRINT *, 'Caesar Cipher'
167     PRINT *, ''
168
169 !Do loop to encrypt, decrypt, and solve
170     iloop: do i = 1, SIZE(testStrArr)
171         tempStr = (testStrArr(i))
172         PRINT *, 'Original String: ', tempStr
173
174         encryptedTempStr = encrypt(tempStr, shiftAmount)
175         PRINT *, 'Encrypted String: ', encryptedTempStr
176         decryptedTempStr = decrypt(encryptedTempStr, shiftAmount)
177         PRINT *, 'Decrypted String: ', decryptedTempStr
178
179         PRINT *, 'Solve: '
180         solvedTempStr = solve(encryptedTempStr, maxShiftAmount)
181         PRINT *, ''
182     end do iloop
183 end program MAIN

```

## 1.4 OUTPUT

```

1  Caesar Cipher
2
3  Original String: IBM
4  Encrypted String: HAL
5  Decrypted String: IBM

```

```

6 Solve:
7 Caesar 26: HAL
8 Caesar 25 : GZK
9 Caesar 24 : FYJ
10 Caesar 23 : EXI
11 Caesar 22 : DWH
12 Caesar 21 : CVG
13 Caesar 20 : BUF
14 Caesar 19 : ATE
15 Caesar 18 : ZSD
16 Caesar 17 : YRC
17 Caesar 16 : XQB
18 Caesar 15 : WPA
19 Caesar 14 : VOZ
20 Caesar 13 : UNY
21 Caesar 12 : TMX
22 Caesar 11 : SLW
23 Caesar 10 : RKV
24 Caesar 9 : QJU
25 Caesar 8 : PIT
26 Caesar 7 : OHS
27 Caesar 6 : NGR
28 Caesar 5 : MFQ
29 Caesar 4 : LEP
30 Caesar 3 : KDO
31 Caesar 2 : JCN
32 Caesar 1 : IBM
33 Caesar 0 : HAL
34
35 Original String: Hello World
36 Encrypted String: Gdkkn Vnqkc
37 Decrypted String: Hello World
38 Solve:
39 Caesar 26: Gdkkn Vnqkc
40 Caesar 25 : Fcjjm Umpjb
41 Caesar 24 : Ebiil Tloia
42 Caesar 23 : Dahhk Sknhz
43 Caesar 22 : Czggj Rjmgy
44 Caesar 21 : Byffi Qilfx
45 Caesar 20 : Axeeh Phkew
46 Caesar 19 : Zwddg Ogjdv
47 Caesar 18 : Yvccf Nficu
48 Caesar 17 : Xubbe Mehbt
49 Caesar 16 : Wtaad Ldgas
50 Caesar 15 : Vszzc Kcfzr
51 Caesar 14 : Uryyb Jbeyq
52 Caesar 13 : Tqxxa Iadxp
53 Caesar 12 : Spwwz Hzcwo
54 Caesar 11 : Rovvy Gybv n
55 Caesar 10 : Qnuux Fxaum
56 Caesar 9 : Pmttw Ewztl
57 Caesar 8 : Olssv Dvysk
58 Caesar 7 : Nkrru Cuxrj
59 Caesar 6 : Mjqqt Btwqi
60 Caesar 5 : Lipps Asvph
61 Caesar 4 : Khoor Zruog
62 Caesar 3 : Jgnnq Yqtnf
63 Caesar 2 : Ifmmp Xpsme
64 Caesar 1 : Hello World
65 Caesar 0 : Gdkkn Vnqkc
66
67 Original String: This is a test
68 Encrypted String: Sghr hr z sdrs
69 Decrypted String: This is a test
70 Solve:

```

```

71 Caesar 26: Sghr hr z sdrs
72 Caesar      25 : Rfgq gq y rcqr
73 Caesar      24 : Qefp fp x qbpq
74 Caesar      23 : Pdeo eo w paop
75 Caesar      22 : Ocdn dn v ozno
76 Caesar      21 : Nbcm cm u nymn
77 Caesar      20 : Mabl bl t mxlm
78 Caesar      19 : Lzak ak s lwkl
79 Caesar      18 : Kyzj zj r kvjk
80 Caesar      17 : Jxyi yi q juij
81 Caesar      16 : Iwxh xh p ithi
82 Caesar      15 : Hvwg wg o hsgh
83 Caesar      14 : Guvf vf n grfg
84 Caesar      13 : Ftue ue m fgef
85 Caesar      12 : Estd td l epde
86 Caesar      11 : Drsc sc k docd
87 Caesar      10 : Cqrb rb j cnbc
88 Caesar       9 : Bpqa qa i bmab
89 Caesar       8 : Aopz pz h alza
90 Caesar       7 : Znoy oy g zkyz
91 Caesar       6 : Ymnx nx f yjxy
92 Caesar       5 : Xlmw mw e xiwx
93 Caesar       4 : Wklv lv d whvw
94 Caesar       3 : Vjku ku c vguv
95 Caesar       2 : Uijt jt b uftu
96 Caesar       1 : This is a test
97 Caesar       0 : Sghr hr z sdrs
98
99 Original String: Hi my name is Robbie
100 Encrypted String: Gh lx mzld hr Qnaahd
101 Decrypted String: Hi my name is Robbie
102 Solve:
103 Caesar 26: Gh lx mzld hr Qnaahd
104 Caesar      25 : Fg kw lykc gq Pmzzgc
105 Caesar      24 : Ef jv kxjb fp Olyyfb
106 Caesar      23 : De iu jwia eo Nkxxea
107 Caesar      22 : Cd ht ivhz dn Mjwwdz
108 Caesar      21 : Bc gs hugy cm Livvcy
109 Caesar      20 : Ab fr gtfx bl Khuubx
110 Caesar      19 : Za eq fsew ak Jgttaw
111 Caesar      18 : Yz dp erdv zj Ifsszv
112 Caesar      17 : Xy co dqcu yi Herryu
113 Caesar      16 : Wx bn cpbt xh Gdqqxt
114 Caesar      15 : Vw am boas wg Fcppws
115 Caesar      14 : Uv zl anzr vf Eboovr
116 Caesar      13 : Tu yk zmyq ue Dannuq
117 Caesar      12 : St xj ylxp td Czmmtp
118 Caesar      11 : Rs wi xkwo sc Byllso
119 Caesar      10 : Qr vh wjvn rb Axkkrn
120 Caesar       9 : Pq ug vium qa Zwjjqm
121 Caesar       8 : Op tf uhtl pz Yviipl
122 Caesar       7 : No se tgsk oy Xuhhok
123 Caesar       6 : Mn rd sfrj nx Wtggnj
124 Caesar       5 : Lm qc reqi mw Vsffmi
125 Caesar       4 : Kl pb qdph lv Ureelh
126 Caesar       3 : Jk oa pcog ku Tqddkg
127 Caesar       2 : Ij nz obnf jt Spccjf
128 Caesar       1 : Hi my name is Robbie
129 Caesar       0 : Gh lx mzld hr Qnaahd
130
131 Original String: WandaVision
132 Encrypted String: VmzczUhrhnm
133 Decrypted String: WandaVision
134 Solve:
135 Caesar 26: VmzczUhrhnm

```

```

136 Caesar 25 : UylbyTgqgm1
137 Caesar 24 : TxxaxSfpflk
138 Caesar 23 : SwjzwReoekj
139 Caesar 22 : RviyvQdndji
140 Caesar 21 : QuhxuPcmcih
141 Caesar 20 : PtgwtOblbhg
142 Caesar 19 : OsfvsNakagf
143 Caesar 18 : NreurMzjzfe
144 Caesar 17 : MqdtqLyiyed
145 Caesar 16 : LpcspKxhxdc
146 Caesar 15 : KobroJwgwcb
147 Caesar 14 : JnaqnIvfvba
148 Caesar 13 : ImzpmHueuaz
149 Caesar 12 : HlyolGtdtzy
150 Caesar 11 : GkxnfFscsyx
151 Caesar 10 : FjwmjErbrxw
152 Caesar 9 : EivliDqaqvw
153 Caesar 8 : DhukhCpzipvu
154 Caesar 7 : CgtjgBoyout
155 Caesar 6 : BfsifAnxnts
156 Caesar 5 : AerheZmwmsr
157 Caesar 4 : ZdqgdYlvlrq
158 Caesar 3 : YcpfcXkukqp
159 Caesar 2 : XboebWjtjpo
160 Caesar 1 : WandaVision
161 Caesar 0 : VzmczUhrhnm

```

```

162
163 Original String: Abed
164 Encrypted String: Zadc
165 Decrypted String: Abed
166 Solve:

```

```

167 Caesar 26: Zadc
168 Caesar 25 : Yzcb
169 Caesar 24 : Xyba
170 Caesar 23 : Wxaz
171 Caesar 22 : Vwzy
172 Caesar 21 : Uvyx
173 Caesar 20 : Tuxw
174 Caesar 19 : Stwv
175 Caesar 18 : Rsvu
176 Caesar 17 : Qrut
177 Caesar 16 : Pqts
178 Caesar 15 : Opsr
179 Caesar 14 : Norq
180 Caesar 13 : Mnqp
181 Caesar 12 : Lmpo
182 Caesar 11 : Klon
183 Caesar 10 : Jknm
184 Caesar 9 : Ijml
185 Caesar 8 : HilK
186 Caesar 7 : Ghkj
187 Caesar 6 : Fgji
188 Caesar 5 : EfiH
189 Caesar 4 : Dehg
190 Caesar 3 : Cdgf
191 Caesar 2 : Bcfe
192 Caesar 1 : Abed
193 Caesar 0 : Zadc

```

```

194
195 Original String: The Mandalorian
196 Encrypted String: Sgd LzmczknqhzM
197 Decrypted String: The Mandalorian
198 Solve:

```

```

199 Caesar 26: Sgd LzmczknqhzM
200 Caesar 25 : Rfc KylbyjMpgyl

```

```

201 Caesar 24 : Qeb Jxkaxilofxk
202 Caesar 23 : Pda Iwjzwhknewj
203 Caesar 22 : Ocz Hviyvgjmdvi
204 Caesar 21 : Nby Guhxufilcuh
205 Caesar 20 : Max Ftgwtehkbttg
206 Caesar 19 : Lzw Esfvsdgjasf
207 Caesar 18 : Kyv Dreurcfizre
208 Caesar 17 : Jxu Cqdtqbehyqd
209 Caesar 16 : Iwt Bpcspadgxp
210 Caesar 15 : Hvs Aobrozcfwob
211 Caesar 14 : Gur Znaqnybevna
212 Caesar 13 : Ftq Ymzpmxadumz
213 Caesar 12 : Esp Xlyolwzctly
214 Caesar 11 : Dro Wkxkvybskx
215 Caesar 10 : Cqn Vjwmjuxarjw
216 Caesar 9 : Bpm Uivlitwzqiv
217 Caesar 8 : Aol Thukhsvyphu
218 Caesar 7 : Znk Sgtjgruxogt
219 Caesar 6 : Ymj Rfsifqtnfs
220 Caesar 5 : Xli Qerhepsvmer
221 Caesar 4 : Wkh Pdqdgoruldq
222 Caesar 3 : Vjg Ocpfcnqtkcp
223 Caesar 2 : Uif Nboebmpsjo
224 Caesar 1 : The Mandalorian
225 Caesar 0 : Sgd Lzmczknqzhm
226
227 Original String: Wow I learned Fortran I think
228 Encrypted String: Vnv H kdzqmdc Enqsqzm H sghmj
229 Decrypted String: Wow I learned Fortran I think
230 Solve:
231 Caesar 26: Vnv H kdzqmdc Enqsqzm H sghmj
232 Caesar 25 : Umu G jcyplcb Dmppy l G rfgli
233 Caesar 24 : Tlt F ibxokba Cloqoxk F qefkh
234 Caesar 23 : SkS E hawnjaz Bknpnwj E pdejg
235 Caesar 22 : Rjr D gzvmizy Ajmomvi D oc dif
236 Caesar 21 : Qiq C fyulhyx Zilnluh C nbche
237 Caesar 20 : Php B extkgxw Yhkmktg B mabgd
238 Caesar 19 : Ogo A dwsjfwv Xgjljsf A lzafc
239 Caesar 18 : Nfn Z cvrievu Wfikire Z kyzeb
240 Caesar 17 : Mem Y buqh dut Vehjhqd Y jxyda
241 Caesar 16 : Ldl X atpgcts Udgigpc X iwxcz
242 Caesar 15 : Kck W zsofbsr Tcfhfob W hvwby
243 Caesar 14 : Jbj V yrnearq Sbegen a V guvax
244 Caesar 13 : Iai U xqmdzqp Radfdmz U ftuzw
245 Caesar 12 : Hzh T wplcypo Qzcecly T estyv
246 Caesar 11 : Gyg S vokbxon Pybdbkx S drsxu
247 Caesar 10 : Fxf R unjawnm Oxacajw R cqrwt
248 Caesar 9 : Ewe Q tmizvml Nwzbziv Q bpqvs
249 Caesar 8 : Dvd P slhyulk Mvyayhu P aopur
250 Caesar 7 : Cuc O rkgxtkj Luxzxgt O znotq
251 Caesar 6 : Btb N qjfwjsi Ktwy wfs N ymns p
252 Caesar 5 : Asa M pievrih Jsvxver M xlmro
253 Caesar 4 : Zrz L ohduqhg Iruwudq L wklqn
254 Caesar 3 : Yqy K ngctpgf Hqvtvcp K vjkpm
255 Caesar 2 : Xpx J mfbsofe Gpsusbo J uijol
256 Caesar 1 : Wow I learned Fortran I think
257 Caesar 0 : Vnv H kdzqmdc Enqsqzm H sghmj

```



## 2 COBOL

### 2.1 CONSULTING LOG

Expected hours needed: 12

<i>Date</i>	<i>Hours Spent</i>	<i>Tasks / Accomplishments / Issues / Thoughts</i>
03/22/2021	4	Spent a lot of time looking over the basic syntax and learning the program structures with the different areas of the overall program.
03/23/2021	2	I learned a good amount about the tables and how they work. They are still a bit weird and I mostly understand it, but it makes less sense to me in this context and more sense in the other examples I've seen like being able to hold costumer data.
03/25/2021	4	I made some good progress and went step by step more than I have been with any of the other languages, seeing what worked and what didn't. It made things easier and seemingly take less time. I made good progress on the encrypt subprogram and now it's just time to debug it and make it better. I am still struggling on how to trim the strings down with the trailing spaces. Nothing is seeming to work in this version that I found. I also finished the decrypt function, got out the bugs and the solve function is done, but buggy.
03/26/2021	1	I got a lot more done yesterday than I thought I would. I am still having trouble with the solve function because it is only printing out the first 2 letters for each shift on every single string. The letters are correct in the shift, but most of the updated string is lost in the ether and I don't know why.

### 2.2 COMMENTARY

I found COBOL to actually be slightly more tolerable than Fortran in some ways. Although this felt nothing like any of the other programming languages, I think that actually helped in some ways, because I did not get as hung up on trying to write it like the other languages. At the same time, the syntax is even more confusing than Fortran's in a lot of ways and parts of it I did not understand. I think it would not have been that bad, but the IF statements and PERFORM keyword did not work the same in the version on ideone.com as the resources I saw. This version that they had there was more frustrating to work with in many ways that would not have happened if it was a different version of COBOL. There did not seem to be else if statements or else statements. Unless I was just writing them wrong, they did not work in any of the ways I had tried. Perform also did not work for me outside of paragraphs.

### 2.3 SOURCE CODE

```
1 $ SET SOURCEFORMAT "FREE"
2
3 IDENTIFICATION DIVISION.
4 PROGRAM-ID. IDEONE.
5 AUTHOR. Robert Perrone.
6 DATE-WRITTEN. March 22nd 2021
7 *>ENVIRONMENT DIVISION.
8
9 DATA DIVISION.
10 WORKING-STORAGE SECTION.
11 *> Original String Table
12 01 testStrTable.*> PIC X(30).
13     02 testStr1 PIC X(30) VALUE "IBM".
14     02 testStr2 PIC X(30) VALUE "Hello World".
```

```

15      02 testStr3 PIC X(30) VALUE "This is a test".
16      02 testStr4 PIC X(30) VALUE "Hi my name is Robbie".
17      02 testStr5 PIC X(30) VALUE "WandaVision".
18      02 testStr6 PIC X(30) VALUE "Abed".
19      02 testStr7 PIC X(30) VALUE "The Mandalorian".
20      02 testStr8 PIC X(30) VALUE "Wow I learned COBOL I think".
21 01 FILLER REDEFINES testStrTable.
22      02 testStrTb OCCURS 8 INDEXED BY I.
23          03 testStrTb OCCURS 1 TIMES.
24              04 testStr PIC X(30).
25
26 *> Encrypted String Table
27 01 eTestStrTable PIC X(30).
28      02 eTestStr1 PIC X(30) VALUE "".
29      02 eTestStr2 PIC X(30) VALUE "".
30      02 eTestStr3 PIC X(30) VALUE "".
31      02 eTestStr4 PIC X(30) VALUE "".
32      02 eTestStr5 PIC X(30) VALUE "".
33      02 eTestStr6 PIC X(30) VALUE "".
34      02 eTestStr7 PIC X(30) VALUE "".
35      02 eTestStr8 PIC X(30) VALUE "".
36 01 FILLER REDEFINES eTestStrTable.
37      02 eTestStrTb OCCURS 8 TIMES INDEXED BY J.
38          03 eTestStrTb OCCURS 1 TIMES.
39              04 eTestStr PIC X(30).
40
41 01 shiftAmount PIC S9(4) VALUE -1.
42 01 maxShiftAmount PIC 9(2) VALUE 26.
43
44 01 tempStr PIC X(30) VALUE "".
45 01 encryptedStr PIC X(30) VALUE "".
46 01 decryptedStr PIC X(30) VALUE "".
47 01 solvedStr PIC X(30) VALUE "".
48
49 *>01 I PIC 9(1) VALUE 1.
50 *>01 J PIC 9(1) VALUE 1.
51
52 PROCEDURE DIVISION.
53
54 *>      your code goes here
55      DISPLAY "Caesar Cipher ".
56      DISPLAY "".
57
58      SET I TO 1.
59      SET J TO 1.
60      begin.
61          PERFORM iForLoop VARYING I FROM 1 BY 1 UNTIL I>8.
62          STOP RUN.
63
64      iForLoop.
65          MOVE Function UPPER-CASE(testStr(I)) TO tempStr.
66          DISPLAY "Original String: " tempStr.
67
68          CALL "ENCRYPT" USING tempStr, shiftAmount, encryptedStr.
69          DISPLAY "Encrypted String: " encryptedStr.
70
71          CALL "DECRYPT" USING encryptedStr, shiftAmount, decryptedStr.
72          DISPLAY "Decrypted String: " decryptedStr.
73
74          DISPLAY "Solve:".
75          CALL "SOLVE" USING encryptedStr, maxShiftAmount, solvedStr.
76          DISPLAY "".
77          *>MOVE Function solve(encryptedTempStr, maxShiftAmount) TO decryptedTempStr.
78  END PROGRAM IDEONE.
79

```

```

80 *>Encrypt Subprogram
81 IDENTIFICATION DIVISION.
82 PROGRAM-ID. ENCRYPT.
83 DATA DIVISION.
84 WORKING-STORAGE SECTION.
85 01 letter PIC X(1).
86 01 newLetter PIC X(1).
87 01 asciiVal PIC 9(2).
88 01 newAsciiVal PIC 9(2).
89
90 01 E PIC 9(2) VALUE 1.
91 01 len PIC 9(3) VALUE 0.
92
93 LINKAGE SECTION.
94 01 tempStr PIC X(30).
95 01 shiftAmount PIC S9(4).
96 01 encryptedStr PIC X(30).
97
98 PROCEDURE DIVISION USING tempStr, shiftAmount, encryptedStr.
99     *>MOVE "returned" TO encryptedStr.
100     *>DISPLAY "ENCRYPT".
101     begin.
102         SET shiftAmount TO -1.
103         MOVE Function LENGTH(tempStr) TO len.
104         *>DISPLAY len.
105         PERFORM eForLoop VARYING E FROM 1 BY 1 UNTIL E > len.
106         EXIT PROGRAM.
107     eForLoop.
108         MOVE tempStr(E:E) TO letter.
109         *>DISPLAY letter.
110         MOVE Function ORD(letter) TO asciiVal.
111         *>COMPUTE asciiVal = asciiVal - 1.
112         *>DISPLAY asciiVal.
113
114         *> If character from A to Z
115         IF (asciiVal >= 66) AND (asciiVal <= 91) THEN
116             *>If character is A and shift is negative, wrap around
117             IF (asciiVal = 66) AND (shiftAmount < 0) THEN
118                 COMPUTE newAsciiVal = asciiVal + 26 + shiftAmount.
119                 MOVE Function CHAR(newAsciiVal) TO newLetter.
120                 MOVE newLetter TO encryptedStr(E:E).
121
122             *>If character is Z and shift is positive, wrap around
123             IF ((asciiVal = 91) AND (shiftAmount > 0)) THEN
124                 COMPUTE newAsciiVal = asciiVal - 26 + shiftAmount.
125                 MOVE Function CHAR(newAsciiVal) TO newLetter.
126                 MOVE newLetter TO encryptedStr(E:E).
127
128             *>ELSE
129             IF NOT ((asciiVal = 66) AND (shiftAmount < 0)) AND NOT ((asciiVal = 91) AND (shiftAmount > 0)) THEN
130                 *>DISPLAY letter.
131                 *>DISPLAY asciiVal.
132                 COMPUTE newAsciiVal = asciiVal + shiftAmount.
133                 *>DISPLAY shiftAmount.
134                 *>DISPLAY newAsciiVal.
135                 MOVE Function CHAR(newAsciiVal) TO newLetter.
136                 MOVE newLetter TO encryptedStr(E:E).
137                 *>DISPLAY encryptedStr(E:E).
138
139             IF NOT ((asciiVal >= 66) AND (asciiVal <= 91)) THEN
140                 *>If character is space or other
141                 MOVE letter TO encryptedStr(E:E).
142
143 END PROGRAM ENCRYPT.
144 *>Decrypt Subprogram

```

```

145 IDENTIFICATION DIVISION.
146 PROGRAM-ID. DECRYPT.
147 DATA DIVISION.
148 WORKING-STORAGE SECTION.
149 01 letter PIC X(1).
150 01 newLetter PIC X(1).
151 01 asciiVal PIC 9(2).
152 01 newAsciiVal PIC 9(2).
153
154 01 D PIC 9(2) VALUE 1.
155 01 len PIC 9(3) VALUE 0.
156
157 LINKAGE SECTION.
158 01 encryptedStr PIC X(30).
159 01 shiftAmount PIC S9(4).
160 01 decryptedStr PIC X(30).
161
162 PROCEDURE DIVISION USING encryptedStr, shiftAmount, decryptedStr.
163     *>MOVE "returned" TO encryptedStr.
164     *>DISPLAY "DECRYPT".
165     begin.
166         MOVE Function LENGTH(encryptedStr) TO len.
167         *>DISPLAY len.
168         MULTIPLY shiftAmount BY -1 GIVING shiftAmount.
169         PERFORM dForLoop VARYING D FROM 1 BY 1 UNTIL D > len.
170         EXIT PROGRAM.
171     dForLoop.
172         MOVE encryptedStr(D:D) TO letter.
173         *>DISPLAY letter.
174         MOVE Function ORD(letter) TO asciiVal.
175         *>COMPUTE asciiVal = asciiVal - 1.
176         *>DISPLAY asciiVal.
177         *> If character from A to Z
178
179         *>Make shift amount opposite sign
180
181
182         IF (asciiVal) >= 66 AND (asciiVal <= 91) THEN
183             *>If character is A and shift is negative, wrap around
184             IF (asciiVal = 66) AND (shiftAmount < 0) THEN
185                 COMPUTE newAsciiVal = asciiVal + 26 + shiftAmount.
186                 MOVE Function CHAR(newAsciiVal) TO newLetter.
187                 MOVE newLetter TO decryptedStr(D:D).
188
189             *>If character is Z and shift is positive, wrap around
190             IF (asciiVal = 91) AND (shiftAmount > 0) THEN
191                 COMPUTE newAsciiVal = asciiVal - 26 + shiftAmount.
192                 MOVE Function CHAR(newAsciiVal) TO newLetter.
193                 MOVE newLetter TO decryptedStr(D:D).
194             *>ELSE
195             IF NOT ((asciiVal = 66) AND (shiftAmount < 0)) AND NOT ((asciiVal = 91) AND (shiftA
196                 *>DISPLAY letter.
197                 *>DISPLAY asciiVal.
198                 COMPUTE newAsciiVal = asciiVal + shiftAmount.
199                 *>DISPLAY shiftAmount.
200                 *>DISPLAY newAsciiVal.
201                 MOVE Function CHAR(newAsciiVal) TO newLetter.
202                 MOVE newLetter TO decryptedStr(D:D).
203                 *>DISPLAY encryptedStr(D:D).
204
205             IF NOT ((asciiVal >= 66) AND (asciiVal <= 91)) THEN
206                 *>If character is space or other
207                 MOVE letter TO decryptedStr(D:D).
208
209 END PROGRAM DECRYPT.

```

```

210
211 *>Solve Subprogram
212 IDENTIFICATION DIVISION.
213 PROGRAM-ID. SOLVE.
214 DATA DIVISION.
215 WORKING-STORAGE SECTION.
216 01 letter PIC X(1).
217 01 newLetter PIC X(1).
218 01 asciiVal PIC 9(2).
219 01 newAsciiVal PIC 9(2).
220 01 caesarNum PIC 9(2).
221 01 shiftAmount PIC S9(4) VALUE -1.
222
223 01 J PIC 9(2) VALUE 1.
224 01 S PIC 9(2) VALUE 1.
225 01 len PIC 9(3) VALUE 0.
226
227 LINKAGE SECTION.
228 01 encryptedStr PIC X(30).
229 01 maxShiftAmount PIC 9(2).
230 01 solvedStr PIC X(30).
231
232 PROCEDURE DIVISION USING encryptedStr, maxShiftAmount, solvedStr.
233     *>MOVE "returned" TO solvedStr.
234     *>DISPLAY "SOLVE".
235     begin.
236         MOVE Function LENGTH(solvedStr) TO len.
237         SET shiftAmount TO -1.
238         MOVE encryptedStr TO solvedStr.
239         DISPLAY "Caesar 26: " solvedStr.
240         PERFORM jForLoop VARYING J FROM 1 BY 1 UNTIL J > maxShiftAmount.
241         EXIT PROGRAM.
242
243     jForLoop.
244         PERFORM sForLoop VARYING S FROM 1 BY 1 UNTIL S > len.
245         COMPUTE caesarNum = maxShiftAmount - j.
246         DISPLAY "Caesar " caesarNum ": " solvedStr.
247
248     sForLoop.
249         MOVE solvedStr(S:S) TO letter.
250         *>DISPLAY letter.
251         MOVE Function ORD(letter) TO asciiVal.
252         *>COMPUTE asciiVal = asciiVal - 1.
253         *>DISPLAY asciiVal.
254         *> If character from A to Z
255         IF (asciiVal >= 66) AND (asciiVal <= 91) THEN
256             *>If character is A and shift is negative, wrap around
257             IF (asciiVal = 66) AND (shiftAmount < 0) THEN
258                 COMPUTE newAsciiVal = asciiVal + 26 + shiftAmount.
259                 MOVE Function CHAR(newAsciiVal) TO newLetter.
260                 MOVE newLetter TO solvedStr(S:S).
261             *>If character is Z and shift is positive, wrap around
262             IF (asciiVal = 91) AND (shiftAmount > 0) THEN
263                 COMPUTE newAsciiVal = asciiVal - 26 + shiftAmount.
264                 MOVE Function CHAR(newAsciiVal) TO newLetter.
265                 MOVE newLetter TO solvedStr(S:S).
266             *>ELSE
267             IF NOT ((asciiVal = 66) AND (shiftAmount < 0)) AND NOT ((asciiVal = 91) AND (shiftA
268                 *>DISPLAY letter.
269                 *>DISPLAY asciiVal.
270                 COMPUTE newAsciiVal = asciiVal + shiftAmount.
271                 *>DISPLAY shiftAmount.
272                 *>DISPLAY newAsciiVal.
273                 MOVE Function CHAR(newAsciiVal) TO newLetter.
274                 MOVE newLetter TO solvedStr(S:S).

```

```

275         IF NOT ((asciiVal >= 66) AND (asciiVal <= 91)) THEN
276             *>If character is space or other
277             MOVE letter TO solvedStr(S:S).
278
279 END PROGRAM SOLVE.

```

## 2.4 OUTPUT

```

1 Caesar Cipher
2
3 Original String: IBM
4 Encrypted String: HAL
5 Decrypted String: IBM
6 Solve:
7 Caesar 26: HAL
8 Caesar 25: GZ
9 Caesar 24: FY
10 Caesar 23: EX
11 Caesar 22: DW
12 Caesar 21: CV
13 Caesar 20: BU
14 Caesar 19: AT
15 Caesar 18: ZS
16 Caesar 17: YR
17 Caesar 16: XQ
18 Caesar 15: WP
19 Caesar 14: VO
20 Caesar 13: UN
21 Caesar 12: TM
22 Caesar 11: SL
23 Caesar 10: RK
24 Caesar 09: QJ
25 Caesar 08: PI
26 Caesar 07: OH
27 Caesar 06: NG
28 Caesar 05: MF
29 Caesar 04: LE
30 Caesar 03: KD
31 Caesar 02: JC
32 Caesar 01: IB
33 Caesar 00: HA
34
35 Original String: HELLO WORLD
36 Encrypted String: GDKKN VNQKC
37 Decrypted String: HELLO WORLD
38 Solve:
39 Caesar 26: GDKKN VNQKC
40 Caesar 25: FC
41 Caesar 24: EB
42 Caesar 23: DA
43 Caesar 22: CZ
44 Caesar 21: BY
45 Caesar 20: AX
46 Caesar 19: ZW
47 Caesar 18: YV
48 Caesar 17: XU
49 Caesar 16: WT
50 Caesar 15: VS
51 Caesar 14: UR
52 Caesar 13: TQ
53 Caesar 12: SP
54 Caesar 11: RO
55 Caesar 10: QN

```

```

56 Caesar 09: PM
57 Caesar 08: OL
58 Caesar 07: NK
59 Caesar 06: MJ
60 Caesar 05: LI
61 Caesar 04: KH
62 Caesar 03: JG
63 Caesar 02: IF
64 Caesar 01: HE
65 Caesar 00: GD
66
67 Original String: THIS IS A TEST
68 Encrypted String: SGHR HR Z SDRS
69 Decrypted String: THIS IS A TEST
70 Solve:
71 Caesar 26: SGHR HR Z SDRS
72 Caesar 25: RF
73 Caesar 24: QE
74 Caesar 23: PD
75 Caesar 22: OC
76 Caesar 21: NB
77 Caesar 20: MA
78 Caesar 19: LZ
79 Caesar 18: KY
80 Caesar 17: JX
81 Caesar 16: IW
82 Caesar 15: HV
83 Caesar 14: GU
84 Caesar 13: FT
85 Caesar 12: ES
86 Caesar 11: DR
87 Caesar 10: CQ
88 Caesar 09: BP
89 Caesar 08: AO
90 Caesar 07: ZN
91 Caesar 06: YM
92 Caesar 05: XL
93 Caesar 04: WK
94 Caesar 03: VJ
95 Caesar 02: UI
96 Caesar 01: TH
97 Caesar 00: SG
98
99 Original String: HI MY NAME IS ROBBIE
100 Encrypted String: GH LX MZLD HR QNAAHD
101 Decrypted String: HI MY NAME IS ROBBIE
102 Solve:
103 Caesar 26: GH LX MZLD HR QNAAHD
104 Caesar 25: FG
105 Caesar 24: EF
106 Caesar 23: DE
107 Caesar 22: CD
108 Caesar 21: BC
109 Caesar 20: AB
110 Caesar 19: ZA
111 Caesar 18: YZ
112 Caesar 17: XY
113 Caesar 16: WX
114 Caesar 15: VW
115 Caesar 14: UV
116 Caesar 13: TU
117 Caesar 12: ST
118 Caesar 11: RS
119 Caesar 10: QR
120 Caesar 09: PQ

```

121 Caesar 08: OP  
122 Caesar 07: NO  
123 Caesar 06: MN  
124 Caesar 05: LM  
125 Caesar 04: KL  
126 Caesar 03: JK  
127 Caesar 02: IJ  
128 Caesar 01: HI  
129 Caesar 00: GH  
130  
131 Original String: WANDAVISION  
132 Encrypted String: VZMCZUHRHNM  
133 Decrypted String: WANDAVISION  
134 Solve:  
135 Caesar 26: VZMCZUHRHNM  
136 Caesar 25: UY  
137 Caesar 24: TX  
138 Caesar 23: SW  
139 Caesar 22: RV  
140 Caesar 21: QU  
141 Caesar 20: PT  
142 Caesar 19: OS  
143 Caesar 18: NR  
144 Caesar 17: MQ  
145 Caesar 16: LP  
146 Caesar 15: KO  
147 Caesar 14: JN  
148 Caesar 13: IM  
149 Caesar 12: HL  
150 Caesar 11: GK  
151 Caesar 10: FJ  
152 Caesar 09: EI  
153 Caesar 08: DH  
154 Caesar 07: CG  
155 Caesar 06: BF  
156 Caesar 05: AE  
157 Caesar 04: ZD  
158 Caesar 03: YC  
159 Caesar 02: XB  
160 Caesar 01: WA  
161 Caesar 00: VZ  
162  
163 Original String: ABED  
164 Encrypted String: ZADC  
165 Decrypted String: ABED  
166 Solve:  
167 Caesar 26: ZADC  
168 Caesar 25: YZ  
169 Caesar 24: XY  
170 Caesar 23: WX  
171 Caesar 22: VW  
172 Caesar 21: UV  
173 Caesar 20: TU  
174 Caesar 19: ST  
175 Caesar 18: RS  
176 Caesar 17: QR  
177 Caesar 16: PQ  
178 Caesar 15: OP  
179 Caesar 14: NO  
180 Caesar 13: MN  
181 Caesar 12: LM  
182 Caesar 11: KL  
183 Caesar 10: JK  
184 Caesar 09: IJ  
185 Caesar 08: HI



```

186 Caesar 07: GH
187 Caesar 06: FG
188 Caesar 05: EF
189 Caesar 04: DE
190 Caesar 03: CD
191 Caesar 02: BC
192 Caesar 01: AB
193 Caesar 00: ZA
194
195 Original String: THE MANDALORIAN
196 Encrypted String: SGD LZMCZKNQHZM
197 Decrypted String: THE MANDALORIAN
198 Solve:
199 Caesar 26: SGD LZMCZKNQHZM
200 Caesar 25: RF
201 Caesar 24: QE
202 Caesar 23: PD
203 Caesar 22: OC
204 Caesar 21: NB
205 Caesar 20: MA
206 Caesar 19: LZ
207 Caesar 18: KY
208 Caesar 17: JX
209 Caesar 16: IW
210 Caesar 15: HV
211 Caesar 14: GU
212 Caesar 13: FT
213 Caesar 12: ES
214 Caesar 11: DR
215 Caesar 10: CQ
216 Caesar 09: BP
217 Caesar 08: AO
218 Caesar 07: ZN
219 Caesar 06: YM
220 Caesar 05: XL
221 Caesar 04: WK
222 Caesar 03: VJ
223 Caesar 02: UI
224 Caesar 01: TH
225 Caesar 00: SG
226
227 Original String: WOW I LEARNED COBOL I THINK
228 Encrypted String: VNV H KDZQMDC BNANK H SGHMJ
229 Decrypted String: WOW I LEARNED COBOL I THINK
230 Solve:
231 Caesar 26: VNV H KDZQMDC BNANK H SGHMJ
232 Caesar 25: UM
233 Caesar 24: TL
234 Caesar 23: SK
235 Caesar 22: RJ
236 Caesar 21: QI
237 Caesar 20: PH
238 Caesar 19: OG
239 Caesar 18: NF
240 Caesar 17: ME
241 Caesar 16: LD
242 Caesar 15: KC
243 Caesar 14: JB
244 Caesar 13: IA
245 Caesar 12: HZ
246 Caesar 11: GY
247 Caesar 10: FX
248 Caesar 09: EW
249 Caesar 08: DV
250 Caesar 07: CU

```

```

251 Caesar 06: BT
252 Caesar 05: AS
253 Caesar 04: ZR
254 Caesar 03: YQ
255 Caesar 02: XP
256 Caesar 01: WO
257 Caesar 00: VN

```

## 3 BASIC

### 3.1 CONSULTING LOG

Expected hours needed: 6

<i>Date</i>	<i>Hours Spent</i>	<i>Tasks / Accomplishments / Issues / Thoughts</i>
03/19/2021	1	Spent some time learning the syntax, and setting up the base of the code. Also started the encrypt but did not finish it yet.
03/20/2021	2	I really did not have much debugging to do with Visual Basic. It worked exactly how I thought I would and things just felt intuitive and natural. There was also a good amount of resources online to be able to figure out syntax and helpful functions.

### 3.2 COMMENTARY

I personally liked Visual Basic the best, not only because it took me the least amount of time to complete, but because it felt intuitive as I was writing it. It translated really well in my head when I was writing it although I am not sure if that is just because I already wrote the code 2 other times and spent a lot more time debugging the other ones. Also I think what made it the most enjoyable was the amount of documentation online for this version. All the documentation was really helpful and easy to follow. I would definitely base my own language off of some of the syntax and functions in this language.

### 3.3 SOURCE CODE

```

1 Imports System
2 Module Test
3     Function encrypt(testStr as string, shift as integer) as string
4         DIM encryptedStr as string
5         DIM e as integer
6
7         for e = 0 to len(testStr)-1
8             DIM letter as char = testStr(e)
9             if((Asc(letter) >= 65) and (Asc(letter) <= 90)) then
10                 if((Asc(letter) = 65) and (shift < 0)) then
11                     encryptedStr += Chr(Asc(letter) + 26 + shift)
12                 else if((Asc(letter) = 90) and (shift > 0)) then
13                     encryptedStr += Chr(Asc(letter) - 26 + shift)
14                 else
15                     encryptedStr += Chr(Asc(letter) + shift)
16                 End if
17             Else
18                 encryptedStr += letter
19             End if
20         Next

```

```

21         encrypt = encryptedStr
22     End Function
23
24     Function decrypt(testStr as string, shift as integer) as string
25         DIM decryptedStr as string
26         DIM d as integer
27
28         for d = 0 to len(testStr)-1
29             DIM letter as char = testStr(d)
30             if((Asc(letter) >= 65) and (Asc(letter) <= 90)) then
31                 if((Asc(letter) = 65) and (-shift < 0)) then
32                     decryptedStr += Chr(Asc(letter) + 26 - shift)
33                 else if((Asc(letter) = 90) and (-shift > 0)) then
34                     decryptedStr += Chr(Asc(letter) - 26 - shift)
35                 else
36                     decryptedStr += Chr(Asc(letter) - shift)
37                 End if
38             Else
39                 decryptedStr += letter
40             End if
41         Next
42         decrypt = decryptedStr
43     End Function
44
45     Function solve(testStr as string, maxShift as integer) as string
46         DIM solvedChrArr() as char = testStr.ToCharArray
47         DIM solvedStr as string
48         DIM shift as integer = -1
49         DIM j as integer
50         DIM s as integer
51
52         Console.WriteLine(vbTab & "Caesar 26: " + testStr)
53         for j = 0 to maxShift-1
54             for s = 0 to len(testStr)-1
55                 DIM letter as char = solvedChrArr(s)
56                 if((Asc(letter) >= 65) and (Asc(letter) <= 90)) then
57                     if((Asc(letter) = 65) and (shift < 0)) then
58                         solvedChrArr(s) = Chr(Asc(letter) + 26 + shift)
59                     else if((Asc(letter) = 90) and (shift > 0)) then
60                         solvedChrArr(s) = Chr(Asc(letter) - 26 + shift)
61                     else
62                         solvedChrArr(s) = Chr(Asc(letter) + shift)
63                     End if
64                 Else
65                     solvedChrArr(s) = letter
66                 End if
67             Next
68             solvedStr = New String(solvedChrArr)
69             DIM tempNum as integer = maxShift-j-1
70             Console.WriteLine(vbTab & "Caesar " + tempNum.toString() + ": " + solvedStr)
71         Next
72         solve = solvedStr
73     End Function
74
75     Sub Main()
76         ' your code goes here
77         Console.WriteLine("Caesar Cipher" & vbCrLf)
78         DIM testStr1 as string = "IBM"
79         DIM testStr2 as string = "Hello World"
80         DIM testStr3 as string = "This is a test"
81         DIM testStr4 as string = "Hi my name is Robbie"
82         DIM testStr5 as string = "WandaVision"
83         DIM testStr6 as string = "Abed"
84         DIM testStr7 as string = "The Mandalorian"
85         DIM testStr8 as string = "Wow I learned BASIC I think"

```

```

86
87     DIM shiftAmount as integer = -1
88     DIM maxShiftAmount as integer = 26
89
90     DIM testStrArr = New String() {testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7}
91
92     DIM i as integer
93     for i = 0 to (testStrArr.length - 1)
94         'Console.WriteLine(testStrArr(i))
95         DIM tempStr as string = UCase(testStrArr(i))
96         Console.WriteLine("Original String: " + tempStr)
97
98         DIM encryptedTempStr as string = encrypt(tempStr, shiftAmount)
99         Console.WriteLine("Encrypted String: " + encryptedTempStr)
100
101         DIM decryptedTempStr as string = decrypt(encryptedTempStr, shiftAmount)
102         Console.WriteLine("Decrypted String: " + decryptedTempStr)
103
104         Console.WriteLine("Solve: ")
105         solve(encryptedTempStr, maxShiftAmount)
106         Console.WriteLine()
107     next
108 End Sub
109 End Module

```

### 3.4 OUTPUT

```

1 Caesar Cipher
2
3 Original String: IBM
4 Encrypted String: HAL
5 Decrypted String: IBM
6 Solve:
7     Caesar 26: HAL
8     Caesar 25: GZK
9     Caesar 24: FYJ
10    Caesar 23: EXI
11    Caesar 22: DWH
12    Caesar 21: CVG
13    Caesar 20: BUF
14    Caesar 19: ATE
15    Caesar 18: ZSD
16    Caesar 17: YRC
17    Caesar 16: XQB
18    Caesar 15: WPA
19    Caesar 14: VOZ
20    Caesar 13: UNY
21    Caesar 12: TMX
22    Caesar 11: SLW
23    Caesar 10: RKV
24    Caesar 9: QJU
25    Caesar 8: PIT
26    Caesar 7: OHS
27    Caesar 6: NGR
28    Caesar 5: MFQ
29    Caesar 4: LEP
30    Caesar 3: KDO
31    Caesar 2: JCN
32    Caesar 1: IBM
33    Caesar 0: HAL
34
35 Original String: HELLO WORLD
36 Encrypted String: GDKKN VNQKC

```

```

37 Decrypted String: HELLO WORLD
38 Solve:
39     Caesar 26: GDKKN VNQKC
40     Caesar 25: FCJJM UMPJB
41     Caesar 24: EBII L TLOIA
42     Caesar 23: DAHHK SKNHZ
43     Caesar 22: CZGGJ RJMGY
44     Caesar 21: BYFFI QILFX
45     Caesar 20: AXEEH PHKEW
46     Caesar 19: ZWDDG OGJDV
47     Caesar 18: YVCCF NFICU
48     Caesar 17: XUBBE MEHBT
49     Caesar 16: WTAAD LDGAS
50     Caesar 15: VSZZC KCFZR
51     Caesar 14: URY YB JBEYQ
52     Caesar 13: TQXXA IADXP
53     Caesar 12: SPWWZ HZCWO
54     Caesar 11: ROVVY GYBVN
55     Caesar 10: QNUUX FXAUM
56     Caesar 9: PMTTW EWZTL
57     Caesar 8: OLSSV DVYSK
58     Caesar 7: NKRRU CUXRJ
59     Caesar 6: MJQQT BTWQI
60     Caesar 5: LIPPS ASVPH
61     Caesar 4: KHOOR ZRUOG
62     Caesar 3: JGNNQ YQTNF
63     Caesar 2: IFMMP XPSME
64     Caesar 1: HELLO WORLD
65     Caesar 0: GDKKN VNQKC
66
67 Original String: THIS IS A TEST
68 Encrypted String: SGHR HR Z SDRS
69 Decrypted String: THIS IS A TEST
70 Solve:
71     Caesar 26: SGHR HR Z SDRS
72     Caesar 25: RFGQ GQ Y RCQR
73     Caesar 24: QEFP FP X QBPQ
74     Caesar 23: PDEO EO W PAOP
75     Caesar 22: OCDN DN V OZNO
76     Caesar 21: NBCM CM U NYMN
77     Caesar 20: MABL BL T MXLM
78     Caesar 19: LZAK AK S LWKL
79     Caesar 18: KYZJ ZJ R KVJK
80     Caesar 17: JXYI YI Q JUIJ
81     Caesar 16: IWXH XH P ITHI
82     Caesar 15: HVWG WG O HSGH
83     Caesar 14: GUVF VF N GRFG
84     Caesar 13: FTUE UE M FQEF
85     Caesar 12: ESTD TD L EPDE
86     Caesar 11: DRSC SC K DOCD
87     Caesar 10: CQRB RB J CNBC
88     Caesar 9: BPQA QA I BMAB
89     Caesar 8: AOPZ PZ H ALZA
90     Caesar 7: ZNOY OY G ZKYZ
91     Caesar 6: YMN X NX F YJXY
92     Caesar 5: XLMW MW E XIWX
93     Caesar 4: WKLV LV D WHVW
94     Caesar 3: VJKU KU C VGUV
95     Caesar 2: UIJT JT B UFTU
96     Caesar 1: THIS IS A TEST
97     Caesar 0: SGHR HR Z SDRS
98
99 Original String: HI MY NAME IS ROBBIE
100 Encrypted String: GH LX MZLD HR QNA AHD
101 Decrypted String: HI MY NAME IS ROBBIE

```

```

102 Solve:
103 Caesar 26: GH LX MZLD HR QNAAHD
104 Caesar 25: FG KW LYKC GQ PMZZGC
105 Caesar 24: EF JV KXJB FP OLYYFB
106 Caesar 23: DE IU JWIA EO NKXXEA
107 Caesar 22: CD HT IVHZ DN MJWWDZ
108 Caesar 21: BC GS HUGY CM LIVVCY
109 Caesar 20: AB FR GTFX BL KHUUBX
110 Caesar 19: ZA EQ FSEW AK JGTTAW
111 Caesar 18: YZ DP ERDV ZJ IFSSZV
112 Caesar 17: XY CO DQCU YI HERRYU
113 Caesar 16: WX BN CPBT XH GDQQXT
114 Caesar 15: VW AM BOAS WG FCPPWS
115 Caesar 14: UV ZL ANZR VF EBOOVR
116 Caesar 13: TU YK ZMYQ UE DANNUQ
117 Caesar 12: ST XJ YLXP TD CZMMTP
118 Caesar 11: RS WI XKWO SC BYLLSO
119 Caesar 10: QR VH WJVN RB AXKKRN
120 Caesar 9: PQ UG VIUM QA ZWJJQM
121 Caesar 8: OP TF UHTL PZ YVIPL
122 Caesar 7: NO SE TGSK OY XUHHOK
123 Caesar 6: MN RD SFRJ NX WTGGNJ
124 Caesar 5: LM QC REQI MW VSFFMI
125 Caesar 4: KL PB QDPH LV UREELH
126 Caesar 3: JK OA PCOG KU TQDDKG
127 Caesar 2: IJ NZ OBNF JT SPCCJF
128 Caesar 1: HI MY NAME IS ROBBIE
129 Caesar 0: GH LX MZLD HR QNAAHD

```

```

131 Original String: WANDAVISION
132 Encrypted String: VZMCZUHRHNM
133 Decrypted String: WANDAVISION
134 Solve:

```

```

135 Caesar 26: VZMCZUHRHNM
136 Caesar 25: UYLBYTGQGML
137 Caesar 24: TXKAXSFPFLK
138 Caesar 23: SWJZWREOEKJ
139 Caesar 22: RVIYVQDNDJI
140 Caesar 21: QUHXUPCMCIH
141 Caesar 20: PTGWTOLBHG
142 Caesar 19: OSFVSNAKAGF
143 Caesar 18: NREURMZJZFE
144 Caesar 17: MQDTQLYIYED
145 Caesar 16: LPCSPKXHXDC
146 Caesar 15: KOBROJWGWCB
147 Caesar 14: JNAQNIVFVBA
148 Caesar 13: IMZPMHUEVAZ
149 Caesar 12: HLYOLGTDZTY
150 Caesar 11: GKXNKFSCSYX
151 Caesar 10: FJWMJERBRXW
152 Caesar 9: EIVLIDQAQWV
153 Caesar 8: DHUKHCPZPVU
154 Caesar 7: CGTJGBOYOUT
155 Caesar 6: BFSIFANXNTS
156 Caesar 5: AERHEZMWMSR
157 Caesar 4: ZDQGDYLVLRQ
158 Caesar 3: YCPFCXKUKQP
159 Caesar 2: XBOEBWJTJPO
160 Caesar 1: WANDAVISION
161 Caesar 0: VZMCZUHRHNM

```

```

163 Original String: ABED
164 Encrypted String: ZADC
165 Decrypted String: ABED
166 Solve:

```

```

167 Caesar 26: ZADC
168 Caesar 25: YZCB
169 Caesar 24: XYBA
170 Caesar 23: WXAZ
171 Caesar 22: VWZY
172 Caesar 21: UVYX
173 Caesar 20: TUXW
174 Caesar 19: STWV
175 Caesar 18: RSVU
176 Caesar 17: QRUT
177 Caesar 16: PQTS
178 Caesar 15: OPSR
179 Caesar 14: NORQ
180 Caesar 13: MNQP
181 Caesar 12: LMPO
182 Caesar 11: KLON
183 Caesar 10: JKNM
184 Caesar 9: IJML
185 Caesar 8: HILK
186 Caesar 7: GHKJ
187 Caesar 6: FGJI
188 Caesar 5: EFIH
189 Caesar 4: DEHG
190 Caesar 3: CDGF
191 Caesar 2: BCFE
192 Caesar 1: ABED
193 Caesar 0: ZADC
194
195 Original String: THE MANDALORIAN
196 Encrypted String: SGD LZMCZKNQHZM
197 Decrypted String: THE MANDALORIAN
198 Solve:
199 Caesar 26: SGD LZMCZKNQHZM
200 Caesar 25: RFC KYLBYJMPGYL
201 Caesar 24: QEB JXXAXILOFXK
202 Caesar 23: PDA IWJZWHKNEWJ
203 Caesar 22: OCZ HVIYVGJMDVI
204 Caesar 21: NBY GUHXUFILCUH
205 Caesar 20: MAX FTGWTEHKBTG
206 Caesar 19: LZW ESFVSDGJASF
207 Caesar 18: KYV DREURCFIZRE
208 Caesar 17: JXU CQDTQBEHYQD
209 Caesar 16: IWT BPCSPADGXPC
210 Caesar 15: HVS AOBROZCFWOB
211 Caesar 14: GUR ZNAQNYBEVNA
212 Caesar 13: FTQ YMZPMXADUMZ
213 Caesar 12: ESP XLYOLWZCTLY
214 Caesar 11: DRO WKXNKVYBSKX
215 Caesar 10: CQN VJWMJUXARJW
216 Caesar 9: BPM UIVLITWZQIV
217 Caesar 8: AOL THUKHSVYPHU
218 Caesar 7: ZNK SGTJGRUXOGT
219 Caesar 6: YMJ RFSIFQTWNFS
220 Caesar 5: XLI QERHEPSVMER
221 Caesar 4: WKH PDQGDORULDQ
222 Caesar 3: VJG OCPFCNQTKCP
223 Caesar 2: UIF NBOEBMPSJBO
224 Caesar 1: THE MANDALORIAN
225 Caesar 0: SGD LZMCZKNQHZM
226
227 Original String: WOW I LEARNED BASIC I THINK
228 Encrypted String: VNV H KDZQMDC AZRHB H SGHMJ
229 Decrypted String: WOW I LEARNED BASIC I THINK
230 Solve:
231 Caesar 26: VNV H KDZQMDC AZRHB H SGHMJ

```

```

232 Caesar 25: UMU G JCYPLCB ZYQGA G RFGLI
233 Caesar 24: TLT F IBXOKBA YXPFZ F QEFKH
234 Caesar 23: SKS E HAWNJAZ XWOEY E PDEJG
235 Caesar 22: RJR D GZVMIZY WVNDX D OCDIF
236 Caesar 21: QIQ C FYULHYX VUMCW C NBCHE
237 Caesar 20: PHP B EXTGXW UTLBV B MABGD
238 Caesar 19: OGO A DWSJFWV TSKAU A LZAFK
239 Caesar 18: NFN Z CVRIEVU SRJZT Z KYZEB
240 Caesar 17: MEM Y BUQHDUT RQIYS Y JXYDA
241 Caesar 16: LDL X ATPGCTS QPHXR X IWXCZ
242 Caesar 15: KCK W ZSOFBSR POGWQ W HVWBY
243 Caesar 14: JBJ V YRNEARQ ONFVP V GUVAX
244 Caesar 13: IAI U XQMDZQP NMEUO U FTUZW
245 Caesar 12: HZH T WPLCYPO MLDTN T ESTYV
246 Caesar 11: GYG S VOKBXON LKCSM S DRSXU
247 Caesar 10: FFX R UNJAWNM KJBRL R CQRWT
248 Caesar 9: EWE Q TMIZVML JIAQK Q BPQVS
249 Caesar 8: DVD P SLHYULK IHZPJ P AOPUR
250 Caesar 7: CUC O RKGXTKJ HGYOI O ZNOTQ
251 Caesar 6: BTB N QJFWSJI GFXNH N YMNSP
252 Caesar 5: ASA M PIEVRIH FEWMG M XLMRO
253 Caesar 4: ZRZ L OHDUQHG EDVLF L WKLQN
254 Caesar 3: YQY K NGCTPGF DCUKE K VJKPM
255 Caesar 2: XPX J MFBSOFE CBTJD J UIJOL
256 Caesar 1: WOW I LEARNED BASIC I THINK
257 Caesar 0: VNV H KDZQMDC AZRHB H SGHMJ

```

## 4 PASCAL

### 4.1 CONSULTING LOG

Expected hours needed: 8

<i>Date</i>	<i>Hours Spent</i>	<i>Tasks / Accomplishments / Issues / Thoughts</i>
03/15/2021	1	Learned syntax and started it based off of what I had done for Scala.
03/17/2021	3	I wrote out the encrypt function, however I am getting some weird errors that I can't discern. I finally got some of them to go away when I added parenthesis and found out how to properly type cast some variables.
03/18/2021	3	Spent a lot of time debugging the encrypt function and finally got it to work. It is still buggy though. I also wrote some of the decrypt function based off of the encrypt, however, both are buggy with certain letters not moving in the right direction. This is because the if statements are not working how I would like them to. "else if" and "else" syntax are giving me a really hard time.
03/19/2021	2.5	After a couple of hours of trying to figure it out, I finally figured out the BEGIN and END for the nested if statements that contained else and else if's with help from your email. The syntax of it was what threw me for a loop and I could not wrap my head around it for a while.

### 4.2 COMMENTARY

Pascal was not at all bad to work with and it had some great concept things in the context and in functions. However, I thought some of the syntax was confusing especially with If, else if, and else statements. Everything else was fine, but ending these statements was confusing and did not work how I thought they did or should



have. The documentation for this online mainly was not helpful until I emailed you which cleared up how to group/end the statements.

### 4.3 SOURCE CODE

```

1 program ideone;
2 (*define encrypt function*)
3 function encrypt(testStr: string; shift: integer): string;
4 var
5     encryptedChrArr : array of char;
6     letter : char;
7     encryptedStr : string;
8     x, e : integer;
9 BEGIN
10     encryptedStr := '';
11     SetLength(encryptedChrArr, length(testStr));
12
13     for x:=0 to length(testStr)-1 do
14     BEGIN
15         encryptedChrArr[x] := testStr[x+1]; (*Add each character of string to a char array*)
16         (*writeln(encryptedChrArr[x]);*)
17     END; {for x}
18
19     for e:=0 to length(encryptedChrArr)-1 do
20     BEGIN
21         letter := encryptedChrArr[e];
22         (*writeln(letter);*) (*check letters*)
23         if((Integer(letter) >= 65) and (Integer(letter) <= 90)) then BEGIN(*A to Z*)
24             if((Integer(letter) = 65) and (shift < 0)) then BEGIN(*If A and shift is negative,
25                 encryptedChrArr[e] := Chr(Integer(letter) + 26 + shift);
26             END else if((Integer(letter) = 90) and (shift > 0)) then BEGIN(*If Z, and shift is posi
27                 encryptedChrArr[e] := Chr(Integer(letter) - 26 + shift);
28             END else BEGIN(*Perform a normal shift*)
29                 encryptedChrArr[e] := Chr(Integer(letter) + shift);
30             END;
31         END else BEGIN (*Other Characters*)
32             encryptedChrArr[e] := letter;
33         END; {if}
34         (*turn character array into new string*)
35         encryptedStr := encryptedStr + encryptedChrArr[e];
36     END; {for e}
37     encrypt := encryptedStr;
38 END; {function encrypt}
39
40 (*define decrypt function*)
41 function decrypt(testStr: string; shift: integer): string;
42 var
43     decryptedChrArr : array of char;
44     letter : char;
45     decryptedStr : string;
46     y, d : integer;
47
48 BEGIN
49     decryptedStr := '';
50     SetLength(decryptedChrArr, length(testStr));
51
52     for y:=0 to length(testStr)-1 do
53     BEGIN
54         decryptedChrArr[y] := testStr[y+1]; (*Add each character of string to a char array*)
55         (*writeln(decryptedChrArr[x]);*)
56     END; {for y}
57
58     for d:=0 to length(decryptedChrArr)-1 do
59     BEGIN

```

```

60         letter := decryptedChrArr[d];
61         (*writeln(letter);*) (*check letters*)
62         if((Integer(letter) >= 65) and (Integer(letter) <= 90)) then BEGIN(*A to Z*)
63             if((Integer(letter) = 65) and (-shift < 0)) then BEGIN(*If A and shift is negative,
64                 decryptedChrArr[d] := Chr(Integer(letter) + 26 - shift);
65             END else if((Integer(letter) = 90) and (-shift > 0)) then BEGIN(*If Z, and shift is pos
66                 decryptedChrArr[d] := Chr(Integer(letter) - 26 - shift);
67             END else BEGIN(*Perform a normal shift*)
68                 decryptedChrArr[d] := Chr(Integer(letter) - shift);
69             END;
70         END else BEGIN (*Other Characters*)
71             decryptedChrArr[d] := letter;
72         END; {if}
73         (*turn character array into new string*)
74         decryptedStr := decryptedStr + decryptedChrArr[d];
75     END; {for d}
76     decrypt := decryptedStr;
77 END; {function decrpyt}
78
79
80 (*define solve function*)
81 function solve(testStr: string; maxShift: integer) : string;
82 var
83     solvedChrArr : array of char;
84     letter : char;
85     solvedStr : string;
86     shift : integer;
87     tempNum : integer;
88     n, j, s : integer;
89
90 BEGIN
91     writeln(#9'Caesar 26: ' + testStr);
92     solvedStr := '';
93     shift := -1;
94     SetLength(solvedChrArr, length(testStr));
95
96     for n:=0 to length(testStr)-1 do
97     BEGIN
98         solvedChrArr[n] := testStr[n+1]; (*Add each character of string to a char array*)
99         (*writeln(decryptedChrArr[x]);*)
100     END; {for n}
101
102     for j:=0 to maxShift-1 do
103     BEGIN
104         solvedStr := '';
105         for s:=0 to length(solvedChrArr)-1 do
106         BEGIN
107             letter := solvedChrArr[s];
108             (*writeln(letter);*) (*check letters*)
109             if((Integer(letter) >= 65) and (Integer(letter) <= 90)) then BEGIN(*A to Z*)
110                 if((Integer(letter) = 65) and (shift < 0)) then BEGIN(*If A and shift is ne
111                     solvedChrArr[s] := Chr(Integer(letter) + 26 + shift);
112                 END else if((Integer(letter) = 90) and (shift > 0)) then BEGIN(*If Z, and shift
113                     solvedChrArr[s] := Chr(Integer(letter) - 26 + shift);
114                 END else BEGIN(*Perform a normal shift*)
115                     solvedChrArr[s] := Chr(Integer(letter) + shift);
116                 END;
117             END else BEGIN (*Other Characters*)
118                 solvedChrArr[s] := letter;
119             END; {if}
120             (*turn character array into new string*)
121             solvedStr := solvedStr + solvedChrArr[s];
122         END; {for s}
123         tempNum := maxShift-j-1;
124         writeln(#9'Caesar ', tempNum, ': ' + solvedStr);

```

```

125         END; {for j}
126         solve := solvedStr;
127 END; {function solve}
128
129 var
130     testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7, testStr8 : string;
131     shiftAmount, maxShiftAmount : integer;
132     testStrArr : array[0..7] of string;
133
134     tempStr, encryptedTempStr, decryptedTempStr, solvedTempStr : string;
135     i : integer;
136
137 BEGIN
138     writeln('Caesar Cipher');
139     writeln();
140     testStr1 := 'IBM';
141     testStr2 := 'Hello World';
142     testStr3 := 'This is a test';
143     testStr4 := 'Hi my name is Robbie';
144     testStr5 := 'WandaVision';
145     testStr6 := 'Abed';
146     testStr7 := 'The Mandalorian';
147     testStr8 := 'Wow I learned Pascal I think';
148
149     shiftAmount := -1;
150     maxShiftAmount := 26;
151
152     testStrArr[0] := testStr1; testStrArr[1] := testStr2;
153     testStrArr[2] := testStr3; testStrArr[3] := testStr4;
154     testStrArr[4] := testStr5; testStrArr[5] := testStr6;
155     testStrArr[6] := testStr7; testStrArr[7] := testStr8;
156
157     for i:=0 to length(testStrArr)-1 do
158     BEGIN
159         tempStr := UpCase(testStrArr[i]); (*Change all letters to uppercase*)
160         writeln('Original String: ' + tempStr);
161
162         encryptedTempStr := encrypt(tempStr, shiftAmount);
163         writeln('Encrypted String: ' + encryptedTempStr);
164
165         decryptedTempStr := decrypt(encryptedTempStr, shiftAmount);
166         writeln('Decrypted String: ' + decryptedTempStr);
167
168         writeln('Solve: ');
169         solvedTempStr := solve(encryptedTempStr, maxShiftAmount);
170         writeln();
171     END; {for i}
172 END. {program ideone}

```

## 4.4 OUTPUT

```

1 Caesar Cipher
2
3 Original String: IBM
4 Encrypted String: HAL
5 Decrypted String: IBM
6 Solve:
7     Caesar 26: HAL
8     Caesar 25: GZK
9     Caesar 24: FYJ
10    Caesar 23: EXI
11    Caesar 22: DWH
12    Caesar 21: CVG

```

```

13 Caesar 20: BUF
14 Caesar 19: ATE
15 Caesar 18: ZSD
16 Caesar 17: YRC
17 Caesar 16: XQB
18 Caesar 15: WPA
19 Caesar 14: VOZ
20 Caesar 13: UNY
21 Caesar 12: TMX
22 Caesar 11: SLW
23 Caesar 10: RKV
24 Caesar 9: QJU
25 Caesar 8: PIT
26 Caesar 7: OHS
27 Caesar 6: NGR
28 Caesar 5: MFQ
29 Caesar 4: LEP
30 Caesar 3: KDO
31 Caesar 2: JCN
32 Caesar 1: IBM
33 Caesar 0: HAL
34
35 Original String: HELLO WORLD
36 Encrypted String: GDKKN VNQKC
37 Decrypted String: HELLO WORLD
38 Solve:
39 Caesar 26: GDKKN VNQKC
40 Caesar 25: FCJJM UMPJB
41 Caesar 24: EBII L TLOIA
42 Caesar 23: DAHHK SKNHZ
43 Caesar 22: CZGGJ RJMGY
44 Caesar 21: BYFFI QILFX
45 Caesar 20: AXEEH PHKEW
46 Caesar 19: ZWDDG OGJDV
47 Caesar 18: YVCCF NFICU
48 Caesar 17: XUBBE MEHBT
49 Caesar 16: WTAAD LDGAS
50 Caesar 15: VSZZC KCFZR
51 Caesar 14: URY YB JBEYQ
52 Caesar 13: TQXXA IADXP
53 Caesar 12: SPWWZ HZCWO
54 Caesar 11: ROVVY GYBVN
55 Caesar 10: QNUUX FXAUM
56 Caesar 9: PMTTW EWZTL
57 Caesar 8: OLSSV DVYSK
58 Caesar 7: NKRRU CUXRJ
59 Caesar 6: MJQQT BTWQI
60 Caesar 5: LIPPS ASVPH
61 Caesar 4: KHOOR ZRUOG
62 Caesar 3: JGNNQ YQTNF
63 Caesar 2: IFMMP XPSME
64 Caesar 1: HELLO WORLD
65 Caesar 0: GDKKN VNQKC
66
67 Original String: THIS IS A TEST
68 Encrypted String: SGHR HR Z SDRS
69 Decrypted String: THIS IS A TEST
70 Solve:
71 Caesar 26: SGHR HR Z SDRS
72 Caesar 25: RFGQ GQ Y RCQR
73 Caesar 24: QEFP FP X QBPQ
74 Caesar 23: PDEO EO W PAOP
75 Caesar 22: OCDN DN V OZNO
76 Caesar 21: NBCM CM U NYMN
77 Caesar 20: MABL BL T MXLM

```

```

78 Caesar 19: LZAK AK S LWKL
79 Caesar 18: KYZJ ZJ R KVJK
80 Caesar 17: JXYI YI Q JUIJ
81 Caesar 16: IWXH XH P ITHI
82 Caesar 15: HVWG WG O HSGH
83 Caesar 14: GUVF VF N GRFG
84 Caesar 13: FTUE UE M FQEF
85 Caesar 12: ESTD TD L EPDE
86 Caesar 11: DRSC SC K DOCD
87 Caesar 10: CQRB RB J CNBC
88 Caesar 9: BPQA QA I BMAB
89 Caesar 8: AOPZ PZ H ALZA
90 Caesar 7: ZNOY OY G ZKYZ
91 Caesar 6: YMNX NX F YJXY
92 Caesar 5: XLMW MW E XIWX
93 Caesar 4: WKLV LV D WHVW
94 Caesar 3: VJKU KU C VGUU
95 Caesar 2: UIJT JT B UFTU
96 Caesar 1: THIS IS A TEST
97 Caesar 0: SGHR HR Z SDRS
98
99 Original String: HI MY NAME IS ROBBIE
100 Encrypted String: GH LX MZLD HR QNAAHD
101 Decrypted String: HI MY NAME IS ROBBIE
102 Solve:
103 Caesar 26: GH LX MZLD HR QNAAHD
104 Caesar 25: FG KW LYKC GQ PMZZGC
105 Caesar 24: EF JV KXJB FP OLYYFB
106 Caesar 23: DE IU JWIA EO NKXXEA
107 Caesar 22: CD HT IVHZ DN MJWWDZ
108 Caesar 21: BC GS HUGY CM LIVVCY
109 Caesar 20: AB FR GTFX BL KHUUBX
110 Caesar 19: ZA EQ FSEW AK JGTTAW
111 Caesar 18: YZ DP ERDV ZJ IFSSZV
112 Caesar 17: XY CO DQCU YI HERRYU
113 Caesar 16: WX BN CPBT XH GDQQXT
114 Caesar 15: VW AM BOAS WG FCPPWS
115 Caesar 14: UV ZL ANZR VF EBOOVR
116 Caesar 13: TU YK ZMYQ UE DANNUQ
117 Caesar 12: ST XJ YLXP TD CZMMTP
118 Caesar 11: RS WI XKWO SC BYLLSO
119 Caesar 10: QR VH WJVN RB AXKKRN
120 Caesar 9: PQ UG VIUM QA ZWJJQM
121 Caesar 8: OP TF UHTL PZ YVIPL
122 Caesar 7: NO SE TGSK OY XUHHOK
123 Caesar 6: MN RD SFRJ NX WTGGNJ
124 Caesar 5: LM QC REQI MW VSFFMI
125 Caesar 4: KL PB QDPH LV UREELH
126 Caesar 3: JK OA PCOG KU TQDDKG
127 Caesar 2: IJ NZ OBNF JT SPCCJF
128 Caesar 1: HI MY NAME IS ROBBIE
129 Caesar 0: GH LX MZLD HR QNAAHD
130
131 Original String: WANDAVISION
132 Encrypted String: VZMCZUHRHNM
133 Decrypted String: WANDAVISION
134 Solve:
135 Caesar 26: VZMCZUHRHNM
136 Caesar 25: UYLBYTGQGML
137 Caesar 24: TXKAXSFPFLK
138 Caesar 23: SWJZWREOEKJ
139 Caesar 22: RVIYVQDNDJI
140 Caesar 21: QUHXUPCMCIH
141 Caesar 20: PTGWTOBLBHG
142 Caesar 19: OSFVSNKAGF

```

```

143 Caesar 18: NREURMZJZFE
144 Caesar 17: MQDTQLYIYED
145 Caesar 16: LPCSPKXHXDC
146 Caesar 15: KOBROJWGWCB
147 Caesar 14: JNAQNIVFVBA
148 Caesar 13: IMZPMHUEUAZ
149 Caesar 12: HLYOLGTDITY
150 Caesar 11: GKXNKFSCSYX
151 Caesar 10: FJWMJERBRXW
152 Caesar 9: EIVLIDQAQWV
153 Caesar 8: DHUKHCPZPVU
154 Caesar 7: CGTJGBOYOUT
155 Caesar 6: BFSIFANXNTS
156 Caesar 5: AERHEZMWMMSR
157 Caesar 4: ZDQGDYLVLRQ
158 Caesar 3: YCPFCXKUKQP
159 Caesar 2: XBOEBWJTJPO
160 Caesar 1: WANDAVISION
161 Caesar 0: VZMCZUHRHNM

```

```

162
163 Original String: ABED
164 Encrypted String: ZADC
165 Decrypted String: ABED
166 Solve:

```

```

167 Caesar 26: ZADC
168 Caesar 25: YZCB
169 Caesar 24: XYBA
170 Caesar 23: WXAZ
171 Caesar 22: VWZY
172 Caesar 21: UVYX
173 Caesar 20: TUXW
174 Caesar 19: STWV
175 Caesar 18: RSVU
176 Caesar 17: QRUT
177 Caesar 16: PQTS
178 Caesar 15: OPSR
179 Caesar 14: NORQ
180 Caesar 13: MNQP
181 Caesar 12: LMPO
182 Caesar 11: KLON
183 Caesar 10: JKNM
184 Caesar 9: IJML
185 Caesar 8: HILK
186 Caesar 7: GHKJ
187 Caesar 6: FGJI
188 Caesar 5: EFIH
189 Caesar 4: DEHG
190 Caesar 3: CDGF
191 Caesar 2: BCFE
192 Caesar 1: ABED
193 Caesar 0: ZADC

```

```

194
195 Original String: THE MANDALORIAN
196 Encrypted String: SGD LZMCZKNQHZM
197 Decrypted String: THE MANDALORIAN
198 Solve:

```

```

199 Caesar 26: SGD LZMCZKNQHZM
200 Caesar 25: RFC KYLBYPJGYYL
201 Caesar 24: QEB JXXAXILOFXK
202 Caesar 23: PDA IWJZWHKNEWJ
203 Caesar 22: OCZ HVIYVGJMDVI
204 Caesar 21: NBY GUHXUFILCUH
205 Caesar 20: MAX FTGWTEHKBTG
206 Caesar 19: LZW ESFVSDGJASF
207 Caesar 18: KYV DREURCFIZRE

```

```

208 Caesar 17: JXU CQDTQBEHYQD
209 Caesar 16: IWT BPCSPADGXPC
210 Caesar 15: HVS AOBROZCFWOB
211 Caesar 14: GUR ZNAQNYBEVNA
212 Caesar 13: FTQ YMZPMXADUMZ
213 Caesar 12: ESP XLYOLWZCTLY
214 Caesar 11: DRO WKXNKVYBSKX
215 Caesar 10: CQN VJWMJUXARJW
216 Caesar 9: BPM UIVLITWZQIV
217 Caesar 8: AOL THUKHSVYPHU
218 Caesar 7: ZNK SGTJGRUXOGT
219 Caesar 6: YMJ RFSIFQTNFS
220 Caesar 5: XLI QERHEPSVMER
221 Caesar 4: WKH PDQGDRULDQ
222 Caesar 3: VJG OCPFCNQTKCP
223 Caesar 2: UIF NBOEBMPSJBO
224 Caesar 1: THE MANDALORIAN
225 Caesar 0: SGD LZMCZKNQHZM
226
227 Original String: WOW I LEARNED PASCAL I THINK
228 Encrypted String: VNV H KDZQMDC OZRBZK H SGHMJ
229 Decrypted String: WOW I LEARNED PASCAL I THINK
230 Solve:
231 Caesar 26: VNV H KDZQMDC OZRBZK H SGHMJ
232 Caesar 25: UMU G JCYPLCB NYQAYJ G RFGLI
233 Caesar 24: TLT F IBXOKBA MXPZXI F QEFKH
234 Caesar 23: SKS E HAWNJAZ LWOYWH E PDEJG
235 Caesar 22: RJR D GZVMIZY KVNXYG D OCDIF
236 Caesar 21: QIQ C FYULHYX JUMWUF C NBCHE
237 Caesar 20: PHP B EXTGXW ITLVTE B MABGD
238 Caesar 19: OGO A DWSJFWV HSKUSD A LZAFK
239 Caesar 18: NFN Z CVRIEVU GRJTRC Z KYZEB
240 Caesar 17: MEM Y BUQHDUT FQISQB Y JXYDA
241 Caesar 16: LDL X ATPGCTS EPHRPA X IWXCZ
242 Caesar 15: KCK W ZSOFBSR DOGQOZ W HVWBY
243 Caesar 14: JBJ V YRNEARQ CNFPNY V GUVAX
244 Caesar 13: IAI U XQMDZQP BMEOMX U FTUZW
245 Caesar 12: HZH T WPLCYPO ALDNLW T ESTYV
246 Caesar 11: GYG S VOKBXON ZKCMKV S DRSXU
247 Caesar 10: FXF R UNJAWNM YJBLJU R CQRWT
248 Caesar 9: EWE Q TMIZVML XIAKIT Q BPQVS
249 Caesar 8: DVD P SLHYULK WHZJHS P AOPUR
250 Caesar 7: CUC O RKGXTKJ VGYIGR O ZNOTQ
251 Caesar 6: BTB N QJFWSJI UFXHFQ N YMNPS
252 Caesar 5: ASA M PIEVRIH TEWGEP M XLMRO
253 Caesar 4: ZRZ L OHDUQHG SDVFDO L WKLQN
254 Caesar 3: YQY K NGCTPGF RCUECN K VJKPM
255 Caesar 2: XPX J MFBSOFE QBTDBM J UIJOL
256 Caesar 1: WOW I LEARNED PASCAL I THINK
257 Caesar 0: VNV H KDZQMDC OZRBZK H SGHMJ

```

## 5 SCALA

### 5.1 CONSULTING LOG

Expected hours needed: 6

<i>Date</i>	<i>Hours Spent</i>	<i>Tasks / Accomplishments / Issues / Thoughts</i>
03/10/2021	0.5	Time mostly spent learning the basics and learning the syntax.
03/11/2021	1	I mostly just spent a lot of time debugging.
03/12/2021	1	a lot more debugging with some more research and learning. My biggest issue between yesterday and today has just been with replacing the characters in the character array with a new value. So far I have only been able to get spaces to print out rather than what I actually thought it would. Mostly syntax errors but some conceptually errors on what is going on.
03/14	3	I FIGURED IT OUT. I did not work on it for terribly long before I finally got it to work today. Once I got the encryption done, decryption took significantly less time and the solve function took even less time with some room for me to make it look a little nicer. Getting that syntax and getting myself to write code that will do what I know I conceptually need to do was a big relief for me and it made my "late" night all worth it. I also ended up switching back to ideone.com because codingground was not working/ connecting to their servers.

## 5.2 COMMENTARY

Scala felt very close to Java and other languages that I'm more used to, which I know was kind of the point. That made it easier for me to get that baseline of code down and figure out the logistics of writing the program without as many syntactical hiccups. This did not take me too long to write, or at least not as many headaches like Fortran and COBOL. Scala was enjoyable and I can see myself taking ideas from it and using it again in the future.

## 5.3 SOURCE CODE

```

1 object Main {
2     def main(args: Array[String]) {
3         println("Caesar Cipher\n")
4         val testStr1 = "IBM"
5         val testStr2 = "Hello World"
6         val testStr3 = "This is a test"
7         val testStr4 = "Hi my name is Robbie"
8         val testStr5 = "WandaVision"
9         val testStr6 = "Abed"
10        val testStr7 = "The Mandalorian"
11        val testStr8 = "Wow I learned Scala I think"
12
13        val shiftAmount = -1
14        val maxShiftAmount = 26
15
16        var testStrArr = Array(testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7, testStr8)
17
18        for(i <- testStrArr) {
19            var tempStr = i.toUpperCase()
20            println("Original String: " + tempStr)
21
22            var encryptedTempStr = encrypt(tempStr, shiftAmount)
23            println("Encrypted String: " + encryptedTempStr)
24
25            var decryptedTempStr = decrypt(encryptedTempStr, shiftAmount)
26            println("Decrypted String: " + decryptedTempStr)
27
28            println("Solve: ")
29            solve(encryptedTempStr, maxShiftAmount)

```



```

30         println()
31     }
32 }
33 def encrypt(testStr:String, shift:Int) : String /*Array[Char]*/= {
34     //var encryptedStr = ""
35     var encryptedChrArr = testStr.toCharArray()
36     var e = 0
37
38     for(e <- 0 until testStr.length()) {
39         var letter = encryptedChrArr(e) //testStr.charAt(e)
40         //println(letter)
41         if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
42             if(letter.toInt == 65 && shift < 0) { //If A and shift is negative, then loop around to Z
43                 encryptedChrArr(e) = (letter.toInt + 26 + shift).toChar
44             } else if(letter.toInt == 90 && shift > 0) { //If Z, and shift is positive, then loop around
45                 encryptedChrArr(e) = (letter.toInt - 26 + shift).toChar
46             } else { // Else perform a normal shift
47                 encryptedChrArr(e) = (letter.toInt + shift).toChar
48                 //encryptedStr.concat(letter.toString)
49                 //encryptedChrArr(e) = (testStr.charAt(e).toInt + shift)
50                 //println(letter)
51             }
52         }
53         else { //Other Characters
54             //Other Characters should not be changed in Caesar Cipher
55             encryptedChrArr(e) = letter
56             //println(letter)
57         }
58     }
59     var encryptedStr = encryptedChrArr.mkString("")
60     return encryptedStr
61 }
62 def decrypt(testStr:String, shift:Int) : String = {
63     var decryptedChrArr = testStr.toCharArray()
64     var d = 0
65
66     for(d <- 0 until testStr.length()) {
67         var letter = decryptedChrArr(d)
68         //shift = -shift//Switch sign of shift
69         //println(letter)
70         if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
71             if(letter.toInt == 65 && -shift < 0) { //If A and shift is negative, then loop around to Z
72                 decryptedChrArr(d) = (letter.toInt + 26 - shift).toChar
73             } else if(letter.toInt == 90 && -shift > 0) { //If Z, and shift is positive, then loop around
74                 decryptedChrArr(d) = (letter.toInt - 26 - shift).toChar
75             } else { // Else perform a normal shift
76                 decryptedChrArr(d) = (letter.toInt - shift).toChar
77                 //println(letter)
78             }
79         }
80         else { //Other Characters
81             //Other Characters should not be changed in Caesar Cipher
82             decryptedChrArr(d) = letter
83             //println(letter)
84         }
85     }
86     var decryptedStr = decryptedChrArr.mkString("")
87     return decryptedStr
88 }
89 def solve(testStr:String, maxShift:Int) {
90     var solvedChrArr = testStr.toCharArray()
91     val shift = -1
92     var j = 0
93     var s = 0

```

```

95     println("\tCaesar " + (26).toString + ": " + testStr)
96     for(j <- 0 until maxShift) {
97         for(s <- 0 until testStr.length()) {
98             var letter = solvedChrArr(s)
99             //println(letter)
100            if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
101                if(letter.toInt == 65 && shift < 0) { //If A and shift is negative, then loop around
102                    solvedChrArr(s) = (letter.toInt + 26 + shift).toChar
103                } else if(letter.toInt == 90 && shift > 0) { //If Z, and shift is positive, then loop around
104                    solvedChrArr(s) = (letter.toInt - 26 + shift).toChar
105                } else { // Else perform a normal shift
106                    solvedChrArr(s) = (letter.toInt + shift).toChar
107                    //println(letter)
108                }
109            }
110            else { //Other Characters
111                solvedChrArr(s) = letter
112                //println(letter)
113            }
114        }
115        var solvedStr = solvedChrArr.mkString("")
116        println("\tCaesar " + (maxShift-j-1).toString + ": " + solvedStr)
117    }
118 }
119 }

```

## 5.4 OUTPUT

```

1 Caesar Cipher
2
3 Original String: IBM
4 Encrypted String: HAL
5 Decrypted String: IBM
6 Solve:
7     Caesar 26: HAL
8     Caesar 25: GZK
9     Caesar 24: FYJ
10    Caesar 23: EXI
11    Caesar 22: DWH
12    Caesar 21: CVG
13    Caesar 20: BUF
14    Caesar 19: ATE
15    Caesar 18: ZSD
16    Caesar 17: YRC
17    Caesar 16: XQB
18    Caesar 15: WPA
19    Caesar 14: VOZ
20    Caesar 13: UNY
21    Caesar 12: TMX
22    Caesar 11: SLW
23    Caesar 10: RKV
24    Caesar 9: QJU
25    Caesar 8: PIT
26    Caesar 7: OHS
27    Caesar 6: NGR
28    Caesar 5: MFQ
29    Caesar 4: LEP
30    Caesar 3: KDO
31    Caesar 2: JCN
32    Caesar 1: IBM
33    Caesar 0: HAL
34
35 Original String: HELLO WORLD

```

```

36 Encrypted String: GDKKN VNQKC
37 Decrypted String: HELLO WORLD
38 Solve:
39 Caesar 26: GDKKN VNQKC
40 Caesar 25: FCJJM UMPJB
41 Caesar 24: EBIIL TLOIA
42 Caesar 23: DAHHK SKNHZ
43 Caesar 22: CZGGJ RJMGY
44 Caesar 21: BYFFI QILFX
45 Caesar 20: AXEEH PHKEW
46 Caesar 19: ZWDDG OGJDV
47 Caesar 18: YVCCF NFICU
48 Caesar 17: XUBBE MEHBT
49 Caesar 16: WTAAD LDGAS
50 Caesar 15: VSZZC KCFZR
51 Caesar 14: URYYB JBEYQ
52 Caesar 13: TQXXA IADXP
53 Caesar 12: SPWWZ HZCWO
54 Caesar 11: ROVVY GYBVN
55 Caesar 10: QNUUX FXAUM
56 Caesar 9: PMTTW EWZTL
57 Caesar 8: OLSSV DVYSK
58 Caesar 7: NKRRU CUXRJ
59 Caesar 6: MJQQT BTWQI
60 Caesar 5: LIPPS ASVPH
61 Caesar 4: KHOOR ZRUOG
62 Caesar 3: JGNNQ YQTNF
63 Caesar 2: IFMMP XPSME
64 Caesar 1: HELLO WORLD
65 Caesar 0: GDKKN VNQKC
66
67 Original String: THIS IS A TEST
68 Encrypted String: SGHR HR Z SDRS
69 Decrypted String: THIS IS A TEST
70 Solve:
71 Caesar 26: SGHR HR Z SDRS
72 Caesar 25: RFGQ GQ Y RCQR
73 Caesar 24: QEFP FP X QBPQ
74 Caesar 23: PDEO EO W PAOP
75 Caesar 22: OCDN DN V OZNO
76 Caesar 21: NBCM CM U NYMN
77 Caesar 20: MABL BL T MXLM
78 Caesar 19: LZAK AK S LWKL
79 Caesar 18: KYZJ ZJ R KVJK
80 Caesar 17: JXYI YI Q JUIJ
81 Caesar 16: IWXH XH P ITHI
82 Caesar 15: HVWG WG O HSGH
83 Caesar 14: GUVF VF N GRFG
84 Caesar 13: FTUE UE M FQEF
85 Caesar 12: ESTD TD L EPDE
86 Caesar 11: DRSC SC K DOCD
87 Caesar 10: CQRB RB J CNBC
88 Caesar 9: BPQA QA I BMAB
89 Caesar 8: AOPZ PZ H ALZA
90 Caesar 7: ZNOY OY G ZKYZ
91 Caesar 6: YMNX NX F YJXY
92 Caesar 5: XLMW MW E XIWX
93 Caesar 4: WKLW LV D WHVW
94 Caesar 3: VJKU KU C VGUV
95 Caesar 2: UIJT JT B UFTU
96 Caesar 1: THIS IS A TEST
97 Caesar 0: SGHR HR Z SDRS
98
99 Original String: HI MY NAME IS ROBBIE
100 Encrypted String: GH LX MZLD HR QNAAHD

```

```

101 Decrypted String: HI MY NAME IS ROBBIE
102 Solve:
103     Caesar 26: GH LX MZLD HR QNAAHD
104     Caesar 25: FG KW LYKC GQ PMZZGC
105     Caesar 24: EF JV KXJB FP OLYYFB
106     Caesar 23: DE IU JWIA EO NKXXEA
107     Caesar 22: CD HT IVHZ DN MJWWDZ
108     Caesar 21: BC GS HUGY CM LIVVCY
109     Caesar 20: AB FR GTFX BL KHUUBX
110     Caesar 19: ZA EQ FSEW AK JGTTAW
111     Caesar 18: YZ DP ERDV ZJ IFSSZV
112     Caesar 17: XY CO DQCU YI HERRYU
113     Caesar 16: WX BN CPBT XH GDQQXT
114     Caesar 15: VW AM BOAS WG FCPPWS
115     Caesar 14: UV ZL ANZR VF EBOOVR
116     Caesar 13: TU YK ZMYQ UE DANNUQ
117     Caesar 12: ST XJ YLXP TD CZMMTP
118     Caesar 11: RS WI XKWO SC BYLLSO
119     Caesar 10: QR VH WJVN RB AXKKRN
120     Caesar 9: PQ UG VIUM QA ZWJJQM
121     Caesar 8: OP TF UHTL PZ YVLIPL
122     Caesar 7: NO SE TGSK OY XUHHOK
123     Caesar 6: MN RD SFRJ NX WTGGNJ
124     Caesar 5: LM QC REQI MW VSFFMI
125     Caesar 4: KL PB QDPH LV UREELH
126     Caesar 3: JK OA PCOG KU TQDDKG
127     Caesar 2: IJ NZ OBNF JT SPCCJF
128     Caesar 1: HI MY NAME IS ROBBIE
129     Caesar 0: GH LX MZLD HR QNAAHD
130
131 Original String: WANDAVISION
132 Encrypted String: VZMCZUHRHNM
133 Decrypted String: WANDAVISION
134 Solve:
135     Caesar 26: VZMCZUHRHNM
136     Caesar 25: UYLBVTGQGML
137     Caesar 24: TXKAXSFPFLK
138     Caesar 23: SWJZWREOEKJ
139     Caesar 22: RVIYVQDNDJI
140     Caesar 21: QUHXUPCMCIH
141     Caesar 20: PTGWTOLBLBH
142     Caesar 19: OSFVSNKAGF
143     Caesar 18: NREURMZJZFE
144     Caesar 17: MQDTQLYIYED
145     Caesar 16: LPCSPKXHXDC
146     Caesar 15: KOBROJWGWC
147     Caesar 14: JNAQNIVFVBA
148     Caesar 13: IMZPMHUEVAZ
149     Caesar 12: HLYOLGTDZ
150     Caesar 11: GKXNKFSCSYX
151     Caesar 10: FJWMJERBRXW
152     Caesar 9: EIVLIDQAQWV
153     Caesar 8: DHUKHCPZPVU
154     Caesar 7: CGTJGBOYOUT
155     Caesar 6: BFSIFANXNTS
156     Caesar 5: AERHEZMWMSR
157     Caesar 4: ZDQGDYLVLRQ
158     Caesar 3: YCPFCXKUKQP
159     Caesar 2: XBOEBWJTJPO
160     Caesar 1: WANDAVISION
161     Caesar 0: VZMCZUHRHNM
162
163 Original String: ABED
164 Encrypted String: ZADC
165 Decrypted String: ABED

```

```

166 Solve:
167 Caesar 26: ZADC
168 Caesar 25: YZCB
169 Caesar 24: XYBA
170 Caesar 23: WXYZ
171 Caesar 22: VWZY
172 Caesar 21: UVYX
173 Caesar 20: TUXW
174 Caesar 19: STWV
175 Caesar 18: RSVU
176 Caesar 17: QRUT
177 Caesar 16: PQTS
178 Caesar 15: OPSR
179 Caesar 14: NORQ
180 Caesar 13: MNQP
181 Caesar 12: LMPO
182 Caesar 11: KLON
183 Caesar 10: JKNM
184 Caesar 9: IJML
185 Caesar 8: HILK
186 Caesar 7: GHKJ
187 Caesar 6: FGJI
188 Caesar 5: EFIH
189 Caesar 4: DEHG
190 Caesar 3: CDGF
191 Caesar 2: BCFE
192 Caesar 1: ABED
193 Caesar 0: ZADC
194
195 Original String: THE MANDALORIAN
196 Encrypted String: SGD LZMCZKNQHZM
197 Decrypted String: THE MANDALORIAN
198 Solve:
199 Caesar 26: SGD LZMCZKNQHZM
200 Caesar 25: RFC KYLBYJMPGYL
201 Caesar 24: QEB JXXAXILOFXK
202 Caesar 23: PDA IWJZWHKNEWJ
203 Caesar 22: OCZ HVIYVGJMDVI
204 Caesar 21: NBY GUHXUFILCUH
205 Caesar 20: MAX FTGWTEHKBTG
206 Caesar 19: LZW ESFVSDGJASF
207 Caesar 18: KYV DREURCFIZRE
208 Caesar 17: JXU CQDTQBEHYQD
209 Caesar 16: IWT BPCSPADGXPC
210 Caesar 15: HVS AOBROZCFWOB
211 Caesar 14: GUR ZNAQNYBEVNA
212 Caesar 13: FTQ YMZPMXADUMZ
213 Caesar 12: ESP XLYOLWZCTLY
214 Caesar 11: DRO WKXNKVYBSKX
215 Caesar 10: CQN VJWMJUXARJW
216 Caesar 9: BPM UIVLITWZQIV
217 Caesar 8: AOL THUKHSVYPHU
218 Caesar 7: ZNK SGTJGRUXOGT
219 Caesar 6: YMJ RFSIFQTWNFS
220 Caesar 5: XLI QERHEPSVMER
221 Caesar 4: WKH PDQGDORULDQ
222 Caesar 3: VJG OCPFCNQTKCP
223 Caesar 2: UIF NBOEBMPSJBO
224 Caesar 1: THE MANDALORIAN
225 Caesar 0: SGD LZMCZKNQHZM
226
227 Original String: WOW I LEARNED SCALA I THINK
228 Encrypted String: VNV H KDZQMDK RBZKZ H SGHMJ
229 Decrypted String: WOW I LEARNED SCALA I THINK
230 Solve:

```

231	Caesar 26: VNV H KDZQMDC RBZKZ H SGHMJ
232	Caesar 25: UMU G JCYPLCB QAYJY G RFGLI
233	Caesar 24: TLT F IBXOKBA PZXIX F QEFKH
234	Caesar 23: SKS E HAWNJAZ OYWHW E PDEJG
235	Caesar 22: RJR D GZVMIZY NXVGV D OCDIF
236	Caesar 21: QIQ C FYULHYX MWUFU C NBCHE
237	Caesar 20: PHP B EXTKGXW LVTET B MABGD
238	Caesar 19: OGO A DWSJFWV KUSDS A LZAFC
239	Caesar 18: NFN Z CVRIEVU JTRCR Z KYZEB
240	Caesar 17: MEM Y BUQHDUT ISQBQ Y JXYDA
241	Caesar 16: LDL X ATPGCTS HRPAP X IWXCZ
242	Caesar 15: KCK W ZSOFBSR GQOZO W HVWBY
243	Caesar 14: JBJ V YRNEARQ FPNYN V GUVAX
244	Caesar 13: IAI U XQMDZQP EOMXM U FTUZW
245	Caesar 12: HZH T WPLCYPO DNLWL T ESTYV
246	Caesar 11: GYG S VOKBXON CMKVK S DRSXU
247	Caesar 10: FXF R UNJAWNM BLJUJ R CQRWT
248	Caesar 9: EWE Q TMIZVML AKITI Q BPQVS
249	Caesar 8: DVD P SLHYULK ZJHSH P AOPUR
250	Caesar 7: CUC O RKGXTKJ YIGRG O ZNOTQ
251	Caesar 6: BTB N QJFWSJI XHFQF N YMNSP
252	Caesar 5: ASA M PIEVRIH WGEPE M XLMRO
253	Caesar 4: ZRZ L OHDUQHG VFDOD L WKLQN
254	Caesar 3: YQY K NGCTPGF UECNC K VJKPM
255	Caesar 2: XPX J MFBSOFE TDBMB J UIJOL
256	Caesar 1: WOW I LEARNED SCALA I THINK
257	Caesar 0: VNV H KDZQMDC RBZKZ H SGHMJ

## 6 MAIN GOOGLE SEARCHES

### 6.1 FORTRAN

<https://www.tutorialspoint.com/fortran/index.htm>

[https://www.youtube.com/watch?v=\\_\\_2UgFNYgf8](https://www.youtube.com/watch?v=__2UgFNYgf8)

### 6.2 COBOL

<https://www.tutorialspoint.com/cobol/index.htm>

<https://www.youtube.com/watch?v=TBs7HXI76yU>

### 6.3 BASIC

<https://www.tutorialspoint.com/vb.net/index.htm>

<https://docs.microsoft.com/en-us/dotnet/visual-basic/>

### 6.4 PASCAL

<https://www.tutorialspoint.com/pascal/index.htm>

### 6.5 SCALA

<https://www.tutorialspoint.com/scala/index.htm>

## 7 LANGUAGE RANKINGS

1. Visual BASIC
2. Scala
3. Pascal
4. Fortran
5. COBOL