Functional Programming

Robert Perrone

robert.perrone1@marist.edu

May 15, 2021

1 LISP

1.1 Consulting Log

Expected hours needed: 14

Date	***Hours Spent***	***Tasks / Accomplishments / Issues / Thoughts***
05/12/2021	2	I'm really struggling to understand the syntax. Between string variables, list variables and functions, everything is not easy to me. I was not able to start this sooner and now it feels as though I won't be able to finish this one.
05/14/2021	3.5	I'm trying really hard to understand what is going on here. I feel like I'm close and when I look at different resources, my syntax seems right but when I run it, I get so many weird and unhelpful error messages. I honestly give up. If I had a few more days I could maybe do it.

1.2 Commentary

I might have been able to finish this with some extra time, but this week has been hell and I could not get through LISP. I really wanted to actually get this to work but I'm too confused and my head is hurting from trying to think of what is wrong here.
I would still like to actually learn LISP if you could potentially help me during office hours.

1.3 Source Code

```
(defvar testStr1 "IBM")
(defvar testStr2 "Hello World")
(defvar testStr3 "This is a test")
(defvar testStr4 "Hi my name is Robbie")
(defvar testStr5 "WandaVision")
(defvar testStr6 "Abed")
(defvar testStr7 "The Mandalorian")
(defvar testStr8 "Wow I learned LISP I think")

```

```
(defvar testStrList (list testStr1 testStr2 testStr3 testStr4 testStr5 testStr6 testStr7 testStr8))
(defvar shiftAmount (- 1))

(defun encrypt(testStrList)
    ;(mapcar #'pickShift' testStrList)
        (maplist (lambda (testStr) (pickShift testStr)) testStrList)
        ;(pickShift(car testStrList))
        ;(encrypt (rest testStrList))
)

(defun pickShift(testStr)
        (cond
            ;((< shiftAmount 0) (posShift testStr))
            ((< shiftAmount 0) (map 'string #'(lambda (character) (negShift character)) testStr))
                ;;((> shiftAmount 0) (map 'string #' (lambda (character) (posShift(character shiftAmount))
                ;((< shiftAmount 0) (negShift(car testStr)))
                (t (testStr))
        )
        ;(pickShift(rest testStr))
)

(defun negShift(character)
        (setq asciiVal (char-code character))
        (cond
            ((or (char= #\A character) (char= #\a character)) (char(+ shiftAmount (+ asciiVal 26))))
                ((and (char> #\A character) (char<= #\Z character)) (char(+ shiftAmount asciiVal)))
                ((and (char> #\a character) (char<= #\z character)) (char(+ shiftAmount asciiVal)))
                (t (character))
        )
)

(format t "Original Strings:~%~s~%" testStrList)
(format t "Encrypted Strings:~%~s~%" (encrypt testStrList))

(write (map 'list (lambda (x) (+ x 10)) '(1 2 3 4)))

;;; ~a : shows the value
;;; ~s : shows quotes around the value
;;; ~10a : Adds 10 spaces for the value with the extra space to the right
;;; ~10Aa : Adds 10 spaces for the value with the extra space to the left
```

1.4 Output

```
Original Strings:
("IBM" "Hello World" "This is a test" "Hi my name is Robbie" "WandaVision"
 "Abed" "The Mandalorian" "Wow I learned LISP I think")
WARNING: DEFUN/DEFMACRO(ENCRYPT): #<PACKAGE POSIX> is locked
         Ignore the lock and proceed
WARNING: DEFUN/DEFMACRO: redefining function ENCRYPT in
         /home/cg/root/7262401/main.lisp, was defined in C
*** - CHAR-CODE: argument "IBM" is not a character
```

2 F#

2.1 Consulting Log

Expected hours needed: 12

Date	***Hours Spent***	***Tasks / Accomplishments / Issues / Thoughts***
05/10/2021	1.5	I started looking into the basic syntax and writing the basis for the code as usual. I then began to look at the function to do mapping and recursion.
05/11/2021	3	For some reason, when I tried to call a different function in my recursion function, I was getting weird errors that I did not yet understand. I later realized that I just needed to define the function before I called it, which seemed so trivial when I realized what was happening. The other languages that we were working in up until now did not care as much as F#. I then figured out encrypt after a while of fighting the syntax, which then quickly led to decrypt. Later that day I then figured out the solve, which was achieved through a for loop.

2.2 Commentary

F# was interesting, it wasn't too bad since it had a good amount of resources out there online. I enjoyed it for the most part, the syntax was just strange at first, but I got the hang of it pretty quickly once I realized what was going on.

2.3 Source Code

```
open System

//(*Negative Shift*)//
let negShift(character, shiftAmount) =
  let asciiVal = int character
  //if char is A or a
  if (character = 'A') || (character = 'a') then
    char (asciiVal + shiftAmount + 26)
  //if B >= character <= Z or b >= character < Z
  elif (List.contains character ['B'..'Z']) || (List.contains character ['b'..'z']) then
    char (asciiVal + shiftAmount)
  else character
//(*Positive Shift*)//
let posShift(character, shiftAmount) =
  let asciiVal = int character
  //if char is Z or z
  if (character = 'Z') || (character = 'z') then
    char (asciiVal + shiftAmount - 26)
  //if A >= character <= Y or a >= character < y
  elif (List.contains character ['A'..'Y']) || (List.contains character ['a'..'y']) then
    char (asciiVal + shiftAmount)
  else character

//(*Decide which *)//
let pickShift(testStr, shiftAmount) =
  if (shiftAmount < 0) then
    String.map(fun character -> negShift(character, shiftAmount)) testStr
  elif (shiftAmount > 0) then
    String.map(fun character -> posShift(character, shiftAmount)) testStr
  else testStr

//(*ENCRYPT*)//
let encrypt(testStrList, shiftAmount) =
```

```
  List.map(fun testStr -> pickShift(testStr, shiftAmount)) testStrList
  (*match testStrList with
  | [] -> []
  | head :: tail ->
    pickShift(head, shiftAmount)
    encrypt(tail, shiftAmount)*)
//(*DECRYPT*)//
let decrypt(encryptedList, shiftAmount) =
  List.map(fun encryptedStr -> pickShift(encryptedStr, shiftAmount)) encryptedList
//(*SOLVE*)//
let rec solve(solvedList, shiftAmount, maxShift, solveNum) =
  let newList = List.map(fun encryptedStr -> pickShift(encryptedStr, shiftAmount)) solvedList
  printfn "%A" solvedList
  if solveNum <> maxShift then
    solve(newList, shiftAmount, maxShift, (solveNum + 1))
  else newList

//(*MAIN*)//
let main() =
  let testStr1 = "IBM"
  let testStr2 = "Hello World"
  let testStr3 = "This is a test"
  let testStr4 = "Hi my name is Robbie"
  let testStr5 = "WandaVision"
  let testStr6 = "Abed"
  let testStr7 = "The Mandalorian"
  let testStr8 = "Wow I learned F# I think"

  let testStrList = [testStr1; testStr2; testStr3; testStr4; testStr5; testStr6; testStr7; testStr8;]

  let shiftAmount = (-1)
  let maxShift = 26
  let solveNum = 0

  printfn "Original Strings:"
  printfn "%A" testStrList

  printfn "\nEncrypted Strings:"
  let encryptedList = encrypt(testStrList, shiftAmount)
  printfn "%A" encryptedList

  printfn "\nDecrypted Strings:"
  let decryptedList = decrypt(encryptedList, (-shiftAmount))
  printfn "%A" decryptedList

  printfn "\nSolve:"
  let mutable solvedList = encryptedList
  solvedList <- solve(solvedList, shiftAmount, maxShift, solveNum)

main()
```

2.4 Output

```
Original Strings:
["IBM"; "Hello World"; "This is a test"; "Hi my name is Robbie"; "WandaVision";
 "Abed"; "The Mandalorian"; "Wow I learned F# I think"]

Encrypted Strings:
["HAL"; "Gdkkn Vnqkc"; "Sghr hr z sdrs"; "Gh lx mzld hr Qnaahd"; "VzmczUhrhnm";
 "Zadc"; "Sgd Lzmczknqhzm"; "Vnv H kdzqmdc E# H sghmj"]

Decrypted Strings:
["IBM"; "Hello World"; "This is a test"; "Hi my name is Robbie"; "WandaVision";
```

```
 "Abed"; "The Mandalorian"; "Wow I learned F# I think"]

Solve:
["HAL"; "Gdkkn Vnqkc"; "Sghr hr z sdrs"; "Gh lx mzld hr Qnaahd"; "VzmczUhrhnm";
 "Zadc"; "Sgd Lzmczknqhzm"; "Vnv H kdzqmdc E# H sghmj"]
["GZK"; "Fcjjm Umpjb"; "Rfgq gq y rcqr"; "Fg kw lykc gq Pmzzgc"; "UylbyTgqgml";
 "Yzcb"; "Rfc Kylbyjmpgyl"; "Umu G jcyplcb D# G rfgli"]
["FYJ"; "Ebiil Tloia"; "Qefp fp x qbpq"; "Ef jv kxjb fp Olyyfb"; "TxkaxSfpflk";
 "Xyba"; "Qeb Jxkaxilofxk"; "Tlt F ibxokba C# F qefkh"]
["EXI"; "Dahhk Sknhz"; "Pdeo eo w paop"; "De iu jwia eo Nkxxea"; "SwjzwReoekj";
 "Wxaz"; "Pda Iwjzwhknewj"; "Sks E hawnjaz B# E pdejg"]
["DWH"; "Czggj Rjmgy"; "Ocdn dn v ozno"; "Cd ht ivhz dn Mjwwdz"; "RviyvQdndji";
 "Vwzy"; "Ocz Hviyvgjmdvi"; "Rjr D gzvmizy A# D ocdif"]
["CVG"; "Byffi Qilfx"; "Nbcm cm u nymn"; "Bc gs hugy cm Livvcy"; "QuhxuPcmcih";
 "Uvyx"; "Nby Guhxufilcuh"; "Qiq C fyulhyx Z# C nbche"]
["BUF"; "Axeeh Phkew"; "Mabl bl t mxlm"; "Ab fr gtfx bl Khuubx"; "PtgwtOblbhg";
 "Tuxw"; "Max Ftgwtehkbtg"; "Php B extkgxw Y# B mabgd"]
["ATE"; "Zwddg Ogjdv"; "Lzak ak s lwkl"; "Za eq fsew ak Jgttaw"; "OsfvsNakagf";
 "Stwv"; "Lzw Esfvsdgjasf"; "Ogo A dwsjfwv X# A lzafc"]
["ZSD"; "Yvccf Nficu"; "Kyzj zj r kvjk"; "Yz dp erdv zj Ifsszv"; "NreurMzjzfe";
 "Rsvu"; "Kyv Dreurcfizre"; "Nfn Z cvrievu W# Z kyzeb"]
["YRC"; "Xubbe Mehbt"; "Jxyi yi q juij"; "Xy co dqcu yi Herryu"; "MqdtqLyiyed";
 "Qrut"; "Jxu Cqdtqbehyqd"; "Mem Y buqhdut V# Y jxyda"]
["XQB"; "Wtaad Ldgas"; "Iwxh xh p ithi"; "Wx bn cpbt xh Gdqqxt"; "LpcspKxhxdc";
 "Pqts"; "Iwt Bpcspadgxpc"; "Ldl X atpgcts U# X iwxcz"]
["WPA"; "Vszzc Kcfzr"; "Hvwg wg o hsgh"; "Vw am boas wg Fcppws"; "KobroJwgwcb";
 "Opsr"; "Hvs Aobrozcfwob"; "Kck W zsofbsr T# W hvwby"]
["VOZ"; "Uryyb Jbeyq"; "Guvf vf n grfg"; "Uv zl anzr vf Eboovr"; "JnaqnIvfvba";
 "Norq"; "Gur Znaqnybevna"; "Jbj V yrnearq S# V guvax"]
["UNY"; "Tqxxa Iadxp"; "Ftue ue m fqef"; "Tu yk zmyq ue Dannuq"; "ImzpmHueuaz";
 "Mnqp"; "Ftq Ymzpmxadumz"; "Iai U xqmdzqp R# U ftuzw"]
["TMX"; "Spwwz Hzcwo"; "Estd td l epde"; "St xj ylxp td Czmmtp"; "HlyolGtdtzy";
 "Lmpo"; "Esp Xlyolwzctly"; "Hzh T wplcypo Q# T estyv"]
["SLW"; "Rovvy Gybvn"; "Drsc sc k docd"; "Rs wi xkwo sc Byllso"; "GkxnkFscsyx";
 "Klon"; "Dro Wkxnkvybskx"; "Gyg S vokbxon P# S drsxu"]
["RKV"; "Qnuux Fxaum"; "Cqrb rb j cnbc"; "Qr vh wjvn rb Axkkrn"; "FjwmjErbrxw";
 "Jknm"; "Cqn Vjwmjuxarjw"; "Fxf R unjawnm O# R cqrwt"]
["QJU"; "Pmttw Ewztl"; "Bpqa qa i bmab"; "Pq ug vium qa Zwjjqm"; "EivliDqaqwv";
 "Ijml"; "Bpm Uivlitwzqiv"; "Ewe Q tmizvml N# Q bpqvs"]
["PIT"; "Olssv Dvysk"; "Aopz pz h alza"; "Op tf uhtl pz Yviipl"; "DhukhCpzpvu";
 "Hilk"; "Aol Thukhsvyphu"; "Dvd P slhyulk M# P aopur"]
["OHS"; "Nkrru Cuxrj"; "Znoy oy g zkyz"; "No se tgsk oy Xuhhok"; "CgtjgBoyout";
 "Ghkj"; "Znk Sgtjgruxogt"; "Cuc O rkgxtkj L# O znotq"]
["NGR"; "Mjqqt Btwqi"; "Ymnx nx f yjxy"; "Mn rd sfrj nx Wtggnj"; "BfsifAnxnts";
 "Fgji"; "Ymj Rfsifqtwnfs"; "Btb N qjfwsji K# N ymnsp"]
["MFQ"; "Lipps Asvph"; "Xlmw mw e xiwx"; "Lm qc reqi mw Vsffmi"; "AerheZmwmsr";
 "Efih"; "Xli Qerhepsvmer"; "Asa M pievrih J# M xlmro"]
["LEP"; "Khoor Zruog"; "Wklv lv d whvw"; "Kl pb qdph lv Ureelh"; "ZdqgdYlvlrq";
 "Dehg"; "Wkh Pdqgdoruldq"; "Zrz L ohduqhg I# L wklqn"]
["KDO"; "Jgnnq Yqtnf"; "Vjku ku c vguv"; "Jk oa pcog ku Tqddkg"; "YcpfcXkukqp";
 "Cdgf"; "Vjg Ocpfcnqtkcp"; "Yqy K ngctpgf H# K vjkpm"]
["JCN"; "Ifmmp Xpsme"; "Uijt jt b uftu"; "Ij nz obnf jt Spccjf"; "XboebWjtjpo";
 "Bcfe"; "Uif Nboebmpsjbo"; "Xpx J mfbsofe G# J uijol"]
["IBM"; "Hello World"; "This is a test"; "Hi my name is Robbie"; "WandaVision";
 "Abed"; "The Mandalorian"; "Wow I learned F# I think"]
["HAL"; "Gdkkn Vnqkc"; "Sghr hr z sdrs"; "Gh lx mzld hr Qnaahd"; "VzmczUhrhnm";
 "Zadc"; "Sgd Lzmczknqhzm"; "Vnv H kdzqmdc E# H sghmj"]
```

3 Erlang

3.1 Consulting Log

Expected hours needed: 10

Date	*Hours Spent*	*Tasks / Accomplishments / Issues / Thoughts*
05/02/2021	2	To take break from Haskell, I started Erlang. I got some of the concepts pretty quickly, some stuck from Haskell and are pretty similar. The thing that is giving e the most trouble is the syntax of it because I am finding it hard to read. In examples, I am kind of lost on how to translate the map function to what I am trying to do and how to apply it in another function. I thought I had it at one point, and then it did not translate to the next thing I tried. I'm also trying to figure out what the keyword "fun" does still
05/05/2021	2	I finally got some of the syntax down. When I read up on Erlang, it said that the language was designed to be easy to use but I had a lot of trouble in the beginning trying to figure out what everything meant and trying to figure out its syntax. Thankfully I got somewhere today and learned how to use recursion in this syntax. I was trying to find something similar to the (x:xs) notation in haskell, which I found with [H|T]. Taking the head and applying a new function while taking the tail and applying that to the same function (aka: recursion!).
05/07/2021	1	I worked on this for maybe an hour and finally got the encrypt function mostly working. I changed the functions to use the built-in map functions instead of using the [H|T] map functions that I made. This was so that I could include multiple arguments in my recursion instead of just the list. For some reason it's only changing z to a, but it also decrypts just fine, because all it does is change a back a to z. There must be some error in my logic in my if statement but I'll have to look at it later.
05/09/2021	2	I worked on this a little in the morning and finally saw my error in logic. Once I got that fixed, the decrypt function fell into place and all that was left was solve. Solve has been tricky with the functional languages than with the procedural languages which I found very easy once encrypt was done. But it took my a while to figure out the best way to go about repeating that solve function over and over again while also being able to print out all of the different lists in the iterations.

3.2 Commentary

I found the syntax of Erlang a bit more confusing than Haskell, especially at first. It was tough for me to find resources that explained it well at first, and I only began to understand the syntax while looking at a lot of different examples in other people's code. Once I got the hang of it, it was not terrible, but the beginning was really rough and I thought that Haskell did a slightly better job with the syntax. I also was definitely confused with the map function for longer than I though I should. It seemed to me that Haskell's map function was more flexible in how I could use it, while Erlang's was more restrictive. Although I did like Erlang's multiple ways to do recursion/making your own mapping function which was similar to Haskell.

3.3 Source Code

```
1 -module(prog).
2 %-import(string, [len/1, concat/2, chr/2, substr/3, str/2, to_lower/1, to_upper/1]).
3 %-define(ShiftAmount, -1). %Might need this constant global variable
```

```
-export([main/0]).

main() ->
        % your code goes here
        TestStr1 = "IBM",
        TestStr2 = "Hello World",
        TestStr3 = "This is a test",
        TestStr4 = "Hi my name is Robbie",
        TestStr5 = "WandaVision",
        TestStr6 = "Abed",
        TestStr7 = "The Mandalorian",
        TestStr8 = "Wow I learned Erlang I think",

        TestStrList = [TestStr1,TestStr2,TestStr3,TestStr4,TestStr5,TestStr6,TestStr7,TestStr8],

        ShiftAmount = (-1),
        MaxShift = 25,
        SolveNum = 0,

        io:fwrite("Original Strings:~n"),
        io:fwrite("~p~n", [TestStrList]),

        io:fwrite("~nEncrypted Strings:~n"),
        EncryptedList = encrypt(TestStrList, ShiftAmount),
        io:fwrite("~p~n", [EncryptedList]),

        io:fwrite("~nDecrypted Strings:~n"),
        DecryptedList = decrypt(EncryptedList, -ShiftAmount),
        io:fwrite("~p~n", [DecryptedList]),

        io:fwrite("~nSolve:~n"),
        SolvedList = EncryptedList,
        io:fwrite("~p~n ", [SolvedList]),
        SolvedList = solve(SolvedList, ShiftAmount, MaxShift, SolveNum).
        %io:fwrite("~p~n", [SolvedList]).

%encrypt([]) -> [];
%encrypt([H|T]) -> [enShift(H)|encrypt(T)].

% Encrypt
encrypt(TestStrList, ShiftAmount) ->
        lists:map(fun(TestStr) -> pickShift(TestStr, ShiftAmount) end, TestStrList).
% Decrypt
decrypt(EncryptedList, ShiftAmount) ->
        lists:map(fun(TestStr) -> pickShift(TestStr, ShiftAmount) end, EncryptedList).
% Solve
solve(SolvedList, ShiftAmount, MaxShift, SolveNum) ->
        %SolvedList = lists:append(SolvedList, lists:map(fun(TestStr) -> pickShift(TestStr, ShiftAmount) en
        NewList = lists:map(fun(TestStr) -> pickShift(TestStr, ShiftAmount) end, SolvedList),
        io:fwrite("~p~n", [NewList]),
        %io:fwrite("~p~n Caesar ", MaxShift - SolveNum, " : ", [EncryptedList]),

        if
                %Recursive call if SolveNum does not equal MaxShift Value
                SolveNum /= MaxShift -> solve(NewList, ShiftAmount, MaxShift, (SolveNum + 1));
                %Default to return NewList
                true -> NewList
        end.

% Decide whether shift is positive of negative and then decide which shift function to call
pickShift(TestStr, ShiftAmount) ->
        if
                % If shift amount is negative
                ShiftAmount < 0 -> lists:map(fun(Char) -> negShift(Char, ShiftAmount) end, TestStr);
                % If shift amount is positive
```

```
                ShiftAmount > 0 -> lists:map(fun(Char) -> posShift(Char, ShiftAmount) end, TestStr);
                true -> TestStr
        end.

% Shifting if ShiftAmount is a negative value
negShift(Character, ShiftAmount) ->
        %io:fwrite("~c~n", [Character]),
        if
                % shift character based on what the character is
                Character == $A -> Character + ShiftAmount + 26;
                Character == $a -> Character + ShiftAmount + 26;
                % includes everything but A
                (Character > $A) and (Character =< $Z) -> Character + ShiftAmount;
                (Character > $a) and (Character =< $z) -> Character + ShiftAmount;
                true -> Character
        end.
% Shifting if ShiftAmount is a positive value
posShift(Character, ShiftAmount) ->
        if
                % shift character based on what the character is
                Character == $Z -> Character + ShiftAmount - 26;
                Character == $z -> Character + ShiftAmount - 26;
                % includes everything but Z
                ((Character >= $A) and (Character < $Z)) -> Character + ShiftAmount;
                ((Character >= $a) and (Character < $z)) -> Character + ShiftAmount;
                true -> Character
        end.
```

3.4 Output

```
Original Strings:
["IBM","Hello World","This is a test","Hi my name is Robbie","WandaVision",
 "Abed","The Mandalorian","Wow I learned Erlang I think"]

Encrypted Strings:
["HAL","Gdkkn Vnqkc","Sghr hr z sdrs","Gh lx mzld hr Qnaahd","VzmczUhrhnm",
 "Zadc","Sgd Lzmczknqhzm","Vnv H kdzqmdc Dqkzmf H sghmj"]

Decrypted Strings:
["IBM","Hello World","This is a test","Hi my name is Robbie","WandaVision",
 "Abed","The Mandalorian","Wow I learned Erlang I think"]

Solve:
["HAL","Gdkkn Vnqkc","Sghr hr z sdrs","Gh lx mzld hr Qnaahd","VzmczUhrhnm",
 "Zadc","Sgd Lzmczknqhzm","Vnv H kdzqmdc Dqkzmf H sghmj"]
["GZK","Fcjjm Umpjb","Rfgq gq y rcqr","Fg kw lykc gq Pmzzgc","UylbyTgqgml",
 "Yzcb","Rfc Kylbyjmpgyl","Umu G jcyplcb Cpjyle G rfgli"]
["FYJ","Ebiil Tloia","Qefp fp x qbpq","Ef jv kxjb fp Olyyfb","TxkaxSfpflk",
 "Xyba","Qeb Jxkaxilofxk","Tlt F ibxokba Boixkd F qefkh"]
["EXI","Dahhk Sknhz","Pdeo eo w paop","De iu jwia eo Nkxxea","SwjzwReoekj",
 "Wxaz","Pda Iwjzwhknewj","Sks E hawnjaz Anhwjc E pdejg"]
["DWH","Czggj Rjmgy","Ocdn dn v ozno","Cd ht ivhz dn Mjwwdz","RviyvQdndji",
 "Vwzy","Ocz Hviyvgjmdvi","Rjr D gzvmizy Zmgvib D ocdif"]
["CVG","Byffi Qilfx","Nbcm cm u nymn","Bc gs hugy cm Livvcy","QuhxuPcmcih",
 "Uvyx","Nby Guhxufilcuh","Qiq C fyulhyx Ylfuha C nbche"]
["BUF","Axeeh Phkew","Mabl bl t mxlm","Ab fr gtfx bl Khuubx","PtgwtOblbhg",
 "Tuxw","Max Ftgwtehkbtg","Php B extkgxw Xketgz B mabgd"]
["ATE","Zwddg Ogjdv","Lzak ak s lwkl","Za eq fsew ak Jgttaw","OsfvsNakagf",
 "Stwv","Lzw Esfvsdgjasf","Ogo A dwsjfwv Wjdsfy A lzafc"]
["ZSD","Yvccf Nficu","Kyzj zj r kvjk","Yz dp erdv zj Ifsszv","NreurMzjzfe",
 "Rsvu","Kyv Dreurcfizre","Nfn Z cvrievu Vicrex Z kyzeb"]
["YRC","Xubbe Mehbt","Jxyi yi q juij","Xy co dqcu yi Herryu","MqdtqLyiyed",
 "Qrut","Jxu Cqdtqbehyqd","Mem Y buqhdut Uhbqdw Y jxyda"]
```

```
["XQB","Wtaad Ldgas","Iwxh xh p ithi","Wx bn cpbt xh Gdqqxt","LpcspKxhxdc",
 "Pqts","Iwt Bpcspadgxpc","Ldl X atpgcts Tgapcv X iwxcz"]
["WPA","Vszzc Kcfzr","Hvwg wg o hsgh","Vw am boas wg Fcppws","KobroJwgwcb",
 "Opsr","Hvs Aobrozcfwob","Kck W zsofbsr Sfzobu W hvwby"]
["VOZ","Uryyb Jbeyq","Guvf vf n grfg","Uv zl anzr vf Eboovr","JnaqnIvfvba",
 "Norq","Gur Znaqnybevna","Jbj V yrnearq Reynat V guvax"]
["UNY","Tqxxa Iadxp","Ftue ue m fqef","Tu yk zmyq ue Dannuq","ImzpmHueuaz",
 "Mnqp","Ftq Ymzpmxadumz","Iai U xqmdzqp Qdxmzs U ftuzw"]
["TMX","Spwwz Hzcwo","Estd td l epde","St xj ylxp td Czmmtp","HlyolGtdtzy",
 "Lmpo","Esp Xlyolwzctly","Hzh T wplcypo Pcwlyr T estyv"]
["SLW","Rovvy Gybvn","Drsc sc k docd","Rs wi xkwo sc Byllso","GkxnkFscsyx",
 "Klon","Dro Wkxnkvybskx","Gyg S vokbxon Obvkxq S drsxu"]
["RKV","Qnuux Fxaum","Cqrb rb j cnbc","Qr vh wjvn rb Axkkrn","FjwmjErbrxw",
 "Jknm","Cqn Vjwmjuxarjw","Fxf R unjawnm Naujwp R cqrwt"]
["QJU","Pmttw Ewztl","Bpqa qa i bmab","Pq ug vium qa Zwjjqm","EivliDqaqwv",
 "Ijml","Bpm Uivlitwzqiv","Ewe Q tmizvml Mztivo Q bpqvs"]
["PIT","Olssv Dvysk","Aopz pz h alza","Op tf uhtl pz Yviipl","DhukhCpzpvu",
 "Hilk","Aol Thukhsvyphu","Dvd P slhyulk Lyshun P aopur"]
["OHS","Nkrru Cuxrj","Znoy oy g zkyz","No se tgsk oy Xuhhok","CgtjgBoyout",
 "Ghkj","Znk Sgtjgruxogt","Cuc O rkgxtkj Kxrgtm O znotq"]
["NGR","Mjqqt Btwqi","Ymnx nx f yjxy","Mn rd sfrj nx Wtggnj","BfsifAnxnts",
 "Fgji","Ymj Rfsifqtwnfs","Btb N qjfwsji Jwqfsl N ymnsp"]
["MFQ","Lipps Asvph","Xlmw mw e xiwx","Lm qc reqi mw Vsffmi","AerheZmwmsr",
 "Efih","Xli Qerhepsvmer","Asa M pievrih Ivperk M xlmro"]
["LEP","Khoor Zruog","Wklv lv d whvw","Kl pb qdph lv Ureelh","ZdqgdYlvlrq",
 "Dehg","Wkh Pdqgdoruldq","Zrz L ohduqhg Huodqj L wklqn"]
["KDO","Jgnnq Yqtnf","Vjku ku c vguv","Jk oa pcog ku Tqddkg","YcpfcXkukqp",
 "Cdgf","Vjg Ocpfcnqtkcp","Yqy K ngctpgf Gtncpi K vjkpm"]
["JCN","Ifmmp Xpsme","Uijt jt b uftu","Ij nz obnf jt Spccjf","XboebWjtjpo",
 "Bcfe","Uif Nboebmpsjbo","Xpx J mfbsofe Fsmboh J uijol"]
["IBM","Hello World","This is a test","Hi my name is Robbie","WandaVision",
 "Abed","The Mandalorian","Wow I learned Erlang I think"]
["HAL","Gdkkn Vnqkc","Sghr hr z sdrs","Gh lx mzld hr Qnaahd","VzmczUhrhnm",
 "Zadc","Sgd Lzmczknqhzm","Vnv H kdzqmdc Dqkzmf H sghmj"]
```

4 Haskell

4.1 Consulting Log

Expected hours needed: 10

Date	*Hours Spent*	*Tasks / Accomplishments / Issues / Thoughts*
04/25/2021	1.5	Spent a majority of the time just learning the basic syntax and setting up the base of the code. I started looking into how to create the functions and although it looks confusing, I think I got it by the end for the most part. I mostly just need to figure out the syntax for making it more than 1 line, which I think uses the keyword "Where".
04/30/2021	3	I spent a lot of time with the mapping functions and trying to see what I can do with them. I was able to map a list of letters A to Z, shifting them all back a letter. What I am now trying to do is apply the shift to each item in my list of strings. This brought in the issue of basically applying two mapping functions. One used to affect every string in the list, and then one to be able to map a shift onto each character. It is not working right now, but I hope to have it somewhat figured out soon.
05/01/2021	1.5	I figured out the encrypt and decrypt functions pretty much. It does not print out the way I would normally want it to, but I will look into it tomorrow. I will also work on the solve function tomorrow. As for the code I have a this point, I created my own map function, which then passes each individual string in the list. Next, each character in that string is mapped to a specific negative or positive shift depending on the value of shiftAmount.
05/02/2021	2	I ended having to redo some things, realizing that the shifting would only work if the shift is positive or negative and it would not take into account the actual value. Therefore, all shifts would be treated like -1 or +1 even if shiftAmount = -10. I then had to make a couple of other functions to help with decrypting because I needed the opposite of shiftAmount or (-shiftAmount). The solve function has proved to be more difficult and I need to figure out how to repeat the shift 25 times.
05/04/2021	0.5	I finally figured out the solve function after many google searches of how I can loop over a function a certain number of times "n times". It was the simplest solution I found although there seemed to be many other out there. The "iterate" keyword was great, but I needed a way to stop it which is where the "take n" keyword came into play.

4.2 Commentary

Just being able to think in a functional manner to this extent was a lot for my head. I eventually was able to wrap my head around it, but it took me so long to be able to think about not what but how I was going to do it in this way. I did not get to print the output the way I would have liked, since I was not able to print inside of the functions that I was calling. Overall, Haskell was very interesting, it was really annoying to get started with but I actually ended up enjoying it's elegance and the amount of built-in functions available was great. Anything that I needed to do seemed to have an elegant funciton attached. The learning curve was still annoying though.

4.3 Source Code

```
1 import Data.Char
```

```
import System.IO
import Control.Monad

shiftAmount = (-1) -- Global Shift (Constant)

--Alternate Solution, issue was that I could not print out
--repeatNtimes :: (Num n, Ord n) => n -> (a -> a) -> a -> a
--repeatNtimes 1 f x = f x
--repeatNtimes maxShift f x = f (repeatNtimes (maxShift-1) f x) -> print(f x)

--Encrypt
encrypt :: [String] -> [String]
encrypt [] = [] --return blank list if given blank list
encrypt(x:xs) = enShift x : encrypt xs --Own map function to "enShift" each item in the list
--Decrypt
decrypt :: [String] -> [String]
decrypt [] = [] --return blank list if given blank list
decrypt(x:xs) = deShift x : decrypt xs --Own map function to "deShift" each item in the list
--Solve
solve :: [String] -> [String]
solve [] = []
solve(x:xs) = enShift x : solve xs

--Encrypt Shift to decide based on +shiftAmount
enShift :: [Char] -> [Char]
enShift testStr
  | shiftAmount < 0 = (map negEnShift testStr)
  | shiftAmount > 0 = (map posEnShift testStr)
  | shiftAmount == 0 = testStr
--Decrypt Shift to decide based on -shiftAmount
deShift :: [Char] -> [Char]
deShift testStr
  | (-shiftAmount) < 0 = (map negDeShift testStr)
  | (-shiftAmount) > 0 = (map posDeShift testStr)
  | (-shiftAmount) == 0 = testStr

--Negative Encrypt Shift (with +shiftAmount)
negEnShift :: Char -> Char
negEnShift character
  | character == 'A' = chr ((ord character) + shiftAmount + 26)
  | character == 'a' = chr ((ord character) + shiftAmount + 26)
  | character `elem` ['B'..'Z'] = chr ((ord character) + shiftAmount)
  | character `elem` ['b'..'z'] = chr ((ord character) + shiftAmount)
  | otherwise = character
--Positive Encrypt Shift (with +shiftAmount)
posEnShift :: Char -> Char
posEnShift character
  | character == 'Z' = chr ((ord character) + shiftAmount - 26)
  | character == 'z' = chr ((ord character) + shiftAmount - 26)
  | character `elem` ['A'..'Y'] = chr ((ord character) + shiftAmount)
  | character `elem` ['a'..'y'] = chr ((ord character) + shiftAmount)
  | otherwise = character

--Negative Decrypt Shift (with -shiftAmount)
negDeShift :: Char -> Char
negDeShift character
  | character == 'A' = chr ((ord character) - shiftAmount + 26)
  | character == 'a' = chr ((ord character) - shiftAmount + 26)
  | character `elem` ['B'..'Z'] = chr ((ord character) - shiftAmount)
  | character `elem` ['b'..'z'] = chr ((ord character) - shiftAmount)
  | otherwise = character
--Positive Decrypt Shift (with -shiftAmount)
posDeShift :: Char -> Char
posDeShift character
  | character == 'Z' = chr ((ord character) - shiftAmount - 26)
```

```
  | character == 'z' = chr ((ord character) - shiftAmount - 26)
  | character `elem` ['A'..'Y'] = chr ((ord character) - shiftAmount)
  | character `elem` ['a'..'y'] = chr ((ord character) - shiftAmount)
  | otherwise = character

main = do
  let testStr1 = "IBM"
  let testStr2 = "Hello World"
  let testStr3 = "This is a test"
  let testStr4 = "Hi my name is Robbie"
  let testStr5 = "WandaVision"
  let testStr6 = "Abed"
  let testStr7 = "The Mandalorian"
  let testStr8 = "Wow I learned Haskell I think"

  let testStrList = [testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7, testStr8]

  let maxShift = 27 --includes "Caesar 26:" to "Caesar 0:"

  -- FUNCTION CALLS --
  putStrLn "Original Strings: "
  mapM putStrLn testStrList
  putStrLn ""

  putStrLn "Encrypted Strings: "
  let encryptedList = encrypt testStrList
  mapM putStrLn encryptedList
  putStrLn ""

  putStrLn "Decrypted Strings: "
  let decryptedList = decrypt encryptedList
  mapM putStrLn decryptedList
  putStrLn ""

  putStrLn "Solve: "
  let solvedList = encryptedList
  --putStr "Caesar 26: "
  --print(solvedList)

  mapM print(take maxShift $ iterate (map (solve)) [solvedList])
  --print(repeatNtimes maxShift solve solvedList)
  --print(solve encryptedList)
```

4.4 Output

```
Original Strings:
IBM
Hello World
This is a test
Hi my name is Robbie
WandaVision
Abed
The Mandalorian
Wow I learned Haskell I think

Encrypted Strings:
HAL
Gdkkn Vnqkc
Sghr hr z sdrs
Gh lx mzld hr Qnaahd
VzmczUhrhnm
Zadc
Sgd Lzmczknqhzm
```

```
Vnv H kdzqmdc Gzrjdkk H sghmj

Decrypted Strings:
IBM
Hello World
This is a test
Hi my name is Robbie
WandaVision
Abed
The Mandalorian
Wow I learned Haskell I think

Solve:
[["HAL","Gdkkn Vnqkc","Sghr hr z sdrs","Gh lx mzld hr Qnaahd","VzmczUhrhnm","Zadc","Sgd Lzmczknqhzm","Vnv H
[["GZK","Fcjjm Umpjb","Rfgq gq y rcqr","Fg kw lykc gq Pmzzgc","UylbyTgqgml","Yzcb","Rfc Kylbyjmpgyl","Umu G
[["FYJ","Ebiil Tloia","Qefp fp x qbpq","Ef jv kxjb fp Olyyfb","TxkaxSfpflk","Xyba","Qeb Jxkaxilofxk","Tlt F
[["EXI","Dahhk Sknhz","Pdeo eo w paop","De iu jwia eo Nkxxea","SwjzwReoekj","Wxaz","Pda Iwjzwhknewj","Sks E
[["DWH","Czggj Rjmgy","Ocdn dn v ozno","Cd ht ivhz dn Mjwwdz","RviyvQdndji","Vwzy","Ocz Hviyvgjmdvi","Rjr D
[["CVG","Byffi Qilfx","Nbcm cm u nymn","Bc gs hugy cm Livvcy","QuhxuPcmcih","Uvyx","Nby Guhxufilcuh","Qiq C
[["BUF","Axeeh Phkew","Mabl bl t mxlm","Ab fr gtfx bl Khuubx","PtgwtOblbhg","Tuxw","Max Ftgwtehkbtg","Php E
[["ATE","Zwddg Ogjdv","Lzak ak s lwkl","Za eq fsew ak Jgttaw","OsfvsNakagf","Stwv","Lzw Esfvsdgjasf","Ogo A
[["ZSD","Yvccf Nficu","Kyzj zj r kvjk","Yz dp erdv zj Ifsszv","NreurMzjzfe","Rsvu","Kyv Dreurcfizre","Nfn Z
[["YRC","Xubbe Mehbt","Jxyi yi q juij","Xy co dqcu yi Herryu","MqdtqLyiyed","Qrut","Jxu Cqdtqbehyqd","Mem Y
[["XQB","Wtaad Ldgas","Iwxh xh p ithi","Wx bn cpbt xh Gdqqxt","LpcspKxhxdc","Pqts","Iwt Bpcspadgxpc","Ldl X
[["WPA","Vszzc Kcfzr","Hvwg wg o hsgh","Vw am boas wg Fcppws","KobroJwgwcb","Opsr","Hvs Aobrozcfwob","Kck W
[["VOZ","Uryyb Jbeyq","Guvf vf n grfg","Uv zl anzr vf Eboovr","JnaqnIvfvba","Norq","Gur Znaqnybevna","Jbj V
[["UNY","Tqxxa Iadxp","Ftue ue m fqef","Tu yk zmyq ue Dannuq","ImzpmHueuaz","Mnqp","Ftq Ymzpmxadumz","Iai U
[["TMX","Spwwz Hzcwo","Estd td l epde","St xj ylxp td Czmmtp","HlyolGtdtzy","Lmpo","Esp Xlyolwzctly","Hzh T
[["SLW","Rovvy Gybvn","Drsc sc k docd","Rs wi xkwo sc Byllso","GkxnkFscsyx","Klon","Dro Wkxnkvybskx","Gyg S
[["RKV","Qnuux Fxaum","Cqrb rb j cnbc","Qr vh wjvn rb Axkkrn","FjwmjErbrxw","Jknm","Cqn Vjwmjuxarjw","Fxf F
[["QJU","Pmttw Ewztl","Bpqa qa i bmab","Pq ug vium qa Zwjjqm","EivliDqaqwv","Ijml","Bpm Uivlitwzqiv","Ewe Q
[["PIT","Olssv Dvysk","Aopz pz h alza","Op tf uhtl pz Yviipl","DhukhCpzpvu","Hilk","Aol Thukhsvyphu","Dvd P
[["OHS","Nkrru Cuxrj","Znoy oy g zkyz","No se tgsk oy Xuhhok","CgtjgBoyout","Ghkj","Znk Sgtjgruxogt","Cuc C
[["NGR","Mjqqt Btwqi","Ymnx nx f yjxy","Mn rd sfrj nx Wtggnj","BfsifAnxnts","Fgji","Ymj Rfsifqtwnfs","Btb N
[["MFQ","Lipps Asvph","Xlmw mw e xiwx","Lm qc reqi mw Vsffmi","AerheZmwmsr","Efih","Xli Qerhepsvmer","Asa M
[["LEP","Khoor Zruog","Wklv lv d whvw","Kl pb qdph lv Ureelh","ZdqgdYlvlrq","Dehg","Wkh Pdqgdoruldq","Zrz L
[["KDO","Jgnnq Yqtnf","Vjku ku c vguv","Jk oa pcog ku Tqddkg","YcpfcXkukqp","Cdgf","Vjg Ocpfcnqtkcp","Yqy K
[["JCN","Ifmmp Xpsme","Uijt jt b uftu","Ij nz obnf jt Spccjf","XboebWjtjpo","Bcfe","Uif Nboebmpsjbo","Xpx J
[["IBM","Hello World","This is a test","Hi my name is Robbie","WandaVision","Abed","The Mandalorian","Wow I
[["HAL","Gdkkn Vnqkc","Sghr hr z sdrs","Gh lx mzld hr Qnaahd","VzmczUhrhnm","Zadc","Sgd Lzmczknqhzm","Vnv H
```

5 Scala

5.1 Consulting Log

Expected hours needed: 6

Date	***Hours Spent***	***Tasks / Accomplishments / Issues / Thoughts***
04/23/2021	2	Was able to get the encrypt and decrypt functions without any loops and only recursion relatively quickly, which I was surprised about. The only issue is getting the numbers with solve to work correctly, which is annoying only because I have to pass a lot of variables in the recursion. It was much easier with loops. It also is only currently running with the first test string, because it does not loop back to the beginning yet to do the next one.
04/24/2021	1.5	I got the solve function working after going through a few logical errors and adding some parameters to the function. Next I tried looping back to the beginning with a parameter to check if it's the last string. It seems to be working well for the most part, I see some errors with the decrypt function that did not happen in the first string, but I am stepping away at the moment and will come back to it. P.S. I came back to it and it took me two minutes, easy fix.

5.2 Commentary

Using Scala in a functional manner was not too difficult compared to the procedural way. Especially because I already had the basis of the code figured out. I had more lines of code in the end, which I was kind of surprised about. There might have been a way to more efficiently write my code, but I do not know if there is a better way I could write it.

5.3 Source Code

```
object Main {
         def main(args: Array[String]) {
        //Procedural
        println("Caesar Cipher\n")
        val testStr1 = "IBM"
        val testStr2 = "Hello World"
        val testStr3 = "This is a test"
        val testStr4 = "Hi my name is Robbie"
        val testStr5 = "WandaVision"
        val testStr6 = "Abed"
        val testStr7 = "The Mandalorian"
        val testStr8 = "Wow I learned Scala I think"

        val shiftAmount = -1

            var index = 0
            var e = 0

            var testStrArr = Array(testStr1, testStr2, testStr3, testStr4, testStr5, testStr6, testStr7, te

            encrypt(testStrArr, shiftAmount, index, e)
         }
        def encrypt(testStrArr:Array[String], shift:Int, index:Int, e:Int) : String = {
        if(e == 0) {
                testStrArr(index) = testStrArr(index).toUpperCase()
                println("Original String: " + testStrArr(index))
        }
        var encryptedStr = testStrArr(index)
```

```
        var encryptedChrArr = encryptedStr.toCharArray()

        var letter = encryptedStr(e)
        var asciiVal = letter.toInt
        //println(letter) //check letters
        if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
            if(letter.toInt == 65 && shift < 0) { //If A and shift is negative, then loop around to Z
                encryptedChrArr(e) = (asciiVal + 26 + shift).toChar
            } else if(letter.toInt == 90 && shift > 0) { //If Z, and shift is positive, then loop around to
                encryptedChrArr(e) = (asciiVal - 26 + shift).toChar
            } else { // Else perform a normal shift
                encryptedChrArr(e) = (asciiVal + shift).toChar
            }
        }
        else { //Other Characters
            //Other Characters should not be changed in Caesar Cipher
            encryptedChrArr(e) = letter
            //println(letter)
        }
        encryptedStr = encryptedChrArr.mkString("")
        testStrArr(index) = encryptedStr

        if(e == (encryptedStr.length - 1)) {
                println("Encrypted String: " + encryptedStr)
                var d = 0
                var s = 0
                val maxShiftAmount = 26
                //Move to Decrypt
                decrypt(testStrArr, shift, index, d)
                //Move to Solve
                println("Solve: ")
                solve(testStrArr, encryptedStr, maxShiftAmount, index, s, maxShiftAmount)
        } else {
                encrypt(testStrArr, shift, index, (e + 1))
        }
    }
    def decrypt(testStrArr:Array[String], shift:Int, index:Int, d:Int) : String = {
        var decryptedStr = testStrArr(index)
        var decryptedChrArr = decryptedStr.toCharArray()

        var letter = decryptedStr(d)
        var asciiVal = letter.toInt
        //println(letter) //check letters
        if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
            if(letter.toInt == 65 && -shift < 0) { //If A and shift is negative, then loop around to Z
                decryptedChrArr(d) = (asciiVal + 26 - shift).toChar
            } else if(letter.toInt == 90 && -shift > 0) { //If Z, and shift is positive, then loop around t
                decryptedChrArr(d) = (asciiVal - 26 - shift).toChar
            } else { // Else perform a normal shift
                decryptedChrArr(d) = (asciiVal - shift).toChar
            }
        }
        else { //Other Characters
            //Other Characters should not be changed in Caesar Cipher
            decryptedChrArr(d) = letter
            //println(letter)
        }
        decryptedStr = decryptedChrArr.mkString("")
        testStrArr(index) = decryptedStr

        if(d == (decryptedStr.length - 1)) {
                println("Decrypted String: " + decryptedStr)
                return "Decrypt"
        } else {
                decrypt(testStrArr, shift, index, (d + 1))
```

```
        }
    }
    def solve(testStrArr:Array[String], testStr:String, maxShift:Int, index:Int, x:Int, caesarNum:Int) : St
        var s = x
        var solveNum = caesarNum

        var solvedStr = testStr
        var solvedChrArr = solvedStr.toCharArray()
        if(solveNum == 26) {
                println("\tCaesar " + solveNum + ": " + solvedStr)
                solveNum -= 1
        }

        val shift = -1
        var tempMaxShift = 1

        var letter = solvedStr(s)
        var asciiVal = letter.toInt
        //println(letter) //check letters
        if(letter.toInt >= 65 && letter.toInt <= 90) { //A to Z
            if(letter.toInt == 65 && shift < 0) { //If A and shift is negative, then loop around to Z
                solvedChrArr(s) = (asciiVal + 26 + shift).toChar
            } else if(letter.toInt == 90 && shift > 0) { //If Z, and shift is positive, then loop around to
                solvedChrArr(s) = (asciiVal - 26 + shift).toChar
            } else { // Else perform a normal shift
                solvedChrArr(s) = (asciiVal + shift).toChar
            }
        }
        else { //Other Characters
            //Other Characters should not be changed in Caesar Cipher
            solvedChrArr(s) = letter
            //println(letter)
        }
        solvedStr = solvedChrArr.mkString("")
        testStrArr(index) = solvedStr

        if((solveNum == 0) && (s == solvedStr.length - 1)) {
                println("\tCaesar " + solveNum + ": " + solvedStr)
                var e = 0
                if(index != (testStrArr.length - 1)) {
                        println()
                        encrypt(testStrArr, shift, (index + 1), e)
                } else {
                        return "Solve"
                }
        } else {
                if(s == (solvedStr.length - 1)) {
                        println("\tCaesar " + solveNum + ": " + solvedStr)
                        //tempMaxShift += 1
                        s = -1
                        solveNum -= 1
                        solve(testStrArr, solvedStr, (tempMaxShift + 1), index, (s + 1), solveNum)
                } else {
                        solve(testStrArr, solvedStr, (tempMaxShift + 1), index, (s + 1), solveNum)
                }
        }
    }
}
```

5.4 Output

```
Caesar Cipher

```

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

Original String: HELLO WORLD
Encrypted String: GDKKN VNQKC
Decrypted String: HELLO WORLD
Solve:
        Caesar 26: GDKKN VNQKC
        Caesar 25: FCJJM UMPJB
        Caesar 24: EBIIL TLOIA
        Caesar 23: DAHHK SKNHZ
        Caesar 22: CZGGJ RJMGY
        Caesar 21: BYFFI QILFX
        Caesar 20: AXEEH PHKEW
        Caesar 19: ZWDDG OGJDV
        Caesar 18: YVCCF NFICU
        Caesar 17: XUBBE MEHBT
        Caesar 16: WTAAD LDGAS
        Caesar 15: VSZZC KCFZR
        Caesar 14: URYYB JBEYQ
        Caesar 13: TQXXA IADXP
        Caesar 12: SPWWZ HZCWO
        Caesar 11: ROVVY GYBVN
        Caesar 10: QNUUX FXAUM
        Caesar 9: PMTTW EWZTL
        Caesar 8: OLSSV DVYSK
        Caesar 7: NKRRU CUXRJ
        Caesar 6: MJQQT BTWQI
        Caesar 5: LIPPS ASVPH
        Caesar 4: KHOOR ZRUOG
        Caesar 3: JGNNQ YQTNF
        Caesar 2: IFMMP XPSME
        Caesar 1: HELLO WORLD
        Caesar 0: GDKKN VNQKC

Original String: THIS IS A TEST
```

```
Encrypted String: SGHR HR Z SDRS
Decrypted String: THIS IS A TEST
Solve:
        Caesar 26: SGHR HR Z SDRS
        Caesar 25: RFGQ GQ Y RCQR
        Caesar 24: QEFP FP X QBPQ
        Caesar 23: PDEO EO W PAOP
        Caesar 22: OCDN DN V OZNO
        Caesar 21: NBCM CM U NYMN
        Caesar 20: MABL BL T MXLM
        Caesar 19: LZAK AK S LWKL
        Caesar 18: KYZJ ZJ R KVJK
        Caesar 17: JXYI YI Q JUIJ
        Caesar 16: IWXH XH P ITHI
        Caesar 15: HVWG WG O HSGH
        Caesar 14: GUVF VF N GRFG
        Caesar 13: FTUE UE M FQEF
        Caesar 12: ESTD TD L EPDE
        Caesar 11: DRSC SC K DOCD
        Caesar 10: CQRB RB J CNBC
        Caesar 9: BPQA QA I BMAB
        Caesar 8: AOPZ PZ H ALZA
        Caesar 7: ZNOY OY G ZKYZ
        Caesar 6: YMNX NX F YJXY
        Caesar 5: XLMW MW E XIWX
        Caesar 4: WKLV LV D WHVW
        Caesar 3: VJKU KU C VGUV
        Caesar 2: UIJT JT B UFTU
        Caesar 1: THIS IS A TEST
        Caesar 0: SGHR HR Z SDRS

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

Original String: WANDAVISION
Encrypted String: VZMCZUHRHNM
```

```
Decrypted String: WANDAVISION
Solve:
        Caesar 26: VZMCZUHRHNM
        Caesar 25: UYLBYTGQGML
        Caesar 24: TXKAXSFPFLK
        Caesar 23: SWJZWREOEKJ
        Caesar 22: RVIYVQDNDJI
        Caesar 21: QUHXUPCMCIH
        Caesar 20: PTGWTOBLBHG
        Caesar 19: OSFVSNAKAGF
        Caesar 18: NREURMZJZFE
        Caesar 17: MQDTQLYIYED
        Caesar 16: LPCSPKXHXDC
        Caesar 15: KOBROJWGWCB
        Caesar 14: JNAQNIVFVBA
        Caesar 13: IMZPMHUEUAZ
        Caesar 12: HLYOLGTDTZY
        Caesar 11: GKXNKFSCSYX
        Caesar 10: FJWMJERBRXW
        Caesar 9: EIVLIDQAQWV
        Caesar 8: DHUKHCPZPVU
        Caesar 7: CGTJGBOYOUT
        Caesar 6: BFSIFANXNTS
        Caesar 5: AERHEZMWMSR
        Caesar 4: ZDQGDYLVLRQ
        Caesar 3: YCPFCXKUKQP
        Caesar 2: XBOEBWJTJPO
        Caesar 1: WANDAVISION
        Caesar 0: VZMCZUHRHNM

Original String: ABED
Encrypted String: ZADC
Decrypted String: ABED
Solve:
        Caesar 26: ZADC
        Caesar 25: YZCB
        Caesar 24: XYBA
        Caesar 23: WXAZ
        Caesar 22: VWZY
        Caesar 21: UVYX
        Caesar 20: TUXW
        Caesar 19: STWV
        Caesar 18: RSVU
        Caesar 17: QRUT
        Caesar 16: PQTS
        Caesar 15: OPSR
        Caesar 14: NORQ
        Caesar 13: MNQP
        Caesar 12: LMPO
        Caesar 11: KLON
        Caesar 10: JKNM
        Caesar 9: IJML
        Caesar 8: HILK
        Caesar 7: GHKJ
        Caesar 6: FGJI
        Caesar 5: EFIH
        Caesar 4: DEHG
        Caesar 3: CDGF
        Caesar 2: BCFE
        Caesar 1: ABED
        Caesar 0: ZADC

Original String: THE MANDALORIAN
Encrypted String: SGD LZMCZKNQHZM
Decrypted String: THE MANDALORIAN
```

```
Solve:
        Caesar 26: SGD LZMCZKNQHZM
        Caesar 25: RFC KYLBYJMPGYL
        Caesar 24: QEB JXKAXILOFXK
        Caesar 23: PDA IWJZWHKNEWJ
        Caesar 22: OCZ HVIYVGJMDVI
        Caesar 21: NBY GUHXUFILCUH
        Caesar 20: MAX FTGWTEHKBTG
        Caesar 19: LZW ESFVSDGJASF
        Caesar 18: KYV DREURCFIZRE
        Caesar 17: JXU CQDTQBEHYQD
        Caesar 16: IWT BPCSPADGXPC
        Caesar 15: HVS AOBROZCFWOB
        Caesar 14: GUR ZNAQNYBEVNA
        Caesar 13: FTQ YMZPMXADUMZ
        Caesar 12: ESP XLYOLWZCTLY
        Caesar 11: DRO WKXNKVYBSKX
        Caesar 10: CQN VJWMJUXARJW
        Caesar 9: BPM UIVLITWZQIV
        Caesar 8: AOL THUKHSVYPHU
        Caesar 7: ZNK SGTJGRUXOGT
        Caesar 6: YMJ RFSIFQTWNFS
        Caesar 5: XLI QERHEPSVMER
        Caesar 4: WKH PDQGDORULDQ
        Caesar 3: VJG OCPFCNQTKCP
        Caesar 2: UIF NBOEBMPSJBO
        Caesar 1: THE MANDALORIAN
        Caesar 0: SGD LZMCZKNQHZM

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

6 Main Google Searches

6.1 LISP

https://www.tutorialspoint.com/lisp/index.htm https://www.youtube.com/watch?v=ymSq4wHrqyU https://www.gnu.org/software/emacs/manual/html_node/elisp/Building-Lists.html https://jtra.cz/stuff/lisp/sclr/map.html https://stackoverflow.com/questions/22522108/how-to-map-a-function-in-common-lisp https://rosettacode.org/wiki/Caesar_cipher#Common_Lisp

6.2 F#

https://www.tutorialspoint.com/fsharp/index.htm https://www.youtube.com/watch?v=c7eNDJN758U https://stackoverflow.com/questions/5752020/caesar-cipher-in-f https://docs.microsoft.com/en-us/dotnet/fsharp/get-started/get-started-vscode https://stackoverflow.com/questions/43912326/how-to-write-own-list-map-function-in-f https://stackoverflow.com/questions/35147514/does-f-has-a-funct https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/conditional-expressions-if-then-else https://stackoverflow.com/questions/28769658/f-applying-function-to-its-result-n-times

6.3 Erlang

https://www.tutorialspoint.com/erlang/index.htm
https://www.youtube.com/watch?v=IEhwc2q1zG4
https://learnyousomeerlang.com/contents
https://stackoverflow.com/questions/1549364/how-to-use-erlang-listsmap-function
http://erlang.org/documentation/doc-5.9/doc/programming_examples/funs.html
https://stackoverflow.com/questions/32659084/print-each-element-from-a-list-in-erlang/32659829
https://stackoverflow.com/questions/1549364/how-to-use-erlang-listsmap-function
https://erlang.org/doc/man/io.html
https://rosettacode.org/wiki/Caesar_cipher

6.4 Haskell

https://www.tutorialspoint.com/haskell/haskell_if_else_statement.htm
https://www.youtube.com/watch?v=02_H3LjqMr8
https://stackoverflow.com/questions/7423123/how-to-call-the-same-function-n-times/7423199
https://hackage.haskell.org/package/base-4.15.0.0/docs/GHC-List.html#v:iterate-39-
https://stackoverflow.com/questions/3911060/library-function-to-compose-a-function-with-itself-n-times
https://programming-idioms.org/idiom/12/check-if-list-contains-a-value/800/haskell
https://stackoverflow.com/questions/51073535/using-map-with-function-that-has-multiple-arguments
https://stackoverflow.com/questions/7423123/how-to-call-the-same-function-n-times/7423199

6.5 Scala

https://www.tutorialspoint.com/scala/index.htm
https://medium.com/@LiliOuakninFelsen/functional-vs-object-oriented-vs-procedural-programming-a3d458555~:text=Procedural%20programming%20(PP)%2C%20also,relies%20on%20procedures%20or%20routines.&text=Functional%20programming%20(FP)%20is%20about,function%20to%20get%20a%20result.

7 Language Rankings

1. Haskell
2. Erlang
3. F#
4. Scala
5. LISP