Marist College
MS in Computer Science
School of Computer Science and Mathematics

CMPT-331L-111-23s Theory of Programming Languages
Alan Labouseur
Spring 2023

Assignment 2: Functional Programming

Connor H. Johnson
connor.johnson1@marist.edu

Table of Context

Assignment 1: Programming In The Past

Connor H. Johnson

April 21, 2023

Overview

Within this document, I will describe step-by-step the method I took to complete the assignment of coding the Caesar Cipher functionally with 5 different programming languages: CLISP, F#, ERLANG, JavaScript, and Scala.
Each language should contain five functions within the Caesar Cipher code(with an exception to F# for the last two functions): Encrypt(), Decrypt(), and Solve(), positiveShiftValueRecursion(), and negativeShiftValueRecursion(). The Encrypt function will take in the 'mainString' variable containing the message, and a 'shiftAmount' variable that will shift the appropriate letter to said shifted amount through the alphabet. The Decrypt function will take the same arguments but decrypt the encrypted word, meaning the original 'mainString' will print. The Solve function will take the 'mainString' variable and a 'maxShiftValue' variable. Within the Solve function, this shall iterate a 'maxShiftValue' amount of times through the cipher to display all possible encrypted or decrypted words for their shift value. If the 'maxShiftValue' is positive, the solve function should call the positiveShiftValueRecursion() to use the encrypted function to iterate recursively. If the 'maxShiftValue' is negative, then the function shall call the ShiftValueRecursion() to use the decrypted function to iterate.
With each language, I will be providing the code I used to complete the assignment in addition to multiple test cases while also explaining my accomplishments, issues, and thoughts throughout the project.

CLSIP, F#, and SCALA were worked on and tested on the multi-language online IDE: JDOODLE
ERLANG was worked on and tested on the multi-language online IDE: Tutorialspoint
JavaScript was worked on and tested on the multi-language online IDE: Programiz

---

Functional Steps Used for Each Programming Language

1.) Learn the basic syntax and print a string variable with an integer variable in the main().

a.) Since many of these procedural programming languages declare their variables very differently, for example,

CLSIP:

```
(defvar str "hal") ;
```

ERLANG:

```
Str = "HAL",
```

A great way to start would be researching how to declare and print variables on the same line. This was not a difficult step but a good starting point to ease me into a new language.

2.) Learn how to create a function that takes the 'mainString' and 'shiftAmount' variables.

a.) Once I learned how to create a string and integer, I was tasked with creating a function that would take the arguments of 'mainString' and 'shiftAmount.'

b.) I aimed to print out those variables to ensure the main() and the newly created function could communicate.

c.) If the variables were printed out correctly, my next step was to research how to return a variable inside the function and print out the returned variable in the main(). This step also helped me understand how to call functions from other functions, which will help me down the line when completing my Solve() function.

3.) Understand the map or case syntax.

a.) Understanding the map and case syntax was the most confusing to me while completing this project. I am unfamiliar with mapping, and I typically try a different method when implementing a case. With this being said, most of my time was spent learning this syntax, and once this was complete, I could implement everything else quickly. After learning about map and case syntax, it has shown to be an efficient way to perform a standard operation on a sequence of data.

4.) Understand how to change each character into its ASCII value.

a.) Figuring out how to change characters into their ASCII value was a more manageable task than I thought, but most solutions were found via a StackOverflow forum.

F#

```
let asciiValues = str |> Seq.map int
```

5.) Implement the equation used in the past programming assignment.

a.) The trouble with this step was not the implementation of the equation, as it works perfectly fine. The complication was figuring out the math syntax. With this being said, the most interesting syntax I came across was in CLISP:

```
(if (and (>= code 65) (<= code 90))
    (code-char (+ 65 (mod (- (- code 65) shiftAmount) 26))) ;
```

6.) Learn concatenation in the respective language.

a.) Concatenation was an exciting task to find in all the languages. Last time, the easiest way to concatenate was with a '+=' operator. Now I must find the concatenation syntax for functional programming like

CLISP:	JavaScript	F#
`'String`	`.join("")`	`System.String(decrypted |> Seq.toArray)`

7.) Understand the recursion for the solve().

a.) Since the completion of Assignment 1, the solve() function needed some re-working so that I could complete it without a loop. My plan of attack was to have a condition looking to see if maxShiftAmount was positive or negative. Depending on what statement was called, that would call another function named positiveShiftValueRecursion() or negativeShiftValueRecursion().

b.) These following functions would then call their respective cipher function. However, the arguments inside of this call would either be Str, maxShiftAmount, or -maxShiftAmount. This -maxShiftAmount negates using a encrypt function inside negativeShiftValueRecursion() to work correctly.

c.) After a print statement and a function call with the parameters back to positiveShiftValueRecursion(maxShiftAmount - 1) or negativeShiftValueRecursion(maxShiftAmount + 1) to cause recursion.

Introduction to the Caesar Cipher

To begin this assignment, I was familiar with the Ceaser Cipher due to my past work on the assignment before. Using past resources to figure out and lay a foundation for me took an edge off some things now that I knew my coding structure would be capable of completing the task. The only difference between Assignment 1 and Assignment 2 was the use of functional programming; however, I took it upon myself to learn how to implement recursion while doing the assignment rather than programming it in Python first, which helped me keep my mind on the relevant work and figure out functional programming while I complete the task.

Functional Programming Language Code/Test Cases

CLISP

```
(defun encrypt (str shiftAmount)

  (coerce
   (map 'list ; Applies the function to each element of the list
        (lambda (char) ; takes a single argument char
          (let ((code (char-code char))) ; Assigns the ASCII code of char to the variable code
            (if (and (>= code 65) (<= code 90)) ; Checks if char is an uppercase letter
                (code-char (+ 65 (mod (+ (- code 65) shiftAmount) 26))) ; Shifts the letter by the
given amount and returns the resulting character
                char))) ; Returns the original character if it is not an uppercase letter
        str) ; Applies the function to the input string
   'string); Converts characters to a string

)

(defun decrypt (str shiftAmount)

  (coerce
   (map 'list ; Applies the function to each element of the list
        (lambda (char) ; takes a single argument char
          (let ((code (char-code char))) ; Assigns the ASCII code of char to the variable code
            (if (and (>= code 65) (<= code 90)) ; Checks if char is an uppercase letter
                (code-char (+ 65 (mod (- (- code 65) shiftAmount) 26))) ; Shifts the letter by the
given amount and returns the resulting character
                char))) ; Returns the original character if it is not an uppercase letter
        str) ; Applies the function to the input string
   'string); Converts characters to a string

)

(defun solve (str maxShiftAmount)
  (if (>= maxShiftAmount 0)
      (positiveShiftValueRecursion str maxShiftAmount) ; if maxShiftAmount is greater than 0
      (negativeShiftValueRecursion str maxShiftAmount))) ; if maxShiftAmount is less than 0




(defun positiveShiftValueRecursion (str maxShiftAmount)
```

```
  (if (>= maxShiftAmount 0)
        (progn ;used for grouping together characters
            (setf solveEncrypt (encrypt str maxShiftAmount))
            (write-line (format nil "Ceaser ~a: ~a" maxShiftAmount solveEncrypt))
            (positiveShiftValueRecursion str (- maxShiftAmount 1))
        )

    )

)

(defun negativeShiftValueRecursion (str maxShiftValue)
  (if (<= maxShiftValue 0)
        (progn
            (setf solveDecrypt (decrypt str (- maxShiftValue))) ; you must have the - maxShiftValue
for it to properly decrypt (me and chatgpt solution)
            (write-line (format nil "Ceaser ~a: ~a" maxShiftValue solveDecrypt))
            (negativeShiftValueRecursion str (+ maxShiftValue 1))
        )
    )
)




(defvar str "hal") ; Assigns the input string
(defvar shiftAmount 3) ; Assigns the shift amount
(defvar maxShiftAmount -26) ; Assigns the maxShiftAmount for solve

(setf str (string-upcase str)) ; Converts the input string to uppercase

(setf encrypted (encrypt str shiftAmount)) ; Encrypts the string using the given shift amount
(write-line (format nil "Encrypted string: ~a" encrypted))

(setf decrypted (decrypt encrypted shiftAmount)); Decrypts the string using the given shift amount
(write-line (format nil "Decrypted string: ~a" decrypted))

(write-line (format nil "Solve: "))
(solve str maxShiftAmount)
```

Test Cases (CLSIP)

(defvar str "hal") (defvar shiftAmount 3) (defvar maxShiftAmount -26)	(defvar str "&Google Docs#) (defvar shiftAmount -3) (defvar maxShiftAmount -30)	(defvar str "Z E L") (defvar shiftAmount 27) (defvar maxShiftAmount 26)
Encrypted string: KDO Decrypted string: HAL Solve: Ceaser -26: HAL Ceaser -25: IBM Ceaser -24: JCN Ceaser -23: KDO Ceaser -22: LEP Ceaser -21: MFQ Ceaser -20: NGR Ceaser -19: OHS Ceaser -18: PIT Ceaser -17: QJU Ceaser -16: RKV Ceaser -15: SLW Ceaser -14: TMX Ceaser -13: UNY Ceaser -12: VOZ Ceaser -11: WPA Ceaser -10: XQB Ceaser -9: YRC Ceaser -8: ZSD Ceaser -7: ATE Ceaser -6: BUF Ceaser -5: CVG Ceaser -4: DWH Ceaser -3: EXI Ceaser -2: FYJ Ceaser -1: GZK Ceaser 0: HAL	Encrypted string: &DLLDIB ALZP# Decrypted string: &GOOGLE DOCS# Solve: Ceaser -30: &CKKCHA ZKYO# Ceaser -29: &DLLDIB ALZP# Ceaser -28: &EMMEJC BMAQ# Ceaser -27: &FNNFKD CNBR# Ceaser -26: &GOOGLE DOCS# Ceaser -25: &HPPHMF EPDT# Ceaser -24: &IQQING FQEU# Ceaser -23: &JRRJOH GRFV# Ceaser -22: &KSSKPI HSGW# Ceaser -21: &LTTLQJ ITHX# Ceaser -20: &MUUMRK JUIY# Ceaser -19: &NVVNSL KVJZ# Ceaser -18: &OWWOTM LWKA# Ceaser -17: &PXXPUN MXLB# Ceaser -16: &QYYQVO NYMC# Ceaser -15: &RZZRWP OZND# Ceaser -14: &SAASXQ PAOE# Ceaser -13: &TBBTYR QBPF# Ceaser -12: &UCCUZS RCQG# Ceaser -11: &VDDVAT SDRH# Ceaser -10: &WEEWBU TESI# Ceaser -9: &XFFXCV UFTJ# Ceaser -8: &YGGYDW VGUK# Ceaser -7: &ZHHZEX WHVL# Ceaser -6: &AIIAFY XIWM# Ceaser -5: &BJJBGZ YJXN# Ceaser -4: &CKKCHA ZKYO# Ceaser -3: &DLLDIB ALZP# Ceaser -2: &EMMEJC BMAQ# Ceaser -1: &FNNFKD CNBR# Ceaser 0: &GOOGLE DOCS#	Encrypted string: A F M Decrypted string: Z E L Solve: Ceaser 26: Z E L Ceaser 25: Y D K Ceaser 24: X C J Ceaser 23: W B I Ceaser 22: V A H Ceaser 21: U Z G Ceaser 20: T Y F Ceaser 19: S X E Ceaser 18: R W D Ceaser 17: Q V C Ceaser 16: P U B Ceaser 15: O T A Ceaser 14: N S Z Ceaser 13: M R Y Ceaser 12: L Q X Ceaser 11: K P W Ceaser 10: J O V Ceaser 9: I N U Ceaser 8: H M T Ceaser 7: G L S Ceaser 6: F K R Ceaser 5: E J Q Ceaser 4: D I P Ceaser 3: C H O Ceaser 2: B G N Ceaser 1: A F M Ceaser 0: Z E L

F#

```
let encrypt (str: string) (shiftAmount: int) : string =
    let asciiValues = str |> Seq.map int

    let encrypted = asciiValues |> Seq.map (fun character ->
        let shifted = (((character - 65 + shiftAmount) % 26 + 26) % 26 + 65)
        char shifted
    )

    let encryptedString = System.String(encrypted |> Seq.toArray)
    encryptedString

let decrypt (str: string) (shiftAmount: int) : string =
    let asciiValues = str |> Seq.map int

    let decrypted = asciiValues |> Seq.map (fun character ->
        let shifted = (((character - 65 - shiftAmount) % 26 + 26) % 26 + 65)
        char shifted
    )

    let decryptedString = System.String(decrypted |> Seq.toArray)
    decryptedString


let rec solve (str: string) (maxShiftAmount: int) : unit =
    if maxShiftAmount > 0 then
        let solveEncrypt = encrypt str maxShiftAmount
        printfn "Caesar %d: %s" maxShiftAmount solveEncrypt
        solve str (maxShiftAmount-1)
    else if maxShiftAmount < 0 then
        let solveDecrypt = decrypt str -maxShiftAmount
        printfn "Caesar %d: %s" maxShiftAmount solveDecrypt
        solve str (maxShiftAmount+1)


let lowerstr = "hal"
let shiftAmount = 3
let maxShiftAmount = -26

let str = lowerstr.ToUpper()

let encrypted = encrypt str shiftAmount
printfn "Encrypted: %s" encrypted
```

```
let decrypted = decrypt encrypted shiftAmount
printfn "Decrypted: %s" decrypted

printfn "Solved:"
solve str maxShiftAmount
```

Test Cases (F#)

let lowerstr = "hal" let shiftAmount = 3 let maxShiftAmount = -26	let str = 'CEASER' let shiftAmount = 30 let maxShiftValue = 35	let = suit let shiftAmount = -5 let maxShiftValue = -30
Encrypted: KDO Decrypted: HAL Solved: Caesar -26: HAL Caesar -25: IBM Caesar -24: JCN Caesar -23: KDO Caesar -22: LEP Caesar -21: MFQ Caesar -20: NGR Caesar -19: OHS Caesar -18: PIT Caesar -17: QJU Caesar -16: RKV Caesar -15: SLW Caesar -14: TMX Caesar -13: UNY Caesar -12: VOZ Caesar -11: WPA Caesar -10: XQB Caesar -9: YRC Caesar -8: ZSD Caesar -7: ATE Caesar -6: BUF Caesar -5: CVG Caesar -4: DWH Caesar -3: EXI Caesar -2: FYJ	Encrypted: GIEWIV Decrypted: CEASER Solved: Caesar 35: LNJBNA Caesar 34: KMIAMZ Caesar 33: JLHZLY Caesar 32: IKGYKX Caesar 31: HJFXJW Caesar 30: GIEWIV Caesar 29: FHDVHU Caesar 28: EGCUGT Caesar 27: DFBTFS Caesar 26: CEASER Caesar 25: BDZRDQ Caesar 24: ACYQCP Caesar 23: ZBXPBO Caesar 22: YAWOAN Caesar 21: XZVNZM Caesar 20: WYUMYL Caesar 19: VXTLXK Caesar 18: UWSKWJ Caesar 17: TVRJVI Caesar 16: SUQIUH Caesar 15: RTPHTG Caesar 14: QSOGSF Caesar 13: PRNFRE Caesar 12: OQMEQD Caesar 11: NPLDPC	Encrypted: NPDO Decrypted: SUIT Solved: Caesar -30: OQEP Caesar -29: PRFQ Caesar -28: QSGR Caesar -27: RTHS Caesar -26: SUIT Caesar -25: TVJU Caesar -24: UWKV Caesar -23: VXLW Caesar -22: WYMX Caesar -21: XZNY Caesar -20: YAOZ Caesar -19: ZBPA Caesar -18: ACQB Caesar -17: BDRC Caesar -16: CESD Caesar -15: DFTE Caesar -14: EGUF Caesar -13: FHVG Caesar -12: GIWH Caesar -11: HJXI Caesar -10: IKYJ Caesar -9: JLZK Caesar -8: KMAL Caesar -7: LNBM Caesar -6: MOCN

Caesar -1: GZK	Caesar 10: MOKCOB Caesar 9: LNJBNA Caesar 8: KMIAMZ Caesar 7: JLHZLY Caesar 6: IKGYKX Caesar 5: HJFXJW Caesar 4: GIEWIV Caesar 3: FHDVHU Caesar 2: EGCUGT Caesar 1: DFBTFS	Caesar -5: NPDO Caesar -4: OQEP Caesar -3: PRFQ Caesar -2: QSGR Caesar -1: RTHS

ERLANG

```
-module(helloworld).
-export([encrypt/2, decrypt/2, solve/2, start/0]).

encrypt(Str, ShiftAmount) ->
    %Create a new list for the encryptChar values     %get each character in the str
    EncryptedStr = [encryptChar(Character, ShiftAmount) || Character <- Str],
    %return EncryptedStr
    EncryptedStr.

    %Takes each charcter found in the new list
encryptChar(Character, ShiftAmount) ->
    %Equation used from old ciphers
    ((Character - 65 + ShiftAmount) rem 26 + 26) rem 26 + 65.

decrypt(Str, ShiftAmount) ->
    %Create a new list for the encryptChar values     %get each character in the str
    DecryptedStr = [decryptedChar(Character, ShiftAmount) || Character <- Str],
    %return DecryptedStr
    DecryptedStr.

    %Takes each charcter found in the new list
decryptedChar(Character, ShiftAmount) ->
    %Equation used from old ciphers
    ((Character - 65 - ShiftAmount) rem 26 + 26) rem 26 + 65.

solve(Str, MaxShiftAmount ) ->
    %check to see where maxshiftvalue lies, debends on what function to use
    case MaxShiftAmount > 0 of
        true ->
            positiveShiftValueRecursion(Str, MaxShiftAmount);
        false ->
            negativeShiftValueRecursion(Str, MaxShiftAmount)
    end.

positiveShiftValueRecursion(Str, MaxShiftAmount) ->
    case MaxShiftAmount > 0 of
        true ->
            SolveEncrypt = encrypt(Str, MaxShiftAmount),
            io:fwrite("Ceaser ~w : ~s~n", [MaxShiftAmount, SolveEncrypt]),
            positiveShiftValueRecursion(Str, MaxShiftAmount - 1);
        false ->
            ok % similar to a none value, just notifiys when done with the recursion
    end.
```

```
negativeShiftValueRecursion(Str, MaxShiftAmount) ->
    case MaxShiftAmount < 0 of
        true ->
            SolveDecrypt = decrypt(Str, -MaxShiftAmount), % must have -MaxshiftValue to replecate
encryption
            io:fwrite("Ceaser ~w : ~s~n", [MaxShiftAmount, SolveDecrypt]),
            negativeShiftValueRecursion(Str, MaxShiftAmount + 1);
        false ->
            ok % similar to a none value, just notifiys when done with the recursion
    end.

start() ->
    Str = "HAL",
    ShiftAmount = 3,
    MaxShiftAmount = 26,

    %print statements make it so that the encruptedStr and decryptedStr are in String values '~s~n'
    EncryptedStr = encrypt(Str, ShiftAmount),
    io:fwrite("Encrypted : ~s~n", [EncryptedStr]),

    DecryptedStr = decrypt(EncryptedStr, ShiftAmount),
    io:fwrite("Decrypted : ~s~n", [DecryptedStr]),

    io:fwrite("Solve: \n"),
    solve(Str, MaxShiftAmount).
```

Test Cases (ERLANG)

Str = "HAL", ShiftAmount = 3, MaxShiftAmount = 26,	**Str = "ZNOTES", ShiftAmount = -3, MaxShiftAmount = 30,**	**Str = “TONY", ShiftAmount = 28, MaxShiftAmount = -30,**

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

```
Encrypted : WKLQBP
Decrypted : ZNOTES
Solve:
Ceaser 30 : DRSXIW
Ceaser 29 : CQRWHV
Ceaser 28 : BPQVGU
Ceaser 27 : AOPUFT
Ceaser 26 : ZNOTES
Ceaser 25 : YMNSDR
Ceaser 24 : XLMRCQ
Ceaser 23 : WKLQBP
Ceaser 22 : VJKPAO
Ceaser 21 : UIJOZN
Ceaser 20 : THINYM
Ceaser 19 : SGHMXL
Ceaser 18 : RFGLWK
Ceaser 17 : QEFKVJ
Ceaser 16 : PDEJUI
Ceaser 15 : OCDITH
Ceaser 14 : NBCHSG
Ceaser 13 : MABGRF
Ceaser 12 : LZAFQE
Ceaser 11 : KYZEPD
Ceaser 10 : JXYDOC
Ceaser 9 : IWXCNB
Ceaser 8 : HVWBMA
Ceaser 7 : GUVALZ
Ceaser 6 : FTUZKY
Ceaser 5 : ESTYJX
Ceaser 4 : DRSXIW
Ceaser 3 : CQRWHV
Ceaser 2 : BPQVGU
Ceaser 1 : AOPUFT
```

```
Encrypted : VQPA
Decrypted : TONY
Solve:
Ceaser -30 : PKJU
Ceaser -29 : QLKV
Ceaser -28 : RMLW
Ceaser -27 : SNMX
Ceaser -26 : TONY
Ceaser -25 : UPOZ
Ceaser -24 : VQPA
Ceaser -23 : WRQB
Ceaser -22 : XSRC
Ceaser -21 : YTSD
Ceaser -20 : ZUTE
Ceaser -19 : AVUF
Ceaser -18 : BWVG
Ceaser -17 : CXWH
Ceaser -16 : DYXI
Ceaser -15 : EZYJ
Ceaser -14 : FAZK
Ceaser -13 : GBAL
Ceaser -12 : HCBM
Ceaser -11 : IDCN
Ceaser -10 : JEDO
Ceaser -9 : KFEP
Ceaser -8 : LGFQ
Ceaser -7 : MHGR
Ceaser -6 : NIHS
Ceaser -5 : OJIT
Ceaser -4 : PKJU
Ceaser -3 : QLKV
Ceaser -2 : RMLW
Ceaser -1 : SNMX
```

JavaScript

```javascript
var str = "HAL";
var shiftAmount = 3;
var maxShiftAmount = 26;

let encrypted = encrypt(str, shiftAmount);
console.log("Encrypted:", encrypted)

let decrypted = decrypt(encrypted, shiftAmount);
console.log("Decrypted:", decrypted)

console.log("Solved:" )
let solved = solve(str, maxShiftAmount);

function encrypt(str, shiftAmount) {
    //use map to iterate through the string without loop
  const encryptedString = str.split("").map(char => {
    //get the beginning index of char to go through the map
    const charAscii = char.charCodeAt(0);
    //equation used in in all ciphers
    const encryptedChar = ((charAscii - 65 + shiftAmount) % 26 + 26) % 26 + 65;
    //needed to change the ascii back to char value
    return String.fromCharCode(encryptedChar);
  });
  //returns the joined characters(Now string)
  return encryptedString.join("");
}


function decrypt(str, shiftAmount) {
    //use map to iterate through the string without loop
  const decryptedString = str.split("").map(char => {
    //get the beginning index of char to go through the map
    const charAscii = char.charCodeAt(0);
    //equation used in in all ciphers
    const decryptedChar = ((charAscii - 65 - shiftAmount) % 26 + 26) % 26 + 65;
    //needed to change the ascii back to char value
    return String.fromCharCode(decryptedChar);
  });
  //returns the joined characters(Now string)
  return decryptedString.join("");
}

function solve(str, maxShiftAmount) {
```

```
    if(maxShiftAmount >= 0){
        positiveShiftValueRecursion(str, maxShiftAmount);
    }

    if(maxShiftAmount < 0){
        negativeShiftValueRecursion(str, maxShiftAmount);
    }

    function positiveShiftValueRecursion(str, maxShiftAmount){
        if( maxShiftAmount >= 0){
            let solveEncrypt = encrypt(str, maxShiftAmount);
            console.log("Ceaser", maxShiftAmount,  ":",  solveEncrypt);
            positiveShiftValueRecursion(str, maxShiftAmount - 1);
      }
    }

    function negativeShiftValueRecursion(str, maxShiftAmount){
        if(maxShiftAmount <= 0){
            let solveDecrypt = decrypt(str, -maxShiftAmount);
            console.log("Caesar", maxShiftAmount, ":", solveDecrypt);
            negativeShiftValueRecursion(str, maxShiftAmount + 1);
      }
    }

}
```

Test Cases (JavaScript)

var str = "HAL";
var shiftAmount = 3;
var maxShiftAmount = 26;
	var str = "CONNOR";
	var shiftAmount = -3;
	var maxShiftAmount = 28;
		var str = "ZAB";
		var shiftAmount = -3;
		var maxShiftAmount = -28;

Encrypted: KDO Decrypted: HAL Solved: Ceaser 26 : HAL Ceaser 25 : GZK Ceaser 24 : FYJ	Encrypted: ZLKKLO Decrypted: CONNOR Solved: Ceaser 28 : EQPPQT Ceaser 27 : DPOOPS Ceaser 26 : CONNOR	Encrypted: WXY Decrypted: ZAB Solved: Caesar -28 : XYZ Caesar -27 : YZA Caesar -26 : ZAB

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

```
Ceaser 25 : BNMMNQ
Ceaser 24 : AMLLMP
Ceaser 23 : ZLKKLO
Ceaser 22 : YKJJKN
Ceaser 21 : XJIIJM
Ceaser 20 : WIHHIL
Ceaser 19 : VHGGHK
Ceaser 18 : UGFFGJ
Ceaser 17 : TFEEFI
Ceaser 16 : SEDDEH
Ceaser 15 : RDCCDG
Ceaser 14 : QCBBCF
Ceaser 13 : PBAABE
Ceaser 12 : OAZZAD
Ceaser 11 : NZYYZC
Ceaser 10 : MYXXYB
Ceaser 9 : LXWWXA
Ceaser 8 : KWVVWZ
Ceaser 7 : JVUUVY
Ceaser 6 : IUTTUX
Ceaser 5 : HTSSTW
Ceaser 4 : GSRRSV
Ceaser 3 : FRQQRU
Ceaser 2 : EQPPQT
Ceaser 1 : DPOOPS
Ceaser 0 : CONNOR
```

```
Caesar -25 : ABC
Caesar -24 : BCD
Caesar -23 : CDE
Caesar -22 : DEF
Caesar -21 : EFG
Caesar -20 : FGH
Caesar -19 : GHI
Caesar -18 : HIJ
Caesar -17 : IJK
Caesar -16 : JKL
Caesar -15 : KLM
Caesar -14 : LMN
Caesar -13 : MNO
Caesar -12 : NOP
Caesar -11 : OPQ
Caesar -10 : PQR
Caesar -9 : QRS
Caesar -8 : RST
Caesar -7 : STU
Caesar -6 : TUV
Caesar -5 : UVW
Caesar -4 : VWX
Caesar -3 : WXY
Caesar -2 : XYZ
Caesar -1 : YZA
Caesar 0 : ZAB
```

SCALA

```
object CeaserCipher {
  def main(args: Array[String]) {
    var str = "HAL"
    var shiftAmount = 3

    val encryptedStr = encrypt(str, shiftAmount)
    println("Encrypted: " + encryptedStr)

    val decryptedStr = decrypt(encryptedStr, shiftAmount)
    println("Decrypted: " + decryptedStr)

    println("Solve:")
    solve(str) // Pass the encrypted string to the solve function
  }

  def encrypt(str: String, shiftAmount: Int): String = {
    // Using toCharArray method to convert the string to an array of characters
    str.toCharArray.collect {
      // Using a case statement to match on each character in the array
      case character if character.isUpper => (((character - 65 + shiftAmount) % 26 + 26) % 26 +
65).toChar // Encrypt uppercase character
      case character => character // Return unchanged for any other character
    }.mkString // Convert the array of characters back to a string
  }

  def decrypt(str: String, shiftAmount: Int): String = {
    // Using toCharArray method to convert the string to an array of characters
    str.toCharArray.collect {
      // Using a case statement to match on each character in the array
      case character if character.isUpper => (((character - 65 - shiftAmount) % 26 + 26) % 26 +
65).toChar // subtract the shiftamout for decrypt
      case character => character// Return unchanged for any other character
    }.mkString // Convert the array of characters back to a string
  }

  def solve(str:String) : Unit = {
    var maxShiftAmount = 26 // could not set maxShiftAmount in main if I wanted to increment

    //if maxShiftAmount is postive
    if (maxShiftAmount >=0){
      positiveShiftValueRecursion(maxShiftAmount)
    }
```

```
    //if maxShiftAmount is negative
    if (maxShiftAmount <0){
      negativeShiftValueRecursion(maxShiftAmount)
    }

    def positiveShiftValueRecursion(maxShiftAmount: Int): Unit = {
      if( maxShiftAmount >= 0){
        var solveEncrypt = encrypt(str, maxShiftAmount) //calls from the encrypt function to
continuously get results
        println("Ceaser " + maxShiftAmount +  ": " +  solveEncrypt)
        positiveShiftValueRecursion(maxShiftAmount - 1) //keep maxShiftAmount at positive number
and decrease to 0 to print properly
      }
    }

    def negativeShiftValueRecursion(maxShiftAmount: Int): Unit = {
      if(maxShiftAmount <= 0){
        val solveDecrypt = decrypt(str, -maxShiftAmount) // Pass the absolute value of
maxShiftAmount to decrypt
        println("Caesar " + maxShiftAmount + ": " + solveDecrypt)
        negativeShiftValueRecursion(maxShiftAmount + 1)
      }
    }
  }
}
```

Test Cases (SCALA)

var str = "HAL" var shiftAmount = 3 var maxShiftValue = 26	var str = "POLAND SPRINGS" var shiftAmount = 27 var maxShiftValue = -26	var str = "!HOWDY!" var shiftAmount = -4 var maxShiftValue = -28
Encrypted: KDO Decrypted: HAL Solve: Ceaser 26: HAL Ceaser 25: GZK Ceaser 24: FYJ	Encrypted: QPMBOE TQSJOHT Decrypted: POLAND SPRINGS Solve: Caesar -26: POLAND SPRINGS Caesar -25: QPMBOE TQSJOHT Caesar -24: RQNCPF URTKPIU	Encrypted: !DKSZU! Decrypted: !HOWDY! Solve: Caesar -28: !FMUBW! Caesar -27: !GNVCX! Caesar -26: !HOWDY!

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

```
Caesar -23: SRODQG  VSULQJV
Caesar -22: TSPERH  WTVMRKW
Caesar -21: UTQFSI  XUWNSLX
Caesar -20: VURGTJ  YVXOTMY
Caesar -19: WVSHUK  ZWYPUNZ
Caesar -18: XWTIVL  AXZQVOA
Caesar -17: YXUJWM  BYARWPB
Caesar -16: ZYVKXN  CZBSXQC
Caesar -15: AZWLYO  DACTYRD
Caesar -14: BAXMZP  EBDUZSE
Caesar -13: CBYNAQ  FCEVATF
Caesar -12: DCZOBR  GDFWBUG
Caesar -11: EDAPCS  HEGXCVH
Caesar -10: FEBQDT  IFHYDWI
Caesar -9: GFCREU  JGIZEXJ
Caesar -8: HGDSFV  KHJAFYK
Caesar -7: IHETGW  LIKBGZL
Caesar -6: JIFUHX  MJLCHAM
Caesar -5: KJGVIY  NKMDIBN
Caesar -4: LKHWJZ  OLNEJCO
Caesar -3: MLIXKA  PMOFKDP
Caesar -2: NMJYLB  QNPGLEQ
Caesar -1: ONKZMC  ROQHMFR
Caesar 0: POLAND  SPRINGS
```

```
Caesar -25: !IPXEZ!
Caesar -24: !JQYFA!
Caesar -23: !KRZGB!
Caesar -22: !LSAHC!
Caesar -21: !MTBID!
Caesar -20: !NUCJE!
Caesar -19: !OVDKF!
Caesar -18: !PWELG!
Caesar -17: !QXFMH!
Caesar -16: !RYGNI!
Caesar -15: !SZHOJ!
Caesar -14: !TAIPK!
Caesar -13: !UBJQL!
Caesar -12: !VCKRM!
Caesar -11: !WDLSN!
Caesar -10: !XEMTO!
Caesar -9: !YFNUP!
Caesar -8: !ZGOVQ!
Caesar -7: !AHPWR!
Caesar -6: !BIQXS!
Caesar -5: !CJRYT!
Caesar -4: !DKSZU!
Caesar -3: !ELTAV!
Caesar -2: !FMUBW!
Caesar -1: !GNVCX!
Caesar 0: !HOWDY!
```

Experience with the Procedural Programming Language

*In order of preference

Javascript

Time expected ≈ 5 hours
Time finished ≈ 3:25 hours

Task	Accomplishments	Issues	Thoughts
Learn basic syntax	Wasnt anything out of the ordinary with javascript, I felt like I knew most of the syntax already by coding in it before		-Create a string -Create an integer -Print them together
Learn how to call from different functions.	Calling from functions where not hard as well		I tried testing from my actual knowledge of javascript to call variables from other functions, which surprisingly worked. Go me
Understand recursive methods	Ended up using the map method, which was fairly the same as the scala and F# readability implementation		
Implement cipher code for encrypting and decrypting.	Successfully turned individual char values into ascii values, cipher it, and then combined them into a string	str.split("") - took me a while to find this method and implement it correctly	Cipher code for the encrypt and decrypt was again not that hard; implementation of the same equation gets pretty easy
Get solve to work	Pre-made the functions and copied my code from my scala, and fix the var to let		With this being my last language, I noticed that the easiest way of implementing recursion for the condition of positive or negative maxShiftValues

List of Google Searches:

- https://www.programiz.com/javascript/online-compiler/
- https://www.w3schools.com/js/js_variables.asp
- https://www.w3schools.com/js/js_functions.asp
- https://www.freecodecamp.org/news/javascript-switch-case-js-switch-statement-example/#:~:text=The%20computer%20will%20go%20through,that%20case%20clause%20will%20execute.&text=If%20none%20of%20the%20cases%20match%20the%20expression%2C%20then,default%20clause%20will%20be%20executed.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Stri

ng/charCodeAt
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/fromCharCode
- https://www.w3schools.com/jsref/jsref_split.asp

Scala

Time expected ≈ 2 hours
Time finished ≈ 2:30 hours

Task	Accomplishments	Issues	Thoughts
Learn map and case	Do begin, most of this code was taken from my past Scala code		First, I couldn't find any relevant websites that could adequately explain how to use a map or case method, so I used chat gpt to understand better and test my options. Ended up using the case methods
Understand the =>			Now that I understand the functional literal, you can use it to help match what you are looking for. In this case, you want your character to be uppercase, so you use the 'case character if character.isUpper()', and since that is true, the => acts as a function going through those characters and performs the encrypt/decrypt.
.toCharArray, .collect, .mkString		If you print: println(str.toCharArray), you will be returned with hexadecimal. You would have to add the .mkString to it	.toCharArray is used to convert the string into a full array of chars. .collect statement is used to take whatever takes from the case statement .mkString converts the array of characters to a string
Figure out how to perform the while loop functionally.		Had some issues trying to properly make a local function so that it would know to call if the item is positive or negative recursively Having a hard time	Figured out how to complete this using: https://www.learningjournal.guru/article/scala/functions-in-scala/local-functions/. It was not a hard

		figuring out why my solveDecrypt won't properly execute, even though it is the same procedure as I did in the last assignment but recursive Below is a screenshot of the final result I got from chaptgpt's debugging. I knew exactly what the problem was and tried using ways I used from the past assignment however, nothing worked. Finally, Chatgpt suggests adding a '-' sign to my maxshiftvalue when calling the decrypt. This took a while for both of us to figure out, even after knowing and telling it where the problem was the entire time. Guess AI isn't all that great	

List of Google Searches:

- https://www.baeldung.com/scala/loops-functional-scala
- https://www.geeksforgeeks.org/scala-map-method/
- https://stackoverflow.com/questions/10981037/functional-programming-for-and-while-loops
- https://www.baeldung.com/scala/pattern-matching
- https://www.learningjournal.guru/article/scala/functions-in-scala/local-functions/
- https://stackoverflow.com/questions/6951895/what-does-and-mean-in-scala#:~:text=%3D%3E%20is%20the%20%22function%20arrow,like%20void%20in%20other%20languages).

```
  def solve(str:String) : Unit = {
    var maxShiftValue = -26 // could not set maxShiftValue in main if I want

    //if maxShiftValue is postive
    if (maxShiftValue >=0){
      positiveShiftValueRecursion(maxShiftValue)
    }

    //if maxShiftValue is negative
    if (maxShiftValue <=0){
      negativeShiftValueRecursion(maxShiftValue)
    }

    def positiveShiftValueRecursion(maxShiftValue: Int): Unit = {
      if( maxShiftValue >= 0){
        var solveEncrypt = encrypt(str, maxShiftValue) //calls from the encr
        println("Ceaser " + maxShiftValue +  ": " +  solveEncrypt)
        positiveShiftValueRecursion(maxShiftValue - 1) //keep maxShiftValue
      }
    }

    def negativeShiftValueRecursion(maxShiftValue: Int): Unit = {
      if(maxShiftValue <= 0){
        val solveDecrypt = decrypt(str, -maxShiftValue) // Pass the absolute
        println("Caesar " + maxShiftValue + ": " + solveDecrypt)
        negativeShiftValueRecursion(maxShiftValue + 1)
      }
    }
  }
}
```

In the `**negativeShiftValueRecursion**` function, I changed the line `**val solveDecrypt = decrypt(str, maxShiftValue)**` to `**val solveDecrypt = decrypt(str, -maxShiftValue)**`. This takes the absolute value of `**maxShiftValue**` and passes it to the `**decrypt**` function. This way, the `**decrypt**` function subtracts the absolute value of `**maxShiftValue**` from each character in the encrypted string, which effectively adds `**maxShiftValue**` to

Regenerate response

F#

Time expected ≈ 5 hours
Time finished ≈ 2:20 hours

Task	Accomplishments	Issues	Thoughts
Learn basic syntax		I can't add comments to my code if I do, then my code will not run on jdoodle and with throw errors. I used the keybindings command+/ for auto commenting, and that was not working A big thing in a lot of these languages I am seeing is that they don't use parentheses when for example, declaring arguments for a function. It is easier to my eyes to know that something is inside parentheses rather than being in the open	A little weird that you need placeholders in your print statements for the variables Created a toUpper function so that future function doesn't have to worry about lowercase letters. Also helps so that I don't have to constantly call the bulky phrase 'toUpperCase lowerstr' in my code Interesting that type calling for a function like String or Unit appears to be colored, so you would think that those would work, but they have to under case to work
Learn how to call from different functions.		No real issues calling from a different function	Reminds me a lot of how Scala does its functions
Understand recursive methods	Took a little work but got the seq map to work and encrypt the characters. However, I still need to concatenate. -All I had to do was toArray it	|> learning piping (explained to me mainly from chatgpt)	After reading about the map function, research the seq map function
Implement cipher code for encrypting and decrypting.		After completing the encryption with seq.map, I needed to return the encrypted string. By the website on F# functions, I thought you didn't	

		have to return anything and you could just leave it alone. That was a little dumb of me to think I didn't have to return a variable to call it to another outside function, but who knows with these languages, Made the decrypt by changing the +shiftAmount to -shiftAmount and calling encrypted inside the decrypt function call. Got an error. Tried debugging for a little and asked chatgpt who said doing 'let decrypted = decrypt encrypted shiftAmount' can't be done. I re-worded and asked, and it told me to do that exact thing, which worked. AI is not on my side	
Get solve to work	I solved my error with the positiveShifValue function problem by completely deleting it and just having conditions for when maxShiftValue is either > 0 or < 0 and nothing equal. Unforntoanly if I was to add the equal then both would print, but whatever I did, it wouldn't let me call a different function for this to work properly		Finding out that there are nested functions allowed makes my experience a lot easier…I lied, I guess Started to get an error saying: 'The value or constructor 'positiveShiftValueRecursion' is not defined.' even though it is and is constructed before the function is called

List of Google Searches:

- https://www.tutorialspoint.com/fsharp/fsharp_variables.htm

- https://stackoverflow.com/questions/66377947/how-to-simple-make-string-uppercase-in-f
- https://www.tutorialspoint.com/fsharp/fsharp_functions.htm
- https://stackoverflow.com/questions/2980460/how-to-apply-seq-map-function
- https://stackoverflow.com/questions/40467117/nested-functions-in-f
- https://stackoverflow.com/questions/41209152/how-does-f-implement-let-rec

CLISP

Time expected ≈ 5 hours
Time finished ≈ 5:15 hours

Task	Accomplishments	Issues	Thoughts
Learn basic syntax	Figured out how to print variables and print them together		Went off of the first link provided. It's funny How the comment feature is ‘; ’
Learn how to calls from different functions.	Successfully printed from a different function.	I am used to putting comma-separated arguments.	Used the functions link above to figure out how to call in arguments and print said arguments to make sure everything is implemented correctly.
Understand recursive methods	Figured out key functions to help me with creating the cipher	Later on, realized there was a problem with my cipher algorithm, but for a long time, no word was printing for encryption. Chatgpt debugged most of the code due to confusion on why the code wasn’t printing the expected results.	
Implement cipher code for encrypting and decrypting.	Coerce: changes the value of a variable. Needed for list Lambda used to take a single character ‘String creates the character list back into a string	Tried making functions inside of the encrypt, decrypt, and solve, which is not allowed inside of CLISP The operator in front of the equations is something I won't miss. Possible torture	
Get solve to work		The if statements in this language are very confusing. At first, for the	

		solve, I had it both in one if statement looking for if a maxshiftvalue was negative or positive, and even when both function calls were inside of the same if statement, it was still recognizing which one to use. I then switched it to two different if statements but that was not working for my code, so I went back to the one if statement I was experiencing a problem with my negative solve function not printing out correctly at the sam e time as my scala (bc I realized when doing CLISP that it was printing wrong)	

List of Google Searches:

- https://www.tutorialspoint.com/lisp/lisp_variables.htm
- https://www.tutorialspoint.com/lisp/lisp_functions.htm
- https://www.tutorialspoint.com/lisp/lisp_mapping_functions.htm
- https://www.tutorialspoint.com/lisp/lisp_lambda_functions.htm#:~:text=LISP%20allows%20you%20to%20write,(lambda%20(parameters)%20body)
- http://clhs.lisp.se/Body/f_coerce.htm
- https://www.tutorialspoint.com/lisp/lisp_arithmetic_operators.htm

Erlang

Time expected ≈ 6 hours
Time finished ≈ 3:16 hours

Task	Accomplishments	Issues	Thoughts
Learn basic syntax	*Check blow for chaptGPT conversations about print statements		I find it very interesting that inside of the export you have to declare how many arguments are inside of your function And you need to add a comma to every line other than the last line, which is a period. I wonder who thought that was a good idea? You can't have uncapitalized letters as a variable name
Learn how to call from different functions.			Very interesting how you can format a print statement to either be in its ASCII value or its string value (~s~n)=string (~w~n)=ascii
Implement cipher code for encrypt and decrypt	successfully added the list handling via the link of list handling, however, I used chatgpt to understand how to implement the encryptionChar function successfully and have it be in a list Decrypt is always easier as encrypt is done	Wasn't to sure how to implement what I wanted after learning about list handling - so used chatgpt to guide me in the right direction Could noy figure out how to accept special characters/spaces	Look up way to handle how to iterate characters through the string which found through the list handling function
Get solve to work	Chatgpt debug on my solve as I originally had my solve the same way I constructed my solve in the Scala cipher, however when debugging it I had asked chatgpt to see what it does, and it changed it to a case with true and false. I thought that this new method would be better, so I decided to		Randomly it is not letting do >= or <=, and when I implement that, it throws errors at me

	keep it		

List of Google Searches:

- https://www.tutorialspoint.com/erlang/erlang_basic_syntax.htm
- https://www.tutorialspoint.com/erlang/erlang_functions.htm
- https://www.erlang.org/doc/efficiency_guide/listhandling
- https://www.tutorialspoint.com/erlang/erlang_if_statement.htm
- https://www.tutorialspoint.com/compile_erlang_online.php - online IDE
- https://www.tutorialspoint.com/erlang/erlang_case_statements.htm

lua Copy code

```
1> Str = <<"Hello, world!">>.
<<"Hello, world!">>
2> io:fwrite("~w~n", [Str]).
[72,101,108,108,111,44,32,119,111,114,108,100,33]
ok
3> StrList = binary_to_list(Str).
"Hello, world!"
4> io:fwrite("~s~n", [StrList]).
Hello, world!
ok
```

Printing basics in Erlang

scss Copy code

```
-module(helloworld).
-export([encrypt/2, start/0]).

encrypt(Str, ShiftAmount) ->
    EncryptedStr = [encrypt_char(C, ShiftAmount) || C <- Str],
    io:fwrite("~s~n", [EncryptedStr]).

encrypt_char(C, ShiftAmount) ->
    ((C - 65 - ShiftAmount) rem 26 + 26) rem 26 + 65.

start() ->
    Str = "HAL",
    ShiftAmount = 3,
    encrypt(Str, ShiftAmount).
```

In the `**encrypt**` function, we map each character of the string to its encrypted form using a list comprehension and store the result in `**EncryptedStr**`. The `**encrypt_char**` function takes a character `**C**` and the shift amount `**ShiftAmount**`, and returns the encrypted value of the character using the provided equation.

Adding the ecrypt_char function that also creates the list for the filter expression after the || which figures out each character in the string