Gianna Julio

CMPT331 - Theory of Programming Languages

Functional Languages

Contents

1 LISP

1.1 Program

```
(defun char-shift (ch shift)
  (cond
    ((and (char>= ch #\A) (char<= ch #\Z))
     (code-char (+ (char-code #\A) (mod (+ (- (char-code ch) (char-code #\A)) shift) 26))
         ))
    ((and (char>= ch #\a) (char<= ch #\z))
     (code-char (+ (char-code #\a) (mod (+ (- (char-code ch) (char-code #\a)) shift) 26))
         ))
    (t ch)))

(defun caesar-encrypt (text shift)
  (coerce (map 'list (lambda (ch) (char-shift ch shift)) text) 'string))

(defun caesar-decrypt (text shift)
  (caesar-encrypt text (- 26 (mod shift 26))))

(defun prompt (message)
  (format t "~a" message)
  (force-output)
  (read-line))

(defun prompt-int (message)
  (format t "~a" message)
  (force-output)
  (parse-integer (read-line)))

(defun brute-force (text)
  (format t "Solving the cipher with all 26 possible shifts:~%")
  (loop for i from 1 to 26 do
    (format t "Shift ~2d: ~a~%" i (caesar-decrypt text i))))

(defun main ()
  ;; Encrypt
  (let* ((text-to-encrypt (prompt "Enter text to encrypt: "))
         (shift-encrypt (prompt-int "Enter shift value: "))
         (encrypted (caesar-encrypt text-to-encrypt shift-encrypt)))
    (format t "~%Encrypted text: ~a~%~%" (string-upcase encrypted)))

  ;; Decrypt
  (let* ((text-to-decrypt (prompt "Enter text to decrypt: "))
         (shift-decrypt (prompt-int "Enter shift value: "))
         (decrypted (caesar-decrypt text-to-decrypt shift-decrypt)))
    (format t "~%Decrypted text: ~a~%~%" (string-upcase decrypted)))

  ;; Brute-force
  (let ((brute-input (prompt "Enter text for brute-force solve: ")))
    (brute-force brute-input)))

(main)
```

1.2 Sample Output

```
Enter text to encrypt: COMMONLISP
Enter shift value: 2
Encrypted text: EQOOQPNKUR

Enter text to decrypt: XJHHJIGDNK
Enter shift value: 21
Decrypted text: COMMONLISP

Enter text for brute-force solve: NZXXZYWTDA
Solving the cipher with all 26 possible shifts:
Shift  1: MYWWYXVSCZ
Shift  2: LXVVXWURBY
Shift  3: KWUUWVTQAX
Shift  4: JVTTVUSPZW
Shift  5: IUSSUTROYV
Shift  6: HTRRTSQNXU
Shift  7: GSQQSRPMWT
Shift  8: FRPPRQOLVS
Shift  9: EQOOQPNKUR
Shift 10: DPNNPOMJTQ
Shift 11: COMMONLISP
Shift 12: BNLLNMKHRO
Shift 13: AMKKMLJGQN
Shift 14: ZLJJLKIFPM
Shift 15: YKIIKJHEOL
Shift 16: XJHHJIGDNK
Shift 17: WIGGIHFCMJ
Shift 18: VHFFHGEBLI
Shift 19: UGEEGFDAKH
Shift 20: TFDDFECZJG
Shift 21: SECCEDBYIF
Shift 22: RDBBDCAXHE
Shift 23: QCAACBZWGD
Shift 24: PBZZBAYVFC
Shift 25: OAYYAZXUEB
Shift 26: NZXXZYWTDA
```

1.3 Work Log

Predicted Time: 4 hours **Actual Time:** 4 hours **Ranking:** #4

- **Similarities/Differences:** LISP was a very unique language to play around with. The insane amount of parentheses is annoying, but easily manageable with a code editor like VSCode that helps to keep track of matching parens and even colors them to help with readability.
- **Readability/Writability:** Despite all the parens, I think LISP is a pretty readable language once you understand the syntax. Writablility was made way easier with a code editor like VSCode. Two semicolons is a super strange way to denote comments.

- **What I Loved:** The function to encrypt is very short and powerful and easily repurposed for decryption. It was pretty confusing to understand exactly what was going on in the single line to be able to implement it in my program, but after some research and analysis it was impressive how simple LISP made the encryption.
- **What I Hated:** The simplicity of the language made some syntax hard to understand and it took a bit of research to figure some elements of the caesar cipher out, but there was a good amount of documentation (better than some other older languages like COBOL).
- **Why the Time Discrepancy?** There was really no time discrepancy with LISP - the parens and debugging was much easier when using VSCode.
- **AI/Google Searches Used:**
  - "LISP map list"
  - "LISP string formatting"
  - "LISP conditionals"
  - "lambda in LISP"

2 ML

2.1 Program

```
(* Shift a single character by n positions *)
fun shiftChar (c: char, shift: int) : char =
  let
    val code = ord c
    val baseUpper = ord #"A"
    val baseLower = ord #"a"
    fun mod26 n = if n < 0 then mod26 (n + 26) else n mod 26
  in
    if code >= ord #"A" andalso code <= ord #"Z" then
      chr (baseUpper + mod26 (code - baseUpper + shift))
    else if code >= ord #"a" andalso code <= ord #"z" then
      chr (baseLower + mod26 (code - baseLower + shift))
    else
      c
  end

(* Encrypt a string by shifting each character *)
fun encrypt (text: string, shift: int) : string =
  String.implode (List.map (fn c => shiftChar (c, shift)) (String.explode text))

(* Decrypt a string by shifting in the opposite direction *)
fun decrypt (text: string, shift: int) : string =
  encrypt (text, ~shift)

(* Brute force all possible Caesar shifts *)
fun bruteForce (text: string) : unit =
  let
    fun tryShifts n =
      if n > 26 then ()
      else (
        print ("Shift " ^ Int.toString n ^ ": " ^ encrypt(text, 26 - n) ^ "\n");
        tryShifts (n + 1)
      )
  in
    tryShifts 1
  end

(* Example I/O interaction *)
val () =
  let
    val _ = print "Enter text to encrypt: "
    val inputEnc = TextIO.inputLine TextIO.stdIn |> valOf |> String.trim
    val _ = print "Enter shift value: "
    val shiftEnc = TextIO.inputLine TextIO.stdIn |> valOf |> Int.fromString |> valOf
    val encrypted = encrypt(inputEnc, shiftEnc)
    val _ = print ("Encrypted text: " ^ encrypted ^ "\n\n")

    val _ = print "Enter text to decrypt: "
    val inputDec = TextIO.inputLine TextIO.stdIn |> valOf |> String.trim
    val _ = print "Enter shift value: "
```

```
    val shiftDec = TextIO.inputLine TextIO.stdIn |> valOf |> Int.fromString |> valOf
    val decrypted = decrypt(inputDec, shiftDec)
    val _ = print ("Decrypted text: " ^ decrypted ^ "\n\n")

    val _ = print "Enter text for brute-force solve: "
    val inputBrute = TextIO.inputLine TextIO.stdIn |> valOf |> String.trim
    val _ = print "Solving the cipher with all 26 possible shifts:\n"
    val _ = bruteForce inputBrute
  in
    ()
  end
```

2.2 Sample Output

```
Enter text to encrypt: A MAN, A PLAN, A CANAL, PANAMA
Enter shift value: 5
Encrypted text: F RFS, F UWTL, F HFQFQ, UFSTF

Enter text to decrypt: X PDQ, X SODQ, X FDQDO, SDQPD
Enter shift value: 3
Decrypted text: M ERF, M HDRD, M URFSF, HRSK

Enter text for brute-force solve: X PDQ, X SODQ, X FDQDO, SDQPD
Solving the cipher with all 26 possible shifts:
Shift  1: W OCP, W RNCQ, W ECPCN, RCPCO
Shift  2: V NBO, V QMBP, V DBOMB, QBONN
Shift  3: U MAN, U PLZO, U CANNA, PAOMM
Shift  4: T LZM, T OKYN, T BMMZ, OZNLL
Shift  5: S KYL, S NJXM, S ALLY, NYMKK
Shift  6: R JXK, R MIWL, R ZKKX, MXLJJ
Shift  7: Q IWL, Q LHVK, Q YJJW, LWKII
Shift  8: P HVK, P KGUJ, P XIXV, KVJHH
Shift  9: O GUJ, O JFTI, O WHWU, JUIGG
Shift 10: N FTJ, N IESH, N VGVT, ITHFF
Shift 11: M ESJ, M HDRG, M UFUS, HSGEE
Shift 12: L DRI, L GCQF, L TETR, GRFDD
Shift 13: K CQH, K FBP, K SDSQ, FECCC
Shift 14: J BPG, J EAO, J RCRP, EDBBB
Shift 15: I AOF, I DNZ, I QBQO, DCAAA
Shift 16: H ZNE, H CNY, H PAPP, CBZZZ
Shift 17: G YMD, G BMX, G OZOQ, BAYYY
Shift 18: F XLC, F ALW, F NYNP, AZXXX
Shift 19: E WKB, E ZKV, E MXMO, ZYWVW
Shift 20: D VJA, D YJU, D LWLN, YXUUV
Shift 21: C UIZ, C XIT, C KVK, WXTTU
Shift 22: B THY, B WHS, B JUJ, VWSST
Shift 23: A SGX, A VGR, A ITH, UVRRQ
Shift 24: Z RFV, Z UFQ, Z HSG, TUQQP
Shift 25: Y QUE, Y TEF, Y GRF, STPPO
Shift 26: X PDQ, X SODQ, X FDQDO, SDQPD
```

2.3 Work Log

Predicted Time: 3 hours **Actual Time:** 4 hours **Ranking:** #5

- **Similarities/Differences:** ML felt like simplified (and with less parens) version of LISP, but that actually made me like it less. While the fewer amount of parentheses made ML slightly easier to understand, I felt like the overall simplicity made ML harder to understand.
- **Readability/Writability:** ML was more readable than LISP, but not as writable in my opinion. The simplicity of the syntax made it harder for me to understand enough to effectively use it.
- **What I Loved:** Less parentheses!
- **What I Hated:** It was hard to find an online compiler for ML and the error messages produces were very unhelpful for debugging.
- **Why the Time Discrepancy?** Debugging was difficult. I switched between a few free online ML compilers and each was slightly different - but none gave very useful error messages.
- **AI/Google Searches Used:**
  - "ML declaring functions"
  - "ML user input"
  - "ML char manipulation"

3 Erlang

3.1 Program

```erlang
-module(caesar).
-export([main/1, encrypt/2, decrypt/2, brute_force/1]).

%%% Escript entry point
main(_Args) ->
    %% Encrypt section
    EncryptInput = prompt("Enter text to encrypt: "),
    ShiftEncrypt = get_shift_value(),
    Encrypted = encrypt(EncryptInput, ShiftEncrypt),
    io:format("\nEncrypted text: ~s~n~n", [Encrypted]),

    %% Decrypt section
    DecryptInput = prompt("Enter text to decrypt: "),
    ShiftDecrypt = get_shift_value(),
    Decrypted = decrypt(DecryptInput, ShiftDecrypt),
    io:format("\nDecrypted text: ~s~n~n", [Decrypted]),

    %% Brute-force section
    BruteInput = prompt("Enter text for brute-force solve: "),
    io:format("\nSolving the cipher with all 26 possible shifts:~n"),
    brute_force(BruteInput).

%%% Prompt and read trimmed line of input
prompt(Message) ->
    io:format("~s", [Message]),
    string:trim(io:get_line("")).

%%% Prompt for Caesar shift
get_shift_value() ->
    io:format("\nEnter shift value: "),
    {ok, [Shift]} = io:fread("", "~d"),
    Shift rem 26.

%%% Encrypt text with Caesar shift
encrypt(Text, Shift) ->
    lists:map(fun(Char) -> shift_char(Char, Shift) end, Text).

%%% Decrypt = encrypt with (26 - shift)
decrypt(Text, Shift) ->
    encrypt(Text, 26 - (Shift rem 26)).

%%% Brute force all Caesar shifts
brute_force(Text) ->
    brute_force_loop(Text, 1).

brute_force_loop(_, 27) -> ok;
brute_force_loop(Text, Shift) ->
    Dec = decrypt(Text, Shift),
    io:format("Shift ~2w: ~s~n", [Shift, Dec]),
    brute_force_loop(Text, Shift + 1).
```

```
51
52  %%% Shift a single character
53  shift_char(Char, Shift) when Char >= $A, Char =< $Z ->
54      $A + ((Char - $A + Shift) rem 26);
55  shift_char(Char, Shift) when Char >= $a, Char =< $z ->
56      $a + ((Char - $a + Shift) rem 26);
57  shift_char(Char, _Shift) ->
58      Char.
```

3.2 Sample Output

```
1   Enter text to encrypt: ERLANG
2   Enter shift value: 18
3   Encrypted text: WJDSFY
4
5   Enter text to decrypt: UHBQDW
6   Enter shift value: 16
7   Decrypted text: ERLANG
8
9   Enter text for brute-force solve: OBVKXQ
10  Solving the cipher with all 26 possible shifts:
11  Shift  1: NAUJWP
12  Shift  2: MZTIVO
13  Shift  3: LYSHUN
14  Shift  4: KXRGTM
15  Shift  5: JWQFSL
16  Shift  6: IVPERK
17  Shift  7: HUODQJ
18  Shift  8: GTNCPI
19  Shift  9: FSMBOH
20  Shift 10: ERLANG
21  Shift 11: DQKZMF
22  Shift 12: CPJYLE
23  Shift 13: BOIXKD
24  Shift 14: ANHWJC
25  Shift 15: ZMGVIB
26  Shift 16: YLFUHA
27  Shift 17: XKETGZ
28  Shift 18: WJDSFY
29  Shift 19: VICREX
30  Shift 20: UHBQDW
31  Shift 21: TGAPCV
32  Shift 22: SFZOBU
33  Shift 23: REYNAT
34  Shift 24: QDXMZS
35  Shift 25: PCWLYR
36  Shift 26: OBVKXQ
```

3.3 Work Log

Predicted Time: 3 hours **Actual Time:** 4 hours **Ranking:** #3

- **Similarities/Differences:** I felt like Erlang was a weird, worse Python when trying to use it for string manipulation. Erlang was obviously not intended to be general-use. It was easier to use than LISP and ML, but not as easy as Scala and JS.
- **Readability/Writability:** I thought Erlang was more readable than writable - the semi-colons, periods, and commas were a little confusing and added overhead.
- **What I Loved:** The list mapping made encryption a super simple function that I could repurposed for decryption as well.
- **What I Hated:** The syntax was a little bit of an annoyance - all of the different punctuation with different meanings was less intuitive than something like JavaScript.
- **Why the Time Discrepancy?** I had to look up how to do basic things like string iteration and character conversion. Functional logic wasn't the hard part, learning Erlang-specific syntax was.
- **AI/Google Searches Used:**
  - "Erlang character manipulation"
  - "Erlang function examples"
  - "Erlang user input"

4 JavaScript (functional)

4.1 Program

```
const readline = require('readline');

// setup I/O
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function shiftChar(c, shift) {
  // constants for easier read and writability
  const A = 'A'.charCodeAt(0);
  const Z = 'Z'.charCodeAt(0);
  const a = 'a'.charCodeAt(0);
  const z = 'z'.charCodeAt(0);
  const code = c.charCodeAt(0);

  // uppercase
  if (code >= A && code <= Z) {
    return String.fromCharCode(A + ((code - A + shift) % 26));
  // lowercase
  } else if (code >= a && code <= z) {
    return String.fromCharCode(a + ((code - a + shift) % 26));
  // other chars
  } else {
    return c;
  }
}

// if no chars left to encrypt, return accumulator
// otherwise, recurse through the rest of the chars and add shifted char to acc
function encrypt(text, shift) {
  const recurse = (chars, acc) =>
    chars.length === 0 ? acc : recurse(chars.slice(1), acc + shiftChar(chars[0], shift));
  return recurse(text.split(''), '');
}

// reverses encrypt function
function decrypt(text, shift) {
  return encrypt(text, (26 - (shift % 26)) % 26);
}

// brute-forces encryption by calling decrypt recursively with all 26 possible shifts
function bruteForce(text, shift = 1) {
  if (shift > 26) return;
  console.log(`Shift ${shift < 10 ? ' ' : ''}${shift}: ${decrypt(text, shift)}`);
  bruteForce(text, shift + 1);
}

// simplifies user input by calling the callback function with the answer
function ask(question, callback) {
```

```
  rl.question(question, answer => callback(answer));
}

// main function - encrypts, decrypts, brute-forces
function main() {
  ask('\nEnter text to encrypt: ', textToEncrypt => {
    ask('\nEnter shift value: ', shiftEncrypt => {
      const encrypted = encrypt(textToEncrypt, parseInt(shiftEncrypt));
      console.log(`\nEncrypted text: ${encrypted.toUpperCase()}\n`);

      ask('\nEnter text to decrypt: ', textToDecrypt => {
        ask('\nEnter shift value: ', shiftDecrypt => {
          const decrypted = decrypt(textToDecrypt, parseInt(shiftDecrypt));
          console.log(`\nDecrypted text: ${decrypted.toUpperCase()}\n`);

          ask('\nEnter text for brute-force solve: ', textToBrute => {
            console.log('\nSolving the cipher with all 26 possible shifts:');
            bruteForce(textToBrute);
            rl.close();
          });
        });
      });
    });
  });
}

main();
```

4.2 Sample Output

```
Enter text to encrypt: JAVASCRIPT IS FUN
Enter shift value: 17
Encrypted text: ARMRJTIZGK ZJ WLE


Enter text to decrypt: FWRWOYNELP EO BQJ
Enter shift value: 22
Decrypted text: JAVASCRIPT IS FUN


Enter text for brute-force solve:
Solving the cipher with all 26 possible shifts:
Shift  1: ULGLDNCTAE TD QFY
Shift  2: TKFKCMBSZD SC PEX
Shift  3: SJEJBLARYC RB ODW
Shift  4: RIDIAKZQXB QA NCV
Shift  5: QHCHZJYPWA PZ MBU
Shift  6: PGBGYIXOVZ OY LAT
Shift  7: OFAFXHWNUY NX KZS
Shift  8: NEZEWGVMTX MW JYR
Shift  9: MDYDVFULSW LV IXQ
Shift 10: LCXCUETKRV KU HWP
Shift 11: KBWBTDSJQU JT GVO
```

```
Shift 12: JAVASCRIPT IS FUN
Shift 13: IZUZRBQHOS HR ETM
Shift 14: HYTYQAPGNR GQ DSL
Shift 15: GXSXPZOFMQ FP CRK
Shift 16: FWRWOYNELP EO BQJ
Shift 17: EVQVNXMDKO DN API
Shift 18: DUPUMWLCJN CM ZOH
Shift 19: CTOTLVKBIM BL YNG
Shift 20: BSNSKUJAHL AK XMF
Shift 21: ARMRJTIZGK ZJ WLE
Shift 22: ZQLQISHYFJ YI VKD
Shift 23: YPKPHRGXEI XH UJC
Shift 24: XOJOGQFWDH WG TIB
Shift 25: WNINFPEVCG VF SHA
Shift 26: VMHMEODUBF UE RGZ
```

4.3 Work Log

Predicted Time: 2 hours **Actual Time:** 3 hours **Ranking:** #1

- **Similarities/Differences:** I am already familiar with JavaScript, so the challege here came when trying to program functionally and without loops like I am familiar with.
- **Readability/Writability:** Readability and writability are what you would expect from a popular modern language: pretty simple, although I may be biased as I have some experience with JS.
- **What I Loved:** Arrow functions and callbacks made functional programming pretty easy once I got the hang of them. The readline module also made I/O much easier. Since JS is so modern and so popular, there is a lot of documentation and premade modules to make implementation a lot easier.
- **What I Hated:** The lack of a built-in char type was a little annoying, but declaring some constants made both readability and writability a lot easier.
- **Why the Time Discrepancy?** I got a little bit tripped up with the callback functions and recursion, and it took a little while to debug.
- **AI/Google Searches Used:**
  - "JavaScript readline module"
  - "JavaScript callbacks"
  - "JavaScript arrow functions"

5 Scala (functional)

5.1 Program

```scala
import scala.io.StdIn.readLine

object CaesarCipher {

  def shiftChar(c: Char, shift: Int): Char = {
    val code = c.toInt

    // uppercase
    if (c.isUpper) {
      ('A' + (code - 'A' + shift) % 26).toChar
    // lowercase
    } else if (c.isLower) {
      ('a' + (code - 'a' + shift) % 26).toChar
    // other chars
    } else {
      c
    }
  }

    /**
     * encrypts the given text using the Caesar cipher with the specified shift
     * if the string is empty, return
     * otherwise, reecursively call shift function and add the encrypted char to the
         result
     *
     * @param text the text to encrypt
     * @param shift number of positions to shift each char
     * @return encrypted text
     */
  def encrypt(text: String, shift: Int): String = {
    def recurse(chars: List[Char]): String = chars match {
      case Nil => ""
      case head :: tail => shiftChar(head, shift) + recurse(tail)
    }
    recurse(text.toList)
  }

  /**
   * decrypts the given text using the Caesar cipher with the specified shift
   * reverses encrypt function by shifting char in the opposite direction
   *
   * @param text the text to decrypt
   * @param shift number of positions to shift each char
   * @return decrypted text
   */
  def decrypt(text: String, shift: Int): String = {
    encrypt(text, (26 - shift % 26) % 26)
  }

  /**
```

```
   * brute-force calls decrypt with all 26 possible shifts
   *
   * @param text the text to decrypt
   * @param shift number of positions to shift each char
   */
  def bruteForce(text: String, shift: Int = 1): Unit = {
    if (shift > 26) ()
    else {
      val paddedShift = if (shift < 10) s" $shift" else s"$shift"
      println(s"Shift $paddedShift: ${decrypt(text, shift)}")
      bruteForce(text, shift + 1)
    }
  }

  /**
   * main function
   * encrypt, decrypt, brute-force solve
   *
   * @param args command line arguments (not used)
   */
  def main(args: Array[String]): Unit = {
    val toEncrypt = readLine("Enter text to encrypt: ")
    val shiftEncrypt = readLine("\nEnter shift value: ").toInt
    val encrypted = encrypt(toEncrypt, shiftEncrypt)
    println(s"\nEncrypted text: ${encrypted.toUpperCase()}\n")

    val toDecrypt = readLine("\nEnter text to decrypt: ")
    val shiftDecrypt = readLine("\nEnter shift value: ").toInt
    val decrypted = decrypt(toDecrypt, shiftDecrypt)
    println(s"\nDecrypted text: ${decrypted.toUpperCase()}\n")

    val toBrute = readLine("\nEnter text for brute-force solve: ")
    println("\nSolving the cipher with all 26 possible shifts:")
    bruteForce(toBrute)
  }
}
```

5.2 Sample Output

```
Enter text to encrypt: TESTING SCALA PROGRAM
Enter shift value: 3
Encrypted text: WHVWLQJ VFDOD SURJUDP


Enter text to decrypt: QBPQFKD PZXIX MOLDOXJ
Enter shift value: 23
Decrypted text: TESTING SCALA PROGRAM


Enter text for brute-force solve: KVJKZEX JTRCR GIFXIRD
Solving the cipher with all 26 possible shifts:
Shift  1: JUIJYDW ISQBQ FHEWHQC
Shift  2: ITHIXCV HRPAP EGDVGPB
```

```
Shift  3: HSGHWBU GQOZO DFCUFOA
Shift  4: GRFGVAT FPNYN CEBTENZ
Shift  5: FQEFUZS EOMXM BDASDMY
Shift  6: EPDETYR DNLWL ACZRCLX
Shift  7: DOCDSXQ CMKVK ZBYQBKW
Shift  8: CNBCRWP BLJUJ YAXPAJV
Shift  9: BMABQVO AKITI XZWOZIU
Shift 10: ALZAPUN ZJHSH WYVNYHT
Shift 11: ZKYZOTM YIGRG VXUMXGS
Shift 12: YJXYNSL XHFQF UWTLWFR
Shift 13: XIWXMRK WGEPE TVSKVEQ
Shift 14: WHVWLQJ VFDOD SURJUDP
Shift 15: VGUVKPI UECNC RTQITCO
Shift 16: UFTUJOH TDBMB QSPHSBN
Shift 17: TESTING SCALA PROGRAM
Shift 18: SDRSHMF RBZKZ OQNFQZL
Shift 19: RCQRGLE QAYJY NPMEPYK
Shift 20: QBPQFKD PZXIX MOLDOXJ
Shift 21: PAOPEJC OYWHW LNKCNWI
Shift 22: OZNODIB NXVGV KMJBMVH
Shift 23: NYMNCHA MWUFU JLIALUG
Shift 24: MXLMBGZ LVTET IKHZKTF
Shift 25: LWKLAFY KUSDS HJGYJSE
Shift 26: KVJKZEX JTRCR GIFXIRD
```

5.3 Work Log

Predicted Time: 2 hours **Actual Time:** 3 hours **Ranking:** #2

- **Similarities/Differences:** With the OOP structure, Scala was not too difficult to implement functionally. Like ML but with Java influence. Feels more modern and expressive than most others, but debugging was still a chore as it is in Java.
- **Readability/Writability:** Since Scala looks so similar to Java, I considered it to be very readable and writable, including the recursive functions.
- **What I Loved:** Recursive functions were much easier to write in Scala than most of the other languages. Scala made functional programming very short and simple.
- **What I Hated:** Since this was the second time I had written a caesar cipher in Scala, the user input was not the main source of my debugging, but rather the recursion. The debugging time for recursive functions was minimal, but still an annoyance.
- **Why the Time Discrepancy?** Mostly debugging and getting recursive functions to work properly while being a little unfamiliar with the syntax still.
- **AI/Google Searches Used:**
  - "Scala recursion examples"
  - "Scala string handling functions"
  - "Scala recursive functions"

6 Summary

Again, it was fun to explore new languages, but especially so because LISP and ML have such specific purposes. While I likely wont be using languages like those again in the future, it was a good exercise for understanding functional program. I enjoyed the challenge of also implementing this new way of programming (functionally) into languages I was more familiar with like JavaScript and Scala. I got to see how powerful recursion really is.

7 Resources

7.1 Compilers and IDEs

- OneCompiler: free online compiler to run programs
- SoSML: SML online compiler