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 1: Programming In The Past

Connor H. Johnson
connor.johnson1@marist.edu

Table of Context

Assignment 1: Programming In The Past

Connor H. Johnson

February 15, 2023

Overview

Within this document, I will describe step-by-step the method I took to complete the assignment of coding the Caesar Cipher in 5 different procedural programming languages: Scala, Pascal, BASIC, FORTRAN, and COBOL.
Each language should contain three functions within the Caesar Cipher code: Encrypt(), Decrypt(), and Solve(). The Encrypt function will take in the 'mainString' variable containing the message and a 'shiftAmount'varibale 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 that 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 use the encrypted function to iterate. If the 'maxShiftValue' is negative, then the function shall 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.
All code was worked on and tested on the multi-language online IDE: JDOODLE

Steps Used for Each Procedural 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,

Scala:
```
var mainString = "HAL"
```

COBOL:
```
77 mainString PIC A(3) value "HAL".
```

I thought a great way to start would be researching how to declare variables and print them on the same line. This was not a very difficult step, but it definitely was 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 will take the arguments of 'mainString' and 'shiftAmount.'

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

c.) If the variables were printed out correctly, my next step was researching 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 for-loop syntax.

a.) Every for-loop turned out to have different syntax, so it became really important that I could figure out how to code this loop properly.

b.) The main objective is to have a variable called either 'character', 'position,' or 'characterized' (due to some names being keywords) iterate through the mainString to acquire the char's position.

c.) Once this for-loop is created, I test it by printing out 'character' inside the for loop to print (With Scala being an exception as that will print out the 'mainString'):

```
1
2
3
```

d.) After this, I print out mainString[character] or mainString[character:character] to have the index value of the character as seen above print out the specific char value inside of mainString. This result will be(with mainString = 'HAL'):

```
H
A
L
```

4.) Research the equivalence to the char() and ord() functions in the respective language.

a.) A considerable part of the Caesar Cipher is learning about the char() and the ord() functionality inside of these procedural programming languages. However, the name of these functions was changed in almost every language but still did the same thing.

- Scala: .toInt .toChar
- Pascal: ord() chr()
- FREE BASIC: chr()
- FORTRAN: ichar() achar()
- COBOL: ord() char()

b.) Here is an example of both (FORTRAN):

- `print*, ichar("A")` prints out '65.'
- `print*, achar(65)` prints out 'A.'

The ASCII chart can be referred to here from W3Schools: https://www.w3schools.com/charsets/ref_html_ascii.asp

5.) Understand the if-else syntax.

a.) Now that I know that my for-loop can iterate through the 'mainString' and adequately obtain the integer and string value of each letter individually, I can see if the mainString char is in between the ASCII value of the capitalized alphabet.

b.) Since we were told in class only to use capital letters in our mainString to make it less complicated, the beginning of the if-else statement will look similar to this(Pascal):

```
if (ord(decryptedString[character]) >= 65) and (ord(decryptedString[character]) <= 90) then
```

c.) This code ensures that the capitalized letter's integer value, found by the for-loop presented before and now being changed using the ord() function, is in between the capitalized alphabet

d.) If this statement is true, the next line of code will be to execute the formula that returns me the Caesar Cipher letter shifted 'shiftValue' amount of times. Here is an example of the formula I used(Scala encryption):

```
(((position.toInt - 65 + shiftAmount) % 26 + 26) % 26 + 65).toChar
```

Here is an explanation of what the formula is doing:

- 'position.toInt' converts the character to its ASCII code integer.
- '- 65' subtracts 65 from the position.toInt value to get the character's position between 0('a') to 25('z').
- '+ shiftAmount' adds the 'shiftAmount' to the position so that it can be shifted by a 'shiftAmount' of positions to the right.
- '% 26' mods the shifted position by 26 so the result stays in between 0('a') to 25('z'), and no special characters are printed.
- '+ 26' adds 26 to the result, which is always positive.
- '% 26' mods the result by 26 so that the result stays between 0('A') to 25('Z').
- '+ 65' adds 65 to the result to convert the position back to the ASCII code range 65('A') to 90('Z').
- '.toChar' converts the ASCII code value back to the corresponding character.

6.) Learn concatenation in the respective language.

a.) Now that the formula works, if I were to print that formula, I would get my result. However, the characters will be printed on separate lines. This is due to not

concatenating the char's together once put through the cipher. Just like the functions beforehand, these languages all had different methods of concatenation:

- Scala: encryption +=
- Pascal: encryption :=
- FREE BASIC: encryption +=
- FORTRAN: encrypted = (variable equal to formula) // encrypted
- COBOL: move (variable equal to formula) to encrypted(:)

b.) Now that concatenation is available, I can complete the if statement from before(Scala):

```
    def encrypt( str:String, shiftAmount:Int ) : String = {
        var encrypted = ""
        for (position <- str){    //iterate through the initial message
                                  //no ord() function = use .toInt
            if (position.toInt >= 65 && position.toInt <= 90) {
// ASCII *Uppercase* letter in alphabet
                encrypted += (((position.toInt - 65 + shiftAmount) % 26 + 26) % 26 +
65).toChar  //.toChar takes the ASCII number and converts to number
            } else {
                encrypted += position  //adds if not a character in alphabet
            }
        }
         encrypted



    }
```

c.) Above, you can see the else statement put in place to catch anything not between the range 65 to 90 or, in other words, anything that is not a capital letter. This will result in no changes, and the special character in 'mainString' will be printed out

7.) Understand the while loop syntax.

a.) Now that the encrypt is done, the decryption is fairly the same, except you have to call the encrypted word to decrypt the word back to what 'mainString' was assigned to originally. With that, you also have to change the formula so that you are now subtracting the shift value, for example(Scala):

```
(((position.toInt - 65 - shiftAmount) % 26 + 26) % 26 + 65).toChar
```

b.) Now, the while loop comes into play with the Solve function. Within this function, you want to check whether the 'maxShiftValue' is either positive or negative. This is because if it is positive, you will have to encrypt the message; if it is negative, you will decrypt it. Here is the code(Scala):

```
     def solve(str:String) : Unit = {
       var maxShiftValue = 26
        //if maxShiftValue is positive
       if (maxShiftValue >=0){
           while (maxShiftValue >= 0){
               var solveEncrypt = encrypt(str, maxShiftValue)
```

```
                println("Ceaser " + maxShiftValue +  ": " +  solveEncrypt)
                maxShiftValue -= 1 //keep maxShiftValue at positive number and increment to
0 to print properly
            }
        }else{//if maxShiftValue is negative
            while (maxShiftValue <= 0){
                var solveDecrypt = decrypt(encrypt(str, maxShiftValue).toString,
maxShiftValue)
                println("Ceaser " + maxShiftValue +  ": " +  solveDecrypt)
                maxShiftValue += 1 //keep maxShiftValue at negative number and increment to
0 to print properly
            }
        }
    }
```

As you can see, the while loop is used to iterate the 'maxShiftValue' and show what the encrypted or decrypted word will be at the 'maxShiftValue' amount of shifts.

c.) Creating a Caesar Cipher to encrypt, decrypt, and solve is now completed with these steps.

Introduction to the Caesar Cipher

To begin this assignment, I did not start with any of the languages listed. I started with the object-oriented language: Python. I did this not only to get a proper understanding of the project's requirements but also to test different methods for coding a caesar cipher, as I have never coded it before. This step was incredibly beneficial, in my opinion, when starting my first procedural programming language, as it gave me a better understanding of how the cipher works, what different parts of the equation will do, and allowed me to focus on figuring out the syntax of a language, rather than continuously test my code to make sure the requirements and test cases were correctly being printed.

List of Google Searches:

- https://www.scaler.com/topics/caesar-cipher-python/
- https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/
- https://www.javatpoint.com/caesar-cipher-in-python
- https://likegeeks.com/python-caesar-cipher/
- https://www.youtube.com/watch?v=JEsUlx0Ps9k

Python code:

```
def encrypt(str, shiftAmount):
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    encrypted = ""
    for character in str:
# iterate the letters through the message
        if character in alphabet:
# check if letter is in alphabet (no special characters)
            position = (alphabet.find(character) + shiftAmount) % len(alphabet)
# formula to check for letter to ciphertext equivalent for encryption (+ sign moves
forwards in shift)
            encrypted = encrypted + alphabet[position]
# creates the cipher letter and puts it in its list to make the word
        else:
            encrypted = encrypted + character
# takes whatever letter is not in the 'alphabet' to add it into the encrypted
message as its original value
    return encrypted
# finish and return to print statement
#
def decrypt(str, shiftAmount):
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    decrypted = ""
    for character in str:
        if character in alphabet:
            position = (alphabet.find(character) - shiftAmount) % len(alphabet)   #
formula to check for a letter to ciphertext equivalent for decryption (- sign moves
```

```
back in shift)
            decrypted = decrypted + alphabet[position]
        else:
            decrypted = decrypted + character
    return decrypted



def encrypt(message, shifted):
    encrypted = ""
    for i in range(len(message)): #iterates through the message
        message_char = message[i]
        encrypted += chr((ord(message_char) + shifted-65) % 26 + 65)
    return encrypted


def decrypt(message, shifted):
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    decrypted = ""
    for character in message:
        if character in alphabet:
            position = alphabet.find(character)
            new_pos = (position + shifted) % 26
            new_char = alphabet[new_pos]
            decrypted += new_char
        else:
            decrypted += character

    return decrypted


def solve(str, maxShiftValue):
    while maxShiftValue >=0:                                        #iterate
through maxShiftValue until 0 but will print 0
        solve = encrypt(str, maxShiftValue)                         #whatever
shift it's at goes through encrypt function for desired output in the assignment
(decrypt flips the list)
        message = print("Caesar",maxShiftValue ,":", solve.upper())     #print the
respective message
        maxShiftValue -= 1                                          #-1 on
maxShiftValue, so the while loop doesn't continuously loop
    and returns a message

str = input("Please enter a text you wish to cipher: \n").strip().upper()
shiftAmount = int(input("Please enter the shift pattern: \n"))
decision = input("What cipher do you want: E/D/S: \n").strip().lower()
```

```
if decision == "e":
    print("Cipher encryption: " + encrypt(str, shiftAmount))

if decision == "d":
    print("Cipher decryption: " + decrypt(str, shiftAmount))

if decision =="s":
    print(solve("HAL", 26))
```

Procedural Programming Language Code/Test Cases

Scala

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


        println("Encrypted: " +  encrypt(str, shiftAmount))
        println("Decrypted: " + decrypt(str, shiftAmount))
        println("Solve:")
        solve(str)


    }//ends main

            //define variables for function
    def encrypt( str:String, shiftAmount:Int ) : String = {
        var encrypted = "" //you need to have an empty string or else it wont run
        for (position <- str){        //no ord() function = use .toInt      //iterate
through the initial message
            if (position.toInt >= 65 && position.toInt <= 90) { // ASCII *Uppercase* letter
in alphabet
                encrypted += (((position.toInt - 65 + shiftAmount) % 26 + 26) % 26 +
65).toChar  //test for positive and negative numbers
            } else {
//.toChar takes the ASCII number and converts to number
                                                                    //^+26 allows for
loop in alphabet
                encrypted += position  //adds if not a character in alphabet
            }
        }
         encrypted



    }

    def decrypt(str:String, shiftAmount:Int) : String = {
        var encryptedstring = encrypt(str, shiftAmount)
        var decrypted = ""


        for (position <- encryptedstring){
            if (position.toInt >= 65 && position.toInt <= 90) {
                decrypted += (((position.toInt - 65 - shiftAmount) % 26 + 26) % 26 +
65).toChar
            } else {                        //minus reverses the shift and creates decrypt
```

```
                decrypted += position
            }
        }
        decrypted
    }

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

         //if maxShiftValue is positive
        if (maxShiftValue >=0){
            while (maxShiftValue >= 0){
                var solveEncrypt = encrypt(str, maxShiftValue)
                println("Ceaser " + maxShiftValue +  ": " +  solveEncrypt)
                maxShiftValue -= 1 //keep maxShiftValue at positive number and increment to
0 to print properly
            }
        }else{//if maxShiftValue is negative
            while (maxShiftValue <= 0){
                var solveDecrypt = decrypt(encrypt(str, maxShiftValue).toString,
maxShiftValue)//figure out why only decrypting it doesn't work //because i was passing a
regular str rather than the function encrypt (str, maxshiftvalue)
                println("Ceaser " + maxShiftValue +  ": " +  solveDecrypt)
                maxShiftValue += 1 //keep maxShiftValue at negative number and increment to
0 to print properly
            }
        }
    }
}//ends CeaserCipher
```

Test Cases (Scala)

str = "HAL" shiftAmount = 3 maxShiftValue = 26	str = "POLAND SPRINGS" shiftAmount = 27 maxShiftValue = -26	str = "!HOWDY!" shiftAmount = -4 maxShiftValue = -28
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	Encrypted: QPMBOE TQSJOHT Decrypted: POLAND SPRINGS Solve: Ceaser -26: POLAND SPRINGS Ceaser -25: QPMBOE TQSJOHT Ceaser -24: RQNCPF URTKPIU Ceaser -23: SRODQG VSULQJV Ceaser -22: TSPERH WTVMRKW Ceaser -21: UTQFSI XUWNSLX Ceaser -20: VURGTJ YVXOTMY Ceaser -19: WVSHUK ZWYPUNZ	Encrypted: !DKSZU! Decrypted: !HOWDY! Solve: Ceaser -28: !FMUBW! Ceaser -27: !GNVCX! Ceaser -26: !HOWDY! Ceaser -25: !IPXEZ! Ceaser -24: !JQYFA! Ceaser -23: !KRZGB! Ceaser -22: !LSAHC! Ceaser -21: !MTBID!

```
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 -18: XWTIVL AXZQVOA
Ceaser -17: YXUJWM BYARWPB
Ceaser -16: ZYVKXN CZBSXQC
Ceaser -15: AZWLYO DACTYRD
Ceaser -14: BAXMZP EBDUZSE
Ceaser -13: CBYNAQ FCEVATF
Ceaser -12: DCZOBR GDFWBUG
Ceaser -11: EDAPCS HEGXCVH
Ceaser -10: FEBQDT IFHYDWI
Ceaser -9: GFCREU JGIZEXJ
Ceaser -8: HGDSFV KHJAFYK
Ceaser -7: IHETGW LIKBGZL
Ceaser -6: JIFUHX MJLCHAM
Ceaser -5: KJGVIY NKMDIBN
Ceaser -4: LKHWJZ OLNEJCO
Ceaser -3: MLIXKA PMOFKDP
Ceaser -2: NMJYLB QNPGLEQ
Ceaser -1: ONKZMC ROQHMFR
Ceaser 0: POLAND SPRINGS
```

```
Ceaser -20: !NUCJE!
Ceaser -19: !OVDKF!
Ceaser -18: !PWELG!
Ceaser -17: !QXFMH!
Ceaser -16: !RYGNI!
Ceaser -15: !SZHOJ!
Ceaser -14: !TAIPK!
Ceaser -13: !UBJQL!
Ceaser -12: !VCKRM!
Ceaser -11: !WDLSN!
Ceaser -10: !XEMTO!
Ceaser -9: !YFNUP!
Ceaser -8: !ZGOVQ!
Ceaser -7: !AHPWR!
Ceaser -6: !BIQXS!
Ceaser -5: !CJRYT!
Ceaser -4: !DKSZU!
Ceaser -3: !ELTAV!
Ceaser -2: !FMUBW!
Ceaser -1: !GNVCX!
Ceaser 0: !HOWDY!
```

Pascal

```
program ceaserCipher;

var
   str: string;
   shiftAmount, maxShiftValue: integer;

function encrypt(str: string; shiftAmount: integer): string; (*need string as that will be
what I am returning*)
var
    character: integer;
begin


    (*Was having trouble figuring out why I could not assign a variable to
'ord(str[character]', decided to leave it as is*)
    for character := 1 to length(str) do
        if (ord(str[character]) >= 65) and (ord(str[character]) <= 90) then (*Need to index
str now with str[character] to get the letter's number value*)
            begin   (*Did not know why I couldn't assign negatives values to shift, with the
mod 26 + 26, it allows negative of all numbers even after 26 *)
                str[character] := chr(((ord(str[character]) - 65 + shiftAmount) mod 26 +
26)mod 26 + 65)
            end
        else
            begin
                str := str[character] (*You can concatenate a string this way*)
            end;
    encrypt := str (*The last expression is its return, you don't need to explicitly say
return*)

end;


function decrypt(str: string; shiftAmount: integer): string;
var
    character: integer;
    decryptedString: String;
begin
decryptedString := encrypt(str, shiftAmount);

    for character := 1 to length(decryptedString) do
        if (ord(decryptedString[character]) >= 65) and (ord(decryptedString[character]) <=
90) then
            begin
                decryptedString[character] := chr(((ord(decryptedString[character]) - 65 -
shiftAmount) mod 26 + 26)mod 26 + 65) (*change to -shiftAmount so that it goes left in the
alphabet*)
            end
        else
            begin
```

```
                decryptedString := decryptedString[character]
            end;
    decrypt := decryptedString

end;


function solve(str: string; maxShiftValue: integer): string ;
var
   solved: string;
begin
    if maxShiftValue >=0 then
        while maxShiftValue >= 0 do
            begin
                solved := encrypt(str, maxShiftValue); (*Call the encrypt function to assign
it, makes it easier to call*)
                writeln('Ceaser ',maxShiftValue ,': ', solved);
                maxShiftValue := maxShiftValue - 1; (*maxShiftValue is 26 and wants to get
to 0*)
            end
    else
        while maxShiftValue <= 0 do
            begin
                solved := decrypt(encrypt(str, maxShiftValue), maxShiftValue);
                writeln('Ceaser ',maxShiftValue ,': ', solved);
                maxShiftValue := maxShiftValue + 1;(*maxShiftValue is -26 and wants to get
to 0*)
            end
end;

begin
    str := 'HAL';
    shiftAmount := 3;
    maxShiftValue := 26;


    writeln('Encrypted: ', encrypt(str, shiftAmount));
    writeln('Decrypted: ', decrypt(str, shiftAmount));
    writeln('Solved:');
    writeln(solve(str, maxShiftValue));
end.
```

Test Cases (Pascal)

str := 'HAL';
shiftAmount := 3;
maxShiftValue := 26;
	str := 'CEASER';
	shiftAmount := 30;
	maxShiftValue := 35;
		str := 'CONNOR';
		shiftAmount := -5;
		maxShiftValue := -30;

Encrypted: KDO	Encrypted: GIEWIV	Encrypted: XJIIJM

```
Decrypted: HAL
Solved:
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
Ceaser 0: HAL
```

```
Decrypted: CEASER
Solved:
Ceaser 35: LNJBNA
Ceaser 34: KMIAMZ
Ceaser 33: JLHZLY
Ceaser 32: IKGYKX
Ceaser 31: HJFXJW
Ceaser 30: GIEWIV
Ceaser 29: FHDVHU
Ceaser 28: EGCUGT
Ceaser 27: DFBTFS
Ceaser 26: CEASER
Ceaser 25: BDZRDQ
Ceaser 24: ACYQCP
Ceaser 23: ZBXPBO
Ceaser 22: YAWOAN
Ceaser 21: XZVNZM
Ceaser 20: WYUMYL
Ceaser 19: VXTLXK
Ceaser 18: UWSKWJ
Ceaser 17: TVRJVI
Ceaser 16: SUQIUH
Ceaser 15: RTPHTG
Ceaser 14: QSOGSF
Ceaser 13: PRNFRE
Ceaser 12: OQMEQD
Ceaser 11: NPLDPC
Ceaser 10: MOKCOB
Ceaser 9: LNJBNA
Ceaser 8: KMIAMZ
Ceaser 7: JLHZLY
Ceaser 6: IKGYKX
Ceaser 5: HJFXJW
Ceaser 4: GIEWIV
Ceaser 3: FHDVHU
Ceaser 2: EGCUGT
Ceaser 1: DFBTFS
Ceaser 0: CEASER
```

```
Decrypted: CONNOR
Solved:
Ceaser -30: YKJJKN
Ceaser -29: ZLKKLO
Ceaser -28: AMLLMP
Ceaser -27: BNMMNQ
Ceaser -26: CONNOR
Ceaser -25: DPOOPS
Ceaser -24: EQPPQT
Ceaser -23: FRQQRU
Ceaser -22: GSRRSV
Ceaser -21: HTSSTW
Ceaser -20: IUTTUX
Ceaser -19: JVUUVY
Ceaser -18: KWVVWZ
Ceaser -17: LXWWXA
Ceaser -16: MYXXYB
Ceaser -15: NZYYZC
Ceaser -14: OAZZAD
Ceaser -13: PBAABE
Ceaser -12: QCBBCF
Ceaser -11: RDCCDG
Ceaser -10: SEDDEH
Ceaser -9: TFEEFI
Ceaser -8: UGFFGJ
Ceaser -7: VHGGHK
Ceaser -6: WIHHIL
Ceaser -5: XJIIJM
Ceaser -4: YKJJKN
Ceaser -3: ZLKKLO
Ceaser -2: AMLLMP
Ceaser -1: BNMMNQ
Ceaser 0: CONNOR
```

FREE BASIC

```
function encrypt(mainString AS STRING, shiftAmount AS INTEGER) AS STRING 'will be returning
a string
    dim character as integer ' all variables assigned inside of the functions are only local
variables
    dim encrypted as string


    for character = 0 to len(mainString)-1 '-1 because for some reason, when character was
printed I would receive a 0 at the end even though the length was 3 for len(mainString)
        if mainString[character] >= 65 and mainString[character] <= 90 then 'not entirely
sure why when I call for the mainString[character] it gives me the integer value rather than
the string value
            encrypted += chr(((mainString[character] - 65 + shiftAmount) mod 26 + 26) mod 26
+ 65)
        else
            encrypted += chr(mainString[character])
        end if
    next
    return encrypted
end function


function decrypt(mainString AS STRING, shiftAmount AS INTEGER) AS STRING
    dim character as integer
    dim decrypted as string
    dim encryptedString as string

    encryptedString = encrypt(mainString, shiftAmount)

    for character = 0 to len(encryptedString)-1
        if encryptedString[character] >= 65 and encryptedString[character] <= 90 then
            decrypted += chr(((encryptedString[character] - 65 - shiftAmount) mod 26 + 26)
mod 26 + 65) 'subtract due to decrypt
        else
            decrypted += chr(encryptedString[character])
        end if
    next 'i find this weird to end a for-loop
    return decrypted
end function


function solve(mainString AS STRING, maxShiftValue AS INTEGER) AS STRING
   dim solveEncypt as string
   dim solveDecrypt as string

    if maxShiftValue >= 0 then 'if maxShiftValue is positive
        while maxShiftValue >= 0
            solveEncypt = encrypt(mainString, maxShiftValue)
            print "Ceaser",maxShiftValue,":",solveEncypt
            maxShiftValue -= 1 'subtract maxShiftValue to get it to 0
```

```
        wend

    else
        while maxShiftValue <= 0  'if maxShiftValue is negative
            solveDecrypt = decrypt(encrypt(mainString, maxShiftValue), maxShiftValue)
            print "Ceaser",maxShiftValue,":",solveDecrypt
            maxShiftValue += 1 'add maxShiftValue to get it to 0
        wend

    end if

end function



'------------Main function------------'
dim mainString as string 'must create and assign to a primitive variable
dim shiftAmount as integer
dim maxShiftValue as integer

dim encrypted as string
dim decrypted as string

mainString = "HAL" 'I wonder if you can just set this equal to when assigning it to its
primitive variable
shiftAmount = 3
maxShiftValue = 26



encrypted = encrypt(mainString, shiftAmount) 'you have to assign these calls so that they
can be printed out correctly
decrypted = decrypt(mainString, shiftAmount)


print "Encrypted:", encrypted
print "Decrypted: ", decrypted
print "Solve:"
solve(mainString, maxShiftValue)
```

Test Cases (FREE BASIC)

mainString = "HAL" shiftAmount = 3 maxShiftValue = 26	mainString = "HAL LAH" shiftAmount = -3 maxShiftValue = -30	mainString = "TONY HAWK'S PRO SKATER 3" shiftAmount = 28 maxShiftValue = 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 :	Encrypted: EXI IXE Decrypted: HAL LAH Solve: Ceaser -30 : DWH HWD Ceaser -29 : EXI IXE Ceaser -28 : FYJ JYF Ceaser -27 : GZK KZG Ceaser -26 : HAL LAH Ceaser -25 : IBM MBI Ceaser -24 : JCN NCJ Ceaser -23 : KDO ODK Ceaser -22 : LEP PEL Ceaser -21 : MFQ QFM Ceaser -20 : NGR RGN Ceaser -19 : OHS SHO Ceaser -18 : PIT TIP Ceaser -17 : QJU UJQ Ceaser -16 : RKV VKR Ceaser -15 : SLW WLS Ceaser -14 : TMX XMT Ceaser -13 : UNY YNU Ceaser -12 : VOZ ZOV Ceaser -11 :	Encrypted: VQPA JCYM'U RTQ UMCVGT 3 Decrypted: TONY HAWK'S PRO SKATER 3 Solve: Ceaser 30 : XSRC LEAO'W TVS WOEXIV 3 Ceaser 29 : WRQB KDZN'V SUR VNDWHU 3 Ceaser 28 : VQPA JCYM'U RTQ UMCVGT 3 Ceaser 27 : UPOZ IBXL'T QSP TLBUFS 3 Ceaser 26 : TONY HAWK'S PRO SKATER 3 Ceaser 25 : SNMX GZVJ'R OQN RJZSDQ 3 Ceaser 24 : RMLW FYUI'Q NPM QIYRCP 3 Ceaser 23 : QLKV EXTH'P MOL PHXQBO 3 Ceaser 22 : PKJU DWSG'O LNK OGWPAN 3 Ceaser 21 : OJIT CVRF'N KMJ NFVOZM 3 Ceaser 20 : NIHS BUQE'M JLI MEUNYL 3 Ceaser 19 : MHGR ATPD'L IKH LDTMXK 3 Ceaser 18 : LGFQ ZSOC'K HJG KCSLWJ 3 Ceaser 17 : KFEP YRNB'J GIF JBRKVI 3 Ceaser 16 : JEDO XQMA'I FHE IAQJUH 3 Ceaser 15 : IDCN WPLZ'H EGD HZPITG 3 Ceaser 14 : HCBM VOKY'G DFC GYOHSF 3 Ceaser 13 : GBAL UNJX'F CEB FXNGRE 3 Ceaser 12 :

```
OHS
Ceaser          6           :
NGR
Ceaser          5           :
MFQ
Ceaser          4           :
LEP
Ceaser          3           :
KDO
Ceaser          2           :
JCN
Ceaser          1           :
IBM
Ceaser          0           :
HAL
```

```
WPA APW
Ceaser         -10          :
XQB BQX
Ceaser         -9           :
YRC CRY
Ceaser         -8           :
ZSD DSZ
Ceaser         -7           :
ATE ETA
Ceaser         -6           :
BUF FUB
Ceaser         -5           :
CVG GVC
Ceaser         -4           :
DWH HWD
Ceaser         -3           :
EXI IXE
Ceaser         -2           :
FYJ JYF
Ceaser         -1           :
GZK KZG
Ceaser          0           :
HAL LAH
```

```
FAZK TMIW'E BDA EWMFQD 3
Ceaser          11          :
EZYJ SLHV'D ACZ DVLEPC 3
Ceaser          10          :
DYXI RKGU'C ZBY CUKDOB 3
Ceaser          9           :
CXWH QJFT'B YAX BTJCNA 3
Ceaser          8           :
BWVG PIES'A XZW ASIBMZ 3
Ceaser          7           :
AVUF OHDR'Z WYV ZRHALY 3
Ceaser          6           :
ZUTE NGCQ'Y VXU YQGZKX 3
Ceaser          5           :
YTSD MFBP'X UWT XPFYJW 3
Ceaser          4           :
XSRC LEAO'W TVS WOEXIV 3
Ceaser          3           :
WRQB KDZN'V SUR VNDWHU 3
Ceaser          2           :
VQPA JCYM'U RTQ UMCVGT 3
Ceaser          1           :
UPOZ IBXL'T QSP TLBUFS 3
Ceaser          0           :
TONY HAWK'S PRO SKATER 3
```

FORTRAN

```
program main

    character(len=3) :: mainString = "HAL" !need to assign length of string
    integer :: shiftAmount = 3


    call encrypt(mainString, shiftAmount)


    !since we are doing subroutines we include this in the contains as it is still in the
main
    contains

        subroutine encrypt(mainString, shiftAmount)

            character(len=3) :: mainString
            character(len=3) :: encrypted
            character(len=1) :: y !only needs to be length of 1 due to it only looking for
singular characters
            integer :: shiftAmount
            integer :: characters
            integer :: x

            characters = 1

            do x = len(mainString), characters, -1
                if (ichar(mainString(x:x)) >= 65 .and. ichar(mainString(x:x)) <= 90) then
!indexing with (x:x) to get position of characters
                    y = achar(mod((mod((ichar(mainString(x:x)) - 65 + shiftAmount),26) +
26), 26) + 65)
                    encrypted = y // encrypted !encrypted //y was not printing anything out,
so I have to decrement the loop to go backwards in the string
                else
                    y = achar(ichar(mainString(x:x))) !need to keep the achar(ichar() )
                    encrypted = y // encrypted
                end if
            end do
            print*, "Encrypted: ", encrypted

            call decrypt(encrypted) !call here as it is now
            call solve(mainString, encrypted)


        end subroutine encrypt

!-----------------------------------------------------------------------------------------!

        subroutine decrypt(encryptedString)

            character (len=3) :: encryptedString
```

```
            character (len=3) :: decrypted
            character (len=1) :: y
            integer :: shiftAmount
            integer :: characters
            integer :: x

            characters = 1

            do x = len(encryptedString), characters, -1
                if (ichar(encryptedString(x:x)) >= 65 .and. ichar(encryptedString(x:x)) <=
90) then
                    y = achar(mod((mod((ichar(mainString(x:x)) - 65 - shiftAmount),26) +
26), 26) + 65)
                    decrypted = y // decrypted !dencrypted //y was not printing anything
out, so I have to decrement the loop to go backwards in the string

                else
                    y = achar(ichar(encryptedString(x:x)))
                    decrypted = y // decrypted
                end if
            end do
            print*, "Decrypted: ", decrypted


        end subroutine

!-------------------------------------------------------------------------------------!

        subroutine solve(mainString, encrypted)

            character (len=3) :: mainString
            character (len=3) :: encrypted
            character (len=3) :: solvedEncrypted
            character (len=3) :: solvedDecrypted
            character (len=1) :: c
            character (len=1) :: d
            integer :: maxShiftValue = 26
            integer :: characters
            integer :: shift
            integer :: shifttwo
            integer :: j
            integer :: i




            characters = 1

            !since we are not allowed to make a call a named variable I needed to created
the encrypt and decrypt formulas again

            if (maxShiftValue >= 0) then
```

```
                do shift = maxShiftValue, 0, -1
                    do j = len(mainString), characters, -1
                        if (ichar(mainString(j:j)) >= 65 .and. ichar(mainString(j:j)) <= 90)
then
                            c = achar(mod((mod((ichar(mainString(j:j)) - 65 + shift),26) +
26), 26) + 65)
                            solvedEncrypted = c // solvedEncrypted !encrypted //y was not
printing anything out, so I have to decrement the loop to go backwards in the string
                        else
                            c = achar(ichar(mainString(j:j)))
                            solvedEncrypted = c // solvedEncrypted
                        end if
                    end do
                    print*, "Ceaser: ", shift, " : ", solvedEncrypted

                end do


            else
                do shift = maxShiftValue, 0, +1
                    do j = len(encrypted), characters, -1
                        if (ichar(mainString(j:j)) >= 65 .and. ichar(mainString(j:j)) <= 90)
then
                            c = achar(mod((mod((ichar(mainString(j:j)) - 65 + shift), 26) +
26), 26) + 65)
                            solvedEncrypted = c // solvedDecrypted !encrypted //y was not
printing anything out, so I have to decrement the loop to go backwards in the string
                            do shifttwo = maxShiftValue, 0, +1
                                do i = len(solvedEncrypted), characters, -1
                                    if (ichar(solvedEncrypted(i:i)) >= 65 .and.
ichar(solvedEncrypted(i:i)) <= 90) then
                                        d = achar(mod((mod((ichar(solvedEncrypted(i:i)) - 65
- shifttwo), 26) + 26), 26) + 65)
                                        solvedDecrypted = d // solvedDecrypted !encrypted
//y was not printing anything out, so I have to decrement the loop to go backwards in the
string
                                    end if
                                end do
                            end do
                        else
                            c = achar(ichar(encrypted(j:j)))
                            solvedDecrypted = c // solvedDecrypted
                        end if
                    end do
                    print*, "Ceaser: ", shift, " : ", solvedDecrypted

                end do

            end if

        end subroutine
end program main
```

Test Cases (FORTRAN)

character(len=3) :: mainString = "HAL" integer :: shiftAmount = 3 integer :: maxShiftValue = 26	character(len=7) :: mainString = "CON NOR" integer :: shiftAmount = -3 integer :: maxShiftValue = 28	character(len=3) :: mainString = "@AB" integer :: shiftAmount = -3 integer :: maxShiftValue = -26
Encrypted: KDO Decrypted: HAL 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 Ceaser: 0 : HAL	Encrypted: ZLK KLO Decrypted: CON NOR Ceaser: 28 : EQP PQT Ceaser: 27 : DPO OPS Ceaser: 26 : CON NOR Ceaser: 25 : BNM MNQ Ceaser: 24 : AML LMP Ceaser: 23 : ZLK KLO Ceaser: 22 : YKJ JKN Ceaser: 21 : XJI IJM Ceaser: 20 : WIH HIL Ceaser: 19 : VHG GHK Ceaser: 18 : UGF FGJ Ceaser: 17 : TFE EFI Ceaser: 16 : SED DEH Ceaser: 15 : RDC CDG Ceaser: 14 : QCB BCF Ceaser: 13 : PBA ABE Ceaser: 12 : OAZ ZAD Ceaser: 11 : NZY YZC Ceaser: 10 : MYX XYB Ceaser: 9 : LXW WXA Ceaser: 8 : KWV VWZ Ceaser: 7 : JVU UVY Ceaser: 6 : IUT TUX Ceaser: 5 : HTS STW Ceaser: 4 : GSR RSV Ceaser: 3 : FRQ QRU Ceaser: 2 : EQP PQT Ceaser: 1 : DPO OPS Ceaser: 0 : CON NOR	Encrypted: @XY Decrypted: @AB Ceaser: -26 : @AB Ceaser: -25 : @BC Ceaser: -24 : @CD Ceaser: -23 : @DE Ceaser: -22 : @EF Ceaser: -21 : @FG Ceaser: -20 : @GH Ceaser: -19 : @HI Ceaser: -18 : @IJ Ceaser: -17 : @JK Ceaser: -16 : @KL Ceaser: -15 : @LM Ceaser: -14 : @MN Ceaser: -13 : @NO Ceaser: -12 : @OP Ceaser: -11 : @PQ Ceaser: -10 : @QR Ceaser: -9 : @RS Ceaser: -8 : @ST Ceaser: -7 : @TU Ceaser: -6 : @UV Ceaser: -5 : @VW Ceaser: -4 : @WX Ceaser: -3 : @XY Ceaser: -2 : @YZ Ceaser: -1 : @ZA Ceaser: 0 : @AB

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CEASERCIPHER.
DATA DIVISION.
    WORKING-STORAGE SECTION.

        77 mainString PIC A(3) value "HAL". *If you are to change the string, you have to
also change the length A(3)
        77 shiftAmount PIC 999 value 3. * the amount of 9's account for the length of the
integer
        77 maxShiftValue PIC 999 value 26.


        77 characterized PIC 999 value 1. *only accounting for a single character at a time
so length is 1
        77 mainStringLength PIC 999.

        77 encrypted PIC A(3).
        77 y PIC A(1).


        77 decrypted PIC A(3).
        77 x PIC A(1).

        77 encryptedSolved PIC A(3).
        77 i PIC A(1).

        77 shift PIC 999 value 1.




PROCEDURE DIVISION.
    INSPECT mainString TALLYING mainStringLength FOR CHARACTERS *gets the length of
mainString

    *Encrypt

    perform VARYING characterized FROM 1 by 1 until characterized = mainStringLength + 1
*Was not printing out the first letter of the string

        if (function ord(mainString(characterized:characterized)) >= 66 and function
ord(mainString(characterized:characterized)) <= 90) *66 is now 'A', because this does not
start at 0
            move function char(function mod((function mod((function
ord(mainString(characterized:characterized)) - 66 + shiftAmount),26) + 26), 26) + 66) to
encrypted(characterized:characterized)
        else
            move function char(function ord(mainString(characterized:characterized))) to
encrypted(characterized:characterized)
```

```
        end-if
    end-perform.
    display "Encrypted: "encrypted


    *Decrypt

    perform VARYING characterized FROM 1 by 1 until characterized = mainStringLength + 1
        if (function ord(encrypted(characterized:characterized)) >= 65 and function
ord(encrypted(characterized:characterized)) <= 90)
            move function char(function mod((function mod((function
ord(encrypted(characterized:characterized)) - 66 - shiftAmount),26) + 26), 26) + 66) to
decrypted(characterized:characterized)
        else
            move function char(function ord(encrypted(characterized:characterized))) to
decrypted(characterized:characterized)
        end-if
    end-perform.
    display "Decrypted: "decrypted


    *Solve
    *did not account for negative as I could not figure out how to call a negative number
    *since we are not creating functions, we need to do the encrypt formula

    if maxShiftValue >= 0
        PERFORM VARYING shift FROM 0 BY 1 UNTIL shift > maxShiftValue
            perform VARYING characterized FROM 1 by 1 until characterized = mainStringLength
+ 1

                if (function ord(mainString(characterized:characterized)) >= 65 and function
ord(mainString(characterized:characterized)) <= 90)
                    move function char(function mod((function mod((function
ord(mainString(characterized:characterized)) - 66 + shift),26) + 26), 26) + 66) to
encryptedSolved(characterized:characterized)
                else
                    move function char(function
ord(mainString(characterized:characterized))) to
encryptedSolved(characterized:characterized)

                end-if

            END-PERFORM

            DISPLAY "Caesar " shift " : " encryptedSolved

        END-PERFORM
    ELSE
        DISPLAY "bad"
    END-if

STOP RUN.
```

Test Cases (COBOL)

77 mainString PIC A(3) value "HAL". 77 shiftAmount PIC 999 value 3. 77 maxShiftValue PIC 999 value 26.	77 mainString PIC A(8) value "$SHMONEY". 77 shiftAmount PIC 999 value 26. 77 maxShiftValue PIC 999 value 28.	77 mainString PIC A(9) value "!HAL LAH!". 77 shiftAmount PIC 999 value 3. 77 maxShiftValue PIC 999 value 26.
Encrypted: KDO Decrypted: HAL Caesar 000 : HAL Caesar 001 : IBM Caesar 002 : JCN Caesar 003 : KDO Caesar 004 : LEP Caesar 005 : MFQ Caesar 006 : NGR Caesar 007 : OHS Caesar 008 : PIT Caesar 009 : QJU Caesar 010 : RKV Caesar 011 : SLW Caesar 012 : TMX Caesar 013 : UNY Caesar 014 : VOZ Caesar 015 : WPA Caesar 016 : XQB Caesar 017 : YRC Caesar 018 : ZSD Caesar 019 : ATE Caesar 020 : BUF Caesar 021 : CVG Caesar 022 : DWH Caesar 023 : EXI Caesar 024 : FYJ Caesar 025 : GZK Caesar 026 : HAL	Encrypted: $SHMONEY Decrypted: $SHMONEY Caesar 000 : $SHMONEY Caesar 001 : $TINPOFZ Caesar 002 : $UJOQPGA Caesar 003 : $VKPRQHB Caesar 004 : $WLQSRIC Caesar 005 : $XMRTSJD Caesar 006 : $YNSUTKE Caesar 007 : $ZOTVULF Caesar 008 : $APUWVMG Caesar 009 : $BQVXWNH Caesar 010 : $CRWYXOI Caesar 011 : $DSXZYPJ Caesar 012 : $ETYAZQK Caesar 013 : $FUZBARL Caesar 014 : $GVACBSM Caesar 015 : $HWBDCTN Caesar 016 : $IXCEDUO Caesar 017 : $JYDFEVP Caesar 018 : $KZEGFWQ Caesar 019 : $LAFHGXR Caesar 020 : $MBGIHYS Caesar 021 : $NCHJIZT Caesar 022 : $ODIKJAU Caesar 023 : $PEJLKBV Caesar 024 : $QFKMLCW Caesar 025 : $RGLNMDX Caesar 026 : $SHMONEY Caesar 027 : $TINPOFZ Caesar 028 : $UJOQPGA	Encrypted: !KDO ODK! Decrypted: !HAL LAH! Caesar 000 : !HAL LAH! Caesar 001 : !IBM MBI! Caesar 002 : !JCN NCJ! Caesar 003 : !KDO ODK! Caesar 004 : !LEP PEL! Caesar 005 : !MFQ QFM! Caesar 006 : !NGR RGN! Caesar 007 : !OHS SHO! Caesar 008 : !PIT TIP! Caesar 009 : !QJU UJQ! Caesar 010 : !RKV VKR! Caesar 011 : !SLW WLS! Caesar 012 : !TMX XMT! Caesar 013 : !UNY YNU! Caesar 014 : !VOZ ZOV! Caesar 015 : !WPA APW! Caesar 016 : !XQB BQX! Caesar 017 : !YRC CRY! Caesar 018 : !ZSD DSZ! Caesar 019 : !ATE ETA! Caesar 020 : !BUF FUB! Caesar 021 : !CVG GVC! Caesar 022 : !DWH HWD! Caesar 023 : !EXI IXE! Caesar 024 : !FYJ JYF! Caesar 025 : !GZK KZG! Caesar 026 : !HAL LAH!

Experience with the Procedural Programming Language

*In order of preference

Scala
Time expected ≈ 7 hours
Time finished ≈ 8 hours

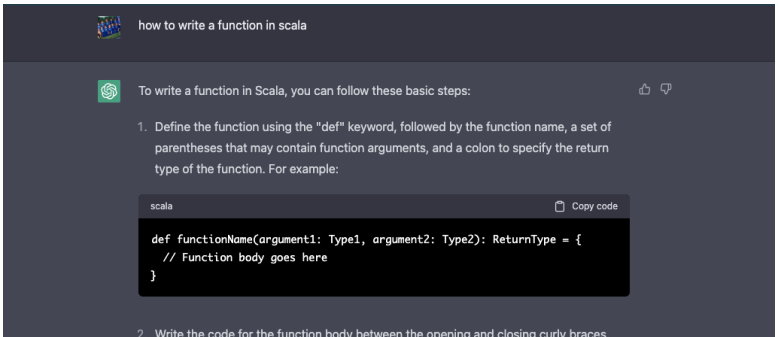
Task	Accomplishments	Issues	Thoughts
Creating the code	Ended up successfully finishing	Trying to complete Scala and Pascal at the same time	I had been working on scala a little and thought I had a good understanding, so I moved to Pascal, and learning two syntaxes at once was not a smart idea. I felt my ideas were always mixed up, and I couldn't concentrate. For example, if my Scala code could accept special characters but pascal couldn't, but Scala wouldn't be shifting correctly, I would try to solve both problems simultaneously.
Changing the value of maxShiftValue	Was able to put maxShiftValue inside of the def(), which allowed it to be altered and iterated through encrypt	If maxShiftValue were outside of the function, it would not increment	A little on the weirder side as to why I can't alter a variable even after passing it as an argument in the def. Received: Expression does not convert to the assignment because the receiver is not assignable.
Formula for the cipher	Fixed with the equation: (((position.toInt - 65 + shiftAmount) % 26 + 26) % 26 + 65).toChar	Whenever I would go over 26 (the end of the alphabet), it would start printing out symbols after ‘Z’ or, if decrypting, before ‘A’	The original equation from my python script was working. Still, I had to modify it with the %26 + 26 so that it would stay in between the alphabet at all times

Task	Accomplishments	Issues	Thoughts
Create the concatenation	Created it using the += syntax with the beginning function: def encrypt(str:String, shiftAmount:Int) : String = {	When attempting to learn concatenation, chatGPT suggested I use the STRINGBUILDER while defining my function instead of the String	I first started looking to this STRINGBUILDER and would also need the: var encrypted = new StringBuilder, feature. However, after trying other methods, I realized I could change the function = String and concatenate by +=, just like how my python and other languages would do it
Thoughts			Scala was definitely the easiest to learn, but this was technically my first language out of the PPLs and with the mistake of also doing Pascal at the same time, it would get confusing at some times - Was definitely the easiest for me to comprehend and learn. Made me think of a python/java syntax - I very much enjoyed the brackets - The compiler was very up to date a lot more descriptive on the error checking in this language than it is in some other ones

List of Google Searches:

Function:

- https://www.tutorialspoint.com/scala/scala_functions.htm



Concatenation:

- https://www.educative.io/answers/what-are-operators-in-scala

Conditional Statements:

- https://www.tutorialspoint.com/scala/scala_for_loop.htm
- https://www.javatpoint.com/scala-conditional-expressions
- https://www.tutorialspoint.com/scala/scala_while_loop.htm

Pascal
Time expected ≈ 8:30 hours
Time finished ≈ 6 hours

Task	Accomplishments	Issues	Thoughts
Comment the code	Found out it was '(* *)' to comment	When I would do '⌘/' as that is the command to comment the highlighted code on a mac, it was giving me '--' which is not right and would throw errors	I think it was just an IDE problem, but very annoying because now I had to do research on how to comment code

Print out the caesar cipher correctly	Was not able to change	Prints an extra mainString
		Example:
		
		`Ceaser 5: MFQ`
		`Ceaser 4: LEP`
		`Ceaser 3: KDO`
		`Ceaser 2: JCN`
		`Ceaser 1: IBM`
		`Ceaser 0: HAL`
		`HAL`
			Out of convenience for the test cases, I left out the extra mainString being printed. However, after reviewing my code, I have no place where I am just printing out the mainString by Itself, so I am very confused as to why it does this

initializing the variables.			I kept forgetting that I have to initialize the variable first inside the var It was also very weird that the var for the main was on the top and the rest of the function had to be in between the var and the main

Task	Accomplishments	Issues	Thoughts
Creating the code entirely from Scala		I was aiming way to high when trying to complete this language	After completing Scala, I thought that I would have an easier time just implementing the code and trying to change the syntax as I went, However, I found that to be extremely time consuming and complex. Going step by step explained above, was a completely different experience.
Getting the single char in the string	Figured out I would have to index it by doing: str[character]	Unlike Scala, you actually have to get the character position by the for-loop to then call the string with the indexed value from that character	It made complete sense and really showed how the rest of these languages will end up looking like. However, it was very interesting to see that Scalal didn't need to do this
Descriptive variables	Used the long version of the ord(decryptedString[character]) in my equations	For some reason, Pascal was not letting me assign certain parts of the equation to a new variable	I wanted to fix the complexity of the code, however, if I did to newString = ord(decryptedString[character]), I wouldn't be able to call newString
Thoughts			Pascal reminded me the most of Java than the rest of the language. Still, a very easily readable and writable language - The walrus equals were very annoying (:=) only because I am not used to writing this, so I kept forgetting to add the (:) - doing ' ; ' after lines was also an annoying feature as I

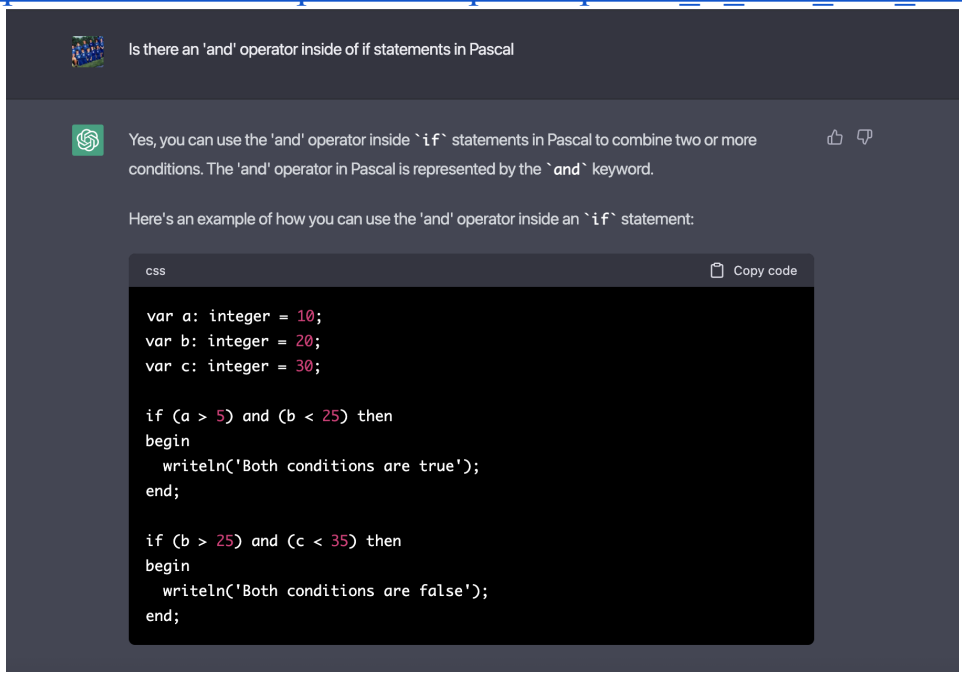
Task	Accomplishments	Issues	Thoughts
			have not had to do that in a javascript - Annoying that you have to declare variables in a var, reminds me of java when initializing variables before using them, but personally I like when I do not have to do that like in python

List of Google Searches:

Conditional Statements:

- https://www.tutorialspoint.com/pascal/pascal_for_do_loop.htm
- https://www.educative.io/answers/how-to-implement-loops-in-pascal
- https://www.tutorialspoint.com/pascal/pascal_if_then_else_statement.ht

m

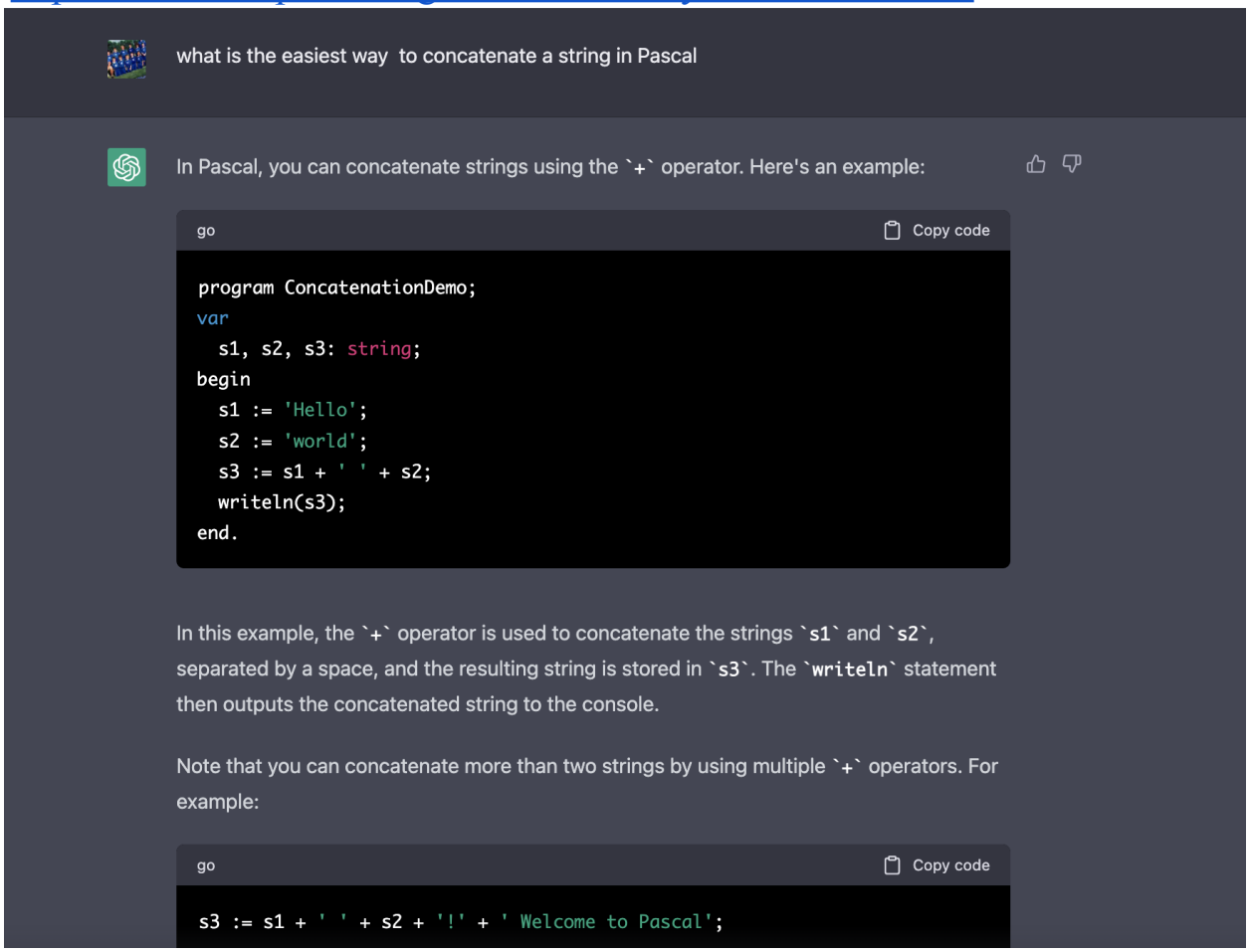


String:

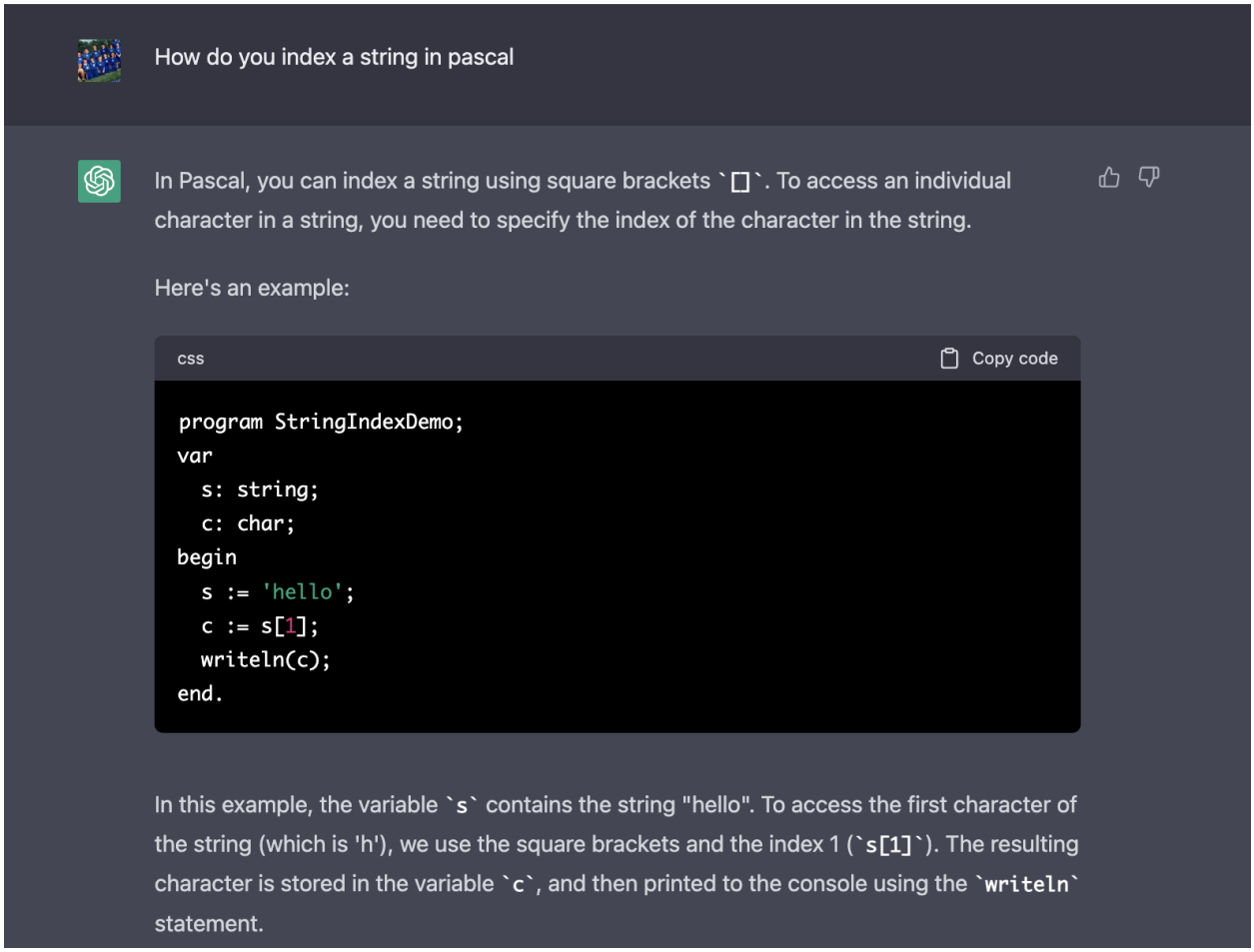
- https://www.tutorialspoint.com/pascal/pascal_strings.htm

Concatenation:

- https://www.freepascal.org/docs-html/rtl/system/concat.html



Indexing the str:



Creating a function:

- https://www.tutorialspoint.com/pascal/pascal_functions.htm
- https://www.educative.io/answers/how-to-declare-a-function-in-pascal

Time expected ≈ 8 hours
Time finished ≈ 8 hours

Task	Accomplishments	Issues	Thoughts
Iterate through the mainString	Successfully made the for loop to iterate through the string and print out the ASCII value	Had to do a -1 to the length of the mainString as 0 kept being printed at the end of the for-loop.	I am honestly not sure why it was giving me the length of 4 for the string “HAL”, however, I did not really dive deep into it but found the solution fairly easily.
Printing for the solve function	Successfully printed out the correct information for the solve function	There are a lot of spaces when printed out.	I would assume that it has to do something with the print function, I am sure there is a way to format the print statement so you can eliminate white spaces.
Creating the beginning if-else statement	Completed a working if-else statement to catch the capitalized alphabet and run it through the integer value of mainString	When I call mainString[character] it prints out an integer rather than a string.	This wasn’t entirely the issue I ran into but was something I needed to put some time in just to realize what was going on because, in normal syntax, that line should be printing out the letter of the indexed value.
Solve function was not accepting my call to decrypt/encrypt	Ended up finding the solution: solveDecrypt = decrypt(str(encrypt(main String, maxShiftValue)), maxShiftValue)	My issue was that I was not passing the encrypt as a String. I had actually used chatGPT to debug this portion	I knew I had to take the encrypted string and pass it through the decrypt to get the negative number, but I am not sure why I needed to make it a string if it was already passing it as a string from the function However, after reviewing my code again, I took that str() function out and my code still seemed to work, I kept it in just in case, but I tried other test cases and nothing seemed to change
Thoughts			FREE BASIC was

Task	Accomplishments	Issues	Thoughts
			surprisingly one of the more enjoyable languages I got to use during this project. I found it very writable, and the perfect not too wordy syntax. - Was very intrigued as I didn't need a main() or anything to do what my main() would be doing - I like the more descriptive ways to end your statements and loops, 'wend' is kinda funny. However, I feel like I enjoy indenting/ {} more than an actual end statement - I feel as if this reminded me a lot of the programing languages we use today

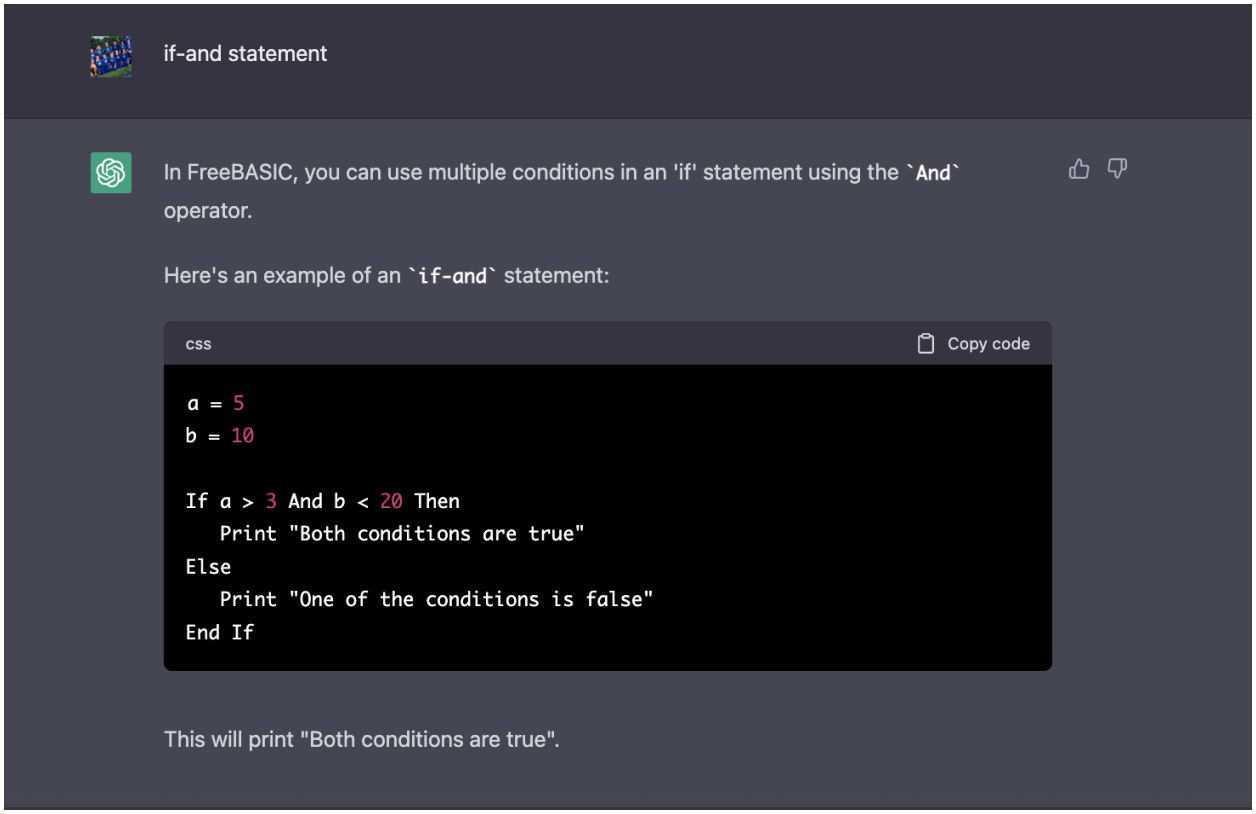
List of Google Searches:

Creating a Function:

- https://www.freebasic.net/wiki/KeyPgFunction

Creating the if and - else statement

- https://www.freebasic.net/wiki/KeyPgIfthen#:~:text=If...Then%20is%20a,if%20a%20condition%20is%20true.



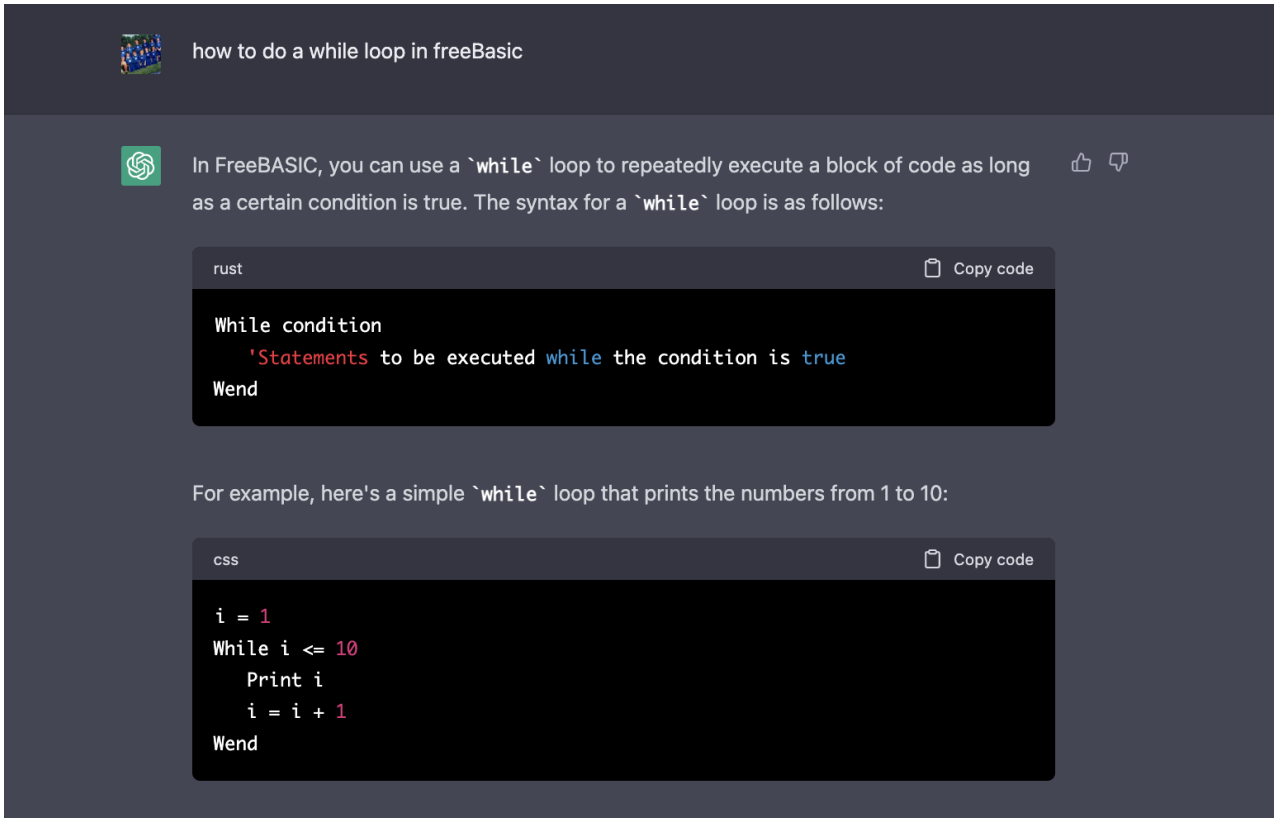
Concatenation:

- https://freebasic.net/forum/viewtopic.php?t=4717

For loop

- https://freebasic.net/forum/viewtopic.php?t=25642

While Loop



Solve calls for encrypt/decrypt:

- Unfortunately, I can not find my history regarding this problem, however, I did use chatGTP to debug this portion of my code

FORTRAN
Time expected ≈ 12 hours
Time finished ≈ 14 hours

Task	Accomplishments	Issues	Thoughts
Implement the equation so that the string gets encrypted		Went step by step and figured it out. Saw that with mod(), you needed to do mod(integer,interger) or, in this case: mod((ichar(mainString(x:x)) - 65 + shiftAmount),26	I feel like it would be easier with doing % instead of (,)
Substrings inside of Fortran		Instead of doing str[characters]. Fortran does not have the	After reading via a link provided, it doesn't make the most sense to me. It

Task	Accomplishments	Issues	Thoughts
		feature to index, so you must index it by (character: character)	Makes sense to take the first and last, but why wouldn't I just be able to just put (character) by itself to get a specific index?
Concatenating strings	Was able to concatenate by doing: encrypted = y // encrypted	With the way I concatenated, the result of encryption would be backward (LAH instead of HAL). I tried switching my concatenation to encrypted = encrypted // y, but that did not work.	Saying my problem out loud made me realize this. In the end, I decided to make my do loop: do x = len(encryptedString), characters, -1 so that it will go through the string backward. The way I fixed this was a different way of looking at a solution. However, it was very annoying knowing that if I had changed my concatenation to encrypted = encrypted // y, I feel like it would have been fixed but was printing me nothing.
Calling subroutines inside subroutines	Able to write and complete my subroutines by using the call function	I was not able to figure out how to call a subroutine. I had tried creating my encrypt, decrypt, and solve inside different functions, for I kept receiving errors during my return.	The problem is that I was using an old version of FORTRAN on Jdoodle. After using ChatGPT and receiving suggestions on how to fix this error, it also resulted in saying me that I am probably using an older version of FORTRAN
Printing during solve	Successfully printed out the correct information for the solve function	There are a large number of spaces when printed out.	I believe this is an issue with assigning a length to the string
Calling a subroutine	Was able to just recreate the formula	I was not allowed to create a variable to assign the call to the encrypt and decrypt	

Task	Accomplishments	Issues	Thoughts
Thoughts		While coding on Jdoodle, my decrypt works and prints HAL, but on OnlineGDP, it does not work. IDEs working with different versions of Fortran?	This syntax was not horrible, but still not my favorite. In my personal opinion, I like a simpler writable syntax rather than a readable one. - Why do I have to set the length of a string? This irritates me - I am not a fan of the concatenation with these older coding languages - print *, is completely dumb, why not just keep it as print

List of Google Searches:

Functions/Subroutines:

- https://web.ics.purdue.edu/~cs154/lectures/lecture014.htm
- https://web.chem.ox.ac.uk/fortran/subprograms.html#:~:text=The%20general%20way%20to%20activate,argument1%2C%20argument2%2C%20.%20.%20.

Substrings:

- https://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vn6s/index.html

Concatenation:

- https://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vn73/index.html#:~:text=The%20only%20character%20operator%20is%20the%20concatenation%20operator%2C%20%2F%2F%20.&text=Concatenate%20a%20with%20z%20..characters%20of%20the%20right%20operand.

For loop/If Statement:

- https://www.tutorialspoint.com/fortran/fortran_do_loop.htm
- https://pages.mtu.edu/~shene/COURSES/cs201/NOTES/chap03/else-if.html

ACHAR/ICHAR:

- https://gcc.gnu.org/onlinedocs/gfortran/ACHAR.html

Mod:

- https://gcc.gnu.org/onlinedocs/gfortran/MOD.html

Ending a subroutine:

5. End your subroutine with the `END SUBROUTINE` statement. This tells the Fortran compiler where your subroutine ends. For example:

less Copy code

```
SUBROUTINE mysub(a, b)
INTEGER :: a, b, c
c = a + b
END SUBROUTINE mysub
```

COBOL
Time expected ≈ 12 hours
Time finished ≈ 8 hours

Task	Accomplishments	Issues	Thoughts
Running test cases	Able to cipher through all alphabetical characters multiple times through the alphabet. Example: shiftValue = 99 will print correctly Was actually able to fix this problem by having the move:*forumla* to encrypted → move:*forumla* to encrypted(characterized:characterized)	If mainString = “HAL!”, the special character is only printed and not the whole word with the ‘!’. However, if mainString = “!HAL”, it prints correctly. If mainString has a space in between, mainString = “HAL HAL”, nothing will print out because of the space	I am not entirely sure why this is occurring because the concatenation is the same (theoretically) across all languages.
Commenting Code		In the Jdoodle IDE, you are actually not allowed to comment using the ‘*’ or ‘/.’	I am honestly unsure how to comment code for this language using JDoodle. Every time I have tried, it will through an error at me
Creating functions	Was able to accomplish this task by creating different loops inside of one function	Within Cobol, I was having a lot of issues creating a function, decided to have my COBOL code all work inside on function and	Multiple forums and websites I had looked up gave me information that unfortunately did not work with my code. Used ChatGPT to have it describe to me how to have a

Task	Accomplishments	Issues	Thoughts
		create different loops for each as it was approved by one student in class by the professor	working function in my code, which also ended in my compiler giving me errors.
Assigning integer variables	I was able to have the multiple numbered integers needed for shiftAmount and maxShiftValue	I am not able to assign a negative number unless I am to add more code to have it define a negative number	
Formula	Was able to change the formula so that it properly displays the correct information, this is due to COBOL indexing at one and not 0	My original formula did not work until I remembered that someone in the class had a comment about cobol starting at one	I find it very weird that program start like this, I guess I am just used to these new, relevant languages
Special Characters	The cipher will take every symbol and not change the character's ASCII value, except the '@' symbol	COBOL's 65th value is the '@' symbol rather than the typical 'A,' so I have to shift the equation by +1 in spots so that the '@' symbols do not show up	Is the '@' symbol considered a separate character in COBOL? I have a special character checker that will realize that '!' and '#' are special characters but not the '@' symbol
Concatenation	Was able to concatenate with the move … to function	A lot of the website resources I searched up gave me a wrong method by saying 'String verb is used to concatenate the strings'	I believe that the String concatenation is used only for Strings. As of right now, my variables are set to chars inside of my formula with the function char().
Thoughts			I am honestly not the biggest fan of the syntax inside of this language, and this will be my least favorite language. - Making a variable is extremely annoying and time-consuming when trying to learn COBOL - I dislike the method (or at least the one I

Task	Accomplishments	Issues	Thoughts
			used) to acquire the length of a string. - Concatenation in COBOL, 'Move ### to ###' is also very annoying. Having your assignment at the end of your code is also very unusual. - Why do I have to have 'DATA DIVISION' and 'PROCEDURE DIVISION' in my code? I tried changing DIVISION and would receive errors. - I find it interesting that 0 is the not the starting index number - I did not enjoy having to say 'function' before my char and ord functions - Having to add a period after almost everything really bugged me, obviously it was put in place probably to have the compiler know you ended the line

List of Google Searches:

Function Issue:

- https://www.ibm.com/docs/en/db2-for-zos/12?topic=function-example-creating-using-user-defined-scalar
- https://www.microfocus.com/documentation/visual-cobol/30pu9/VC_30PU9_for_Eclipse_for_Windows_WH/GUID-B48C9F31-C243-452B-A544-9850B94496A5.html

Variables:

- https://www.geeksforgeeks.org/variables-in-cobol/

Length():

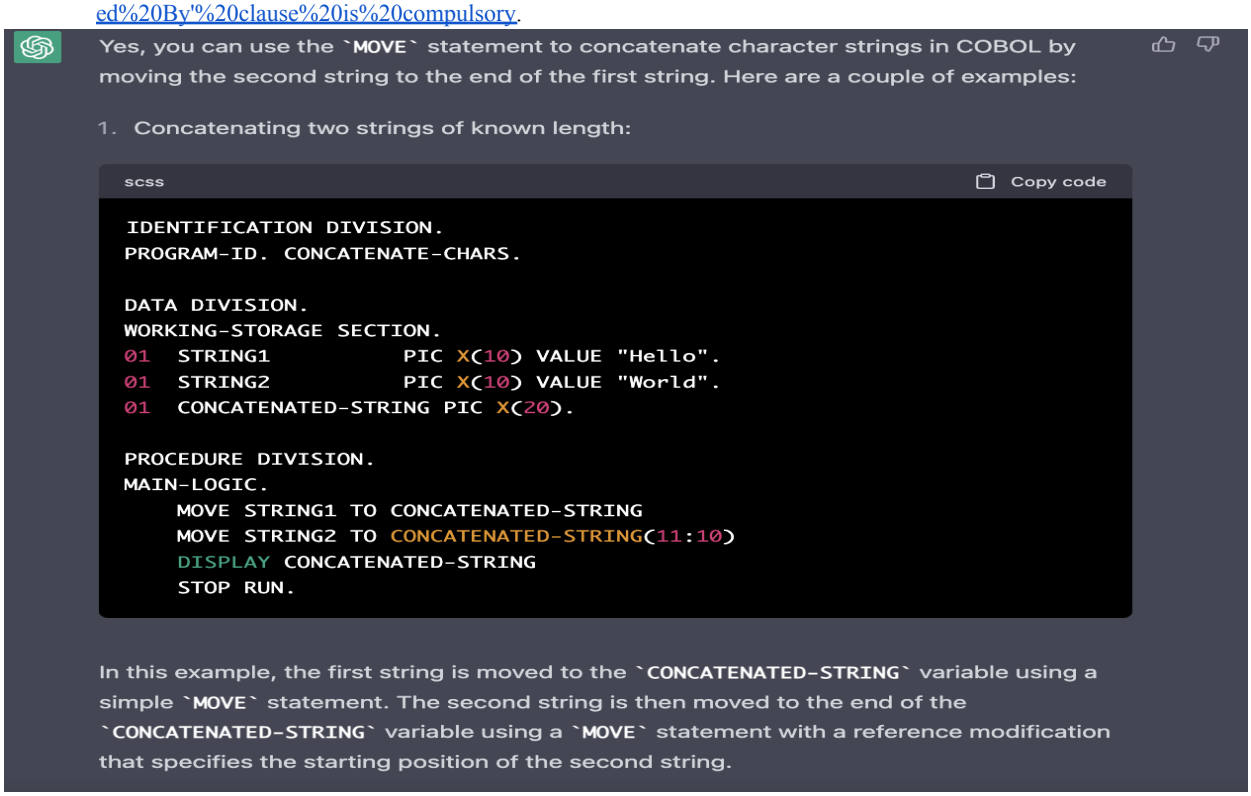
- https://www.mainframestechhelp.com/tutorials/cobol/inspect-tallying-statement.htm

Perform VARYING - until:

- https://www.mainframestechhelp.com/tutorials/cobol/perform-varying.htm

Concatenation:

- https://www.tutorialspoint.com/cobol/cobol_string_handling.htm#:~:text=String%20verb%20is%20used%20to,Delimited%20By'%20clause%20is%20compulsory.



Yes, you can use the `MOVE` statement to concatenate character strings in COBOL by moving the second string to the end of the first string. Here are a couple of examples:

1. Concatenating two strings of known length:

scss — Copy code

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONCATENATE-CHARS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  STRING1           PIC X(10) VALUE "Hello".
01  STRING2           PIC X(10) VALUE "World".
01  CONCATENATED-STRING PIC X(20).

PROCEDURE DIVISION.
MAIN-LOGIC.
    MOVE STRING1 TO CONCATENATED-STRING
    MOVE STRING2 TO CONCATENATED-STRING(11:10)
    DISPLAY CONCATENATED-STRING
    STOP RUN.
```

In this example, the first string is moved to the `CONCATENATED-STRING` variable using a simple `MOVE` statement. The second string is then moved to the end of the `CONCATENATED-STRING` variable using a `MOVE` statement with a reference modification that specifies the starting position of the second string.