Neo Pi

Mr. Labouseur

Programming In the Past

ESTIMATED TOTAL TIME: 40-50 HOURS

Q:

How is each language similar or dissimilar to others?

Explain in detail thoughts on each language regarding readability and writability.

What you loved and hated.

Rank the languages

FORTRAN

DATE	HOURS	TASK(TA)/ISSUES(I)/THOUGHTS(TH)
2/10	**5**	Learning Syntax
	INITIAL THOUGHTS:	Create an array [a-z] mapping with index Shift forward by #, Shift backward by #, solve
	WEBSEARCH	'Fortran syntax'
	GPTSEARCH	Asked its opinion on my initial approach
	(TH)	*Using module instead of program programName*
	(TH)	*Module vs Program*
	(TA)	Create an alphabet array OR string?
	WEBSEARCH	'Fortran subroutines'
	GPTSEARCH	'Fortran subroutines'
	(TH)	*I like how you can use all caps and it still looks neat / functions*
	GPTSEARCH	"Give me an example of a subroutine"
	GPTSEARCH	INTENT(IN/OUT)
	(TH)	Using modules will require me to create an additional program so I am switching to one program. Do not need to focus on reusability in this project. Use program and function
	(I)	Choosing programs/functions or modules/subroutines
	(T)	Had to switch from using all caps because it was getting too annoying to use
	(TH)	Using ASCII is definitely the smartest way and I should have thought of that first, thanks GPT.
2/24	**4**	Finishing program
	WEBSEARCH	Syntax
	GPTSEARCH	Syntax

	GPTSEARCH **(TH)** **(T)** **(T)**	Help with for (do) loops I now dislike the caps and switched everything back to lowercase. Turns out that there are too many words that would need to be shifted and it doesn't look neat anymore. Using ASCII codes makes it easier mod() to help wrap if shift # is too large Interesting that you can use mod() to complete a mathematical operation
2/26	**8** (TH) (T) (TH) (T) **GPTSEARCH** **GPTSEARCH** **GPTSEARCH** (TH) (TH) (TH) (T) (TH)	**COBOL** Immediately hate the vertical lines and took too much time to set up Learning syntax I can see why this language would be used for building business reports, it is basically broken English, the only really important/different syntax is the heading where you define your variables Parts of the code must fit into specific columns, probably makes readability easier, modern COBOL is different Simple example function in COBOL For Loop Example "Important syntax to know for COBOL in order to program a caesar cipher?" Manipulating variables with "move" These VSCode extensions are messing with me. I need a new IDE. badly. Way too literal. I don't like it. I hate how you have to start and end everything as well as so many if statements Backtracking is an issue, when the shift reaches past 'a' might come back to this but I spent too much time on it already Most annoying part was fixing my compiler to make sure it matches the cobol in my vscode.
2/27	6 **(TH)** **(T)** **(T)** **(TH's)**	**BASIC** **MAJOR HEADSLAPS ON SETUP** 50 million head slaps just for configuring and setting up the compiler with my vscode. I hate CLI and I hate this project. sorry not sorry. I hope the syntax is truly basic. I finally got to start actually writing the program. Yay. I already know how I logically want to approach coding this problem so I am assuming it will take less time and I just have to get used to the syntax. As always, set the variables and similar to FORTRAN, you just use input and read. Also kind of similar to COBOL but a little bit different. Comments start with one ' which is nice, but at the same time that must cause a lot of issues if you wanted to use

		single quotes in your program. Also similar in the way you denote the end of something with 'end'(very creative)
	(T)	Implementing if statements / loops
	(TH)	The loops are nicer than I imagined, and definitely easier to understand/use than COBOL or Fortran.
	GPTSEARCH	How do I declare a variable
	GPTSEARCH	Provide an example of a for loop
	GPTSEARCH	Provide an example of an if statement
2/28	**4** **Pascal**	Learning Syntax
	(TH)	Heard pascal thrown around a lot in class, but I had no idea what it was until now so I'm kind of looking forward to it Configuring was so much easier, thank god only took me two seconds to download the compiler and I didn't have to do much else unlike the others, I'm assuming it has to do with the age of the languages as Pascal is so far the newest
	(T)	Just need to implement the caesar cypher again (variables first): using var (wow)
	(TH)	We're getting modern. I can feel it.
	GPTSEARCH	Variables, for loops, if statements
	GPTSEARCH	help.
	(TH)	I found the walrus! No headslaps! Smoothest language so far.
3/1	**6** **Scala (procedural manner)**	Var, loops, functions modify state
		Let's look at some syntax (as always, looking at declaration, operations and control statements first) Similar to TypeScript, and I like that
	GPTSEARCH	For Loop Example
	GPTSEARCH	While Loop Example
	GPTSEARCH	Basic function example
	(TH)	Very easy and familiar I would have to choose this as my favorite out of the four
	(TH)	Nothing to hate, and no trouble with configuring the compiler, finally able to use a vscode extension

Dear Diary,

Interesting that in order to create a program in this language, you can 'program programName.' I like the format of fortran and just the overall appearance, also simple to write, simple to read. I know you hate python but I love the simplicity. I originally liked using caps to make things stand out more, but it was getting more annoying to use caps throughout the whole program so I decided to switch back. Although I also did like the readability, as more

code was on the page I liked the caps less and less. For (d0) loops and if statements are not too complicated or different, just syntax is weird. What you think would be the end of a line is the middle, like if (ascii < ichar('A')) ascii = ascii + 26 out_text(i:i) = char(ascii) It just seems like a runon sentence, there is a lack of punctuation. Readability I would have to give it a 7/10. It is not complicated to understand most of the program, even if you knew nothing about FORTRAN if you had basic knowledge of another programming language. Writability is more on the lower scale, and I would give it a 3/10. Maybe I just don't like new languages, but it feels like ':' and '::' are just overused and appear all over the page. Overall, I started good with this language but started to hate it towards the end.

Starting **COBOL:** so far, I like it better than Fortran. Reminds me more of machine language than fortran does, which is maybe why I like it more. Keeping a strict structure of the code is also a new thing for me, but I like it so that it makes it easier to read, especially if this language is not just for programmers. For every variable, you have to describe the level and use PIC x() to initialize which I thought was weird. Then prompting is also very straight-forward, 9/10 readability if I'm being honest. Even though writability has been good for this language so far, as soon as I got into implementing the functions it got a little annoying. Instead of using what seems like keywords, you have to write full sentences to perform an action. Also what is interesting, is that you can manipulate variables using "move". If you want data copied from one variable to another, all you have to do is say "move _____ to _____". Also, every if statement has a corresponding "end-if" statement that makes it kind of easy to see where some begin and end, although it is not normally challenging to do that. After full implementation and careful consideration, I hate this feature now and it makes it worse. I don't like how you have to end everything with end-if or end-perform. Even though you are programming in basically English, I have to give this writability a 4/10 and that's being generous. Overall, I like FORTRAN better as this gave me way too many complications. Might come back to fix backtracking, but can't be bothered right now I spent as much time as it is.

Just started using **BASIC,** and I can say with confidence so far that it is indeed "basic". I definitely prefer the syntax over Fortran and Cobol, it feels more similar to me I think because it is more similar in syntax compared to modern languages we would use today. In terms of readability and writability, I would have to give this a 8/10 overall for both at least. Maybe even a 9/10. Once again, I struggled with configuring my compiler so having to not worry too much about the syntax was a blessing. Like before, we start and end functions/subroutines/programs

with “end _____” which I am still not a huge fan of, but there are worse things that are happening. I think the control statements were easier to implement this time, and more simpler compared to COBOL. Operators like >= and += exist, which I had no idea how much I missed until now. This is by far the best language so far, and I will try to forget I ever did Fortran or especially COBOL. COBOL -> Fortran -> BASIC so far in terms of overall readability and writability. I hope the next language is just as good as BASIC because I know I am going to have a ton more head slaps just to configure it.

Starting **Pascal:** I can feel the language getting more and more modern, thank god. I like this so far, maybe not better than BASIC, but definitely above COBOL or Fortran. Note: found the walrus. What I found weird is that if you wanted to loop through the function, you have to do it in the function body and not the function call. I like BASIC because it allows that. However, the logic is pretty simple, and languages still have begin/end lines which I find really interesting. I wonder at what point they stopped using that. I would also say that the readability is so far probably the easiest it has been, so I have to give it a 9/10 if BASIC was a 8/10. Writability I will stick with an 8/10 but it could easily be a 9 as well. Declaring global vs. local variables is also something I kind of missed, and if you know a programming language it is easy to determine those. Also, I love the comments finally. Using a single ‘ was really weird.

Starting **Scala.** First impression, I love the syntax as it reminds me of TypeScript. The syntax coloring is also more familiar so I just felt more comfortable with it off the rip. Control statements are straight-forward, and it is definitely the least complicated language so far in terms of how much I had to study/research to learn a new syntax. Good old comments are in ‘//’ which I missed, because the other languages had weird ways of implementing comments, even PASCAL which was {}. In terms of readability and writability, I will give this one a 10/10 for both compared to the other languages. What I started to notice was that as I went down the list both readability and writability improved overall, if not in all individual categories. I also was curious to see when the languages were invented, and I guess it makes sense that as the languages got newer, readability and writability improved. I actually love this language and have nothing negative to say about it.

Overall, I hated COBOL the most and it gave me the most problems. One day, my compiler is working and the next I can’t compile the file and it’s just too frustrating to make changes to it.

On top of that, I have no prior experience with this language so fixing a bug is 10 times more annoying.

Time Estimate: 40-50 hours.
Actual Time: 33 hours.
I spent some additional time after finishing the Pascal program to fix the other programs and edit the output, but some of them are so annoying to deal with I just skipped the issue. For example, my COBOL program does not 100% output correctly, but the syntax is so annoying and the compiler keeps giving me issues I can't even try to fix. I think the shorter time discrepancy also has to do with using AI, because it provided me with example syntax and exact syntax when I needed help. However, maybe because of the age of the language when I needed to fix a bug it was not helpful at all. If my Basic and COBOL compilers were behaving properly, I would have spent a couple more hours trying to fix the errors that appear. However, since the programs were working when I pushed them to my repo, I do not wish to struggle over fixing the 3% of output error that is occurring. All programs should function properly, but might have different structures of output.

Rankings:
COBOL -> Fortran -> BASIC-> Pascal -> Scala

Source Code
Fortran
```
program caesar_cipher
  implicit none
  character(len=100) :: input, encrypted, decrypted
  integer :: shift_amount

  print *, 'Enter text (uppercase or lowercase, no spaces):'
  read *, input
  print *, 'Enter shift amount:'
  read *, shift_amount

  encrypted = encrypt(trim(input), shift_amount)
  print *, 'Encrypted text:', trim(encrypted)
```

```
  decrypted = decrypt(encrypted, shift_amount)
  print *, 'Decrypted text:', trim(decrypted)

  call solve(trim(input), 26)

contains

  function encrypt(text, shift_amount) result(out_text)
    character(len=*), intent(in) :: text
    integer, intent(in) :: shift_amount
    character(len=len(text)) :: out_text
    integer :: i, ascii, new_ascii

    out_text = text
    do i = 1, len_trim(text)
      ascii = ichar(text(i:i))
      if (ascii >= ichar('A') .and. ascii <= ichar('Z')) then
        new_ascii = mod(ascii - ichar('A') + shift_amount, 26) + ichar('A')
      else if (ascii >= ichar('a') .and. ascii <= ichar('z')) then
        new_ascii = mod(ascii - ichar('a') + shift_amount, 26) + ichar('a')
      else
        new_ascii = ascii
      end if
      out_text(i:i) = char(new_ascii)
    end do
  end function encrypt

  function decrypt(text, shift_amount) result(out_text)
    character(len=*), intent(in) :: text
    integer, intent(in) :: shift_amount
    character(len=len(text)) :: out_text
    integer :: i, ascii, new_ascii
```

```
  out_text = text
  do i = 1, len_trim(text)
   ascii = ichar(text(i:i))
   if (ascii >= ichar('A') .and. ascii <= ichar('Z')) then
    new_ascii = mod(ascii - ichar('A') - shift_amount + 26, 26) + ichar('A')
   else if (ascii >= ichar('a') .and. ascii <= ichar('z')) then
    new_ascii = mod(ascii - ichar('a') - shift_amount + 26, 26) + ichar('a')
   else
    new_ascii = ascii
   end if
   out_text(i:i) = char(new_ascii)
  end do
 end function decrypt

 subroutine solve(original_text, maxShift)
  character(len=*), intent(in) :: original_text
  integer, intent(in) :: maxShift
  integer :: s
  character(len=len(original_text)) :: temp

  do s = maxShift, 0, -1
   temp = encrypt(original_text, s)
   print '(A, I2, A, A)', 'Caesar ', s, ': ', trim(temp)
  end do
 end subroutine solve

end program caesar_cipher
```

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CAESAR-CIPHER.

DATA DIVISION.
WORKING-STORAGE SECTION.

01 INPUT-MSG       PIC X(100).
01 ENCRYPTED-MSG    PIC X(100).
01 DECRYPTED-MSG    PIC X(100).
01 RESULT-MSG      PIC X(100).
01 SHIFT           PIC 99.
01 I               PIC 99 VALUE 1.
01 CHAR-CODE       PIC 999.
01 NEW-CODE        PIC 999.
01 CHAR            PIC X.
01 MESSAGE-LENGTH    PIC 99.

PROCEDURE DIVISION.
MAIN.
  PERFORM MAIN-PROCEDURE.
  STOP RUN.

MAIN-PROCEDURE.
  DISPLAY "Enter message to encrypt (Any Case Allowed): ".
  ACCEPT INPUT-MSG.
  DISPLAY "Enter shift value (1-25): ".
  ACCEPT SHIFT.

  PERFORM CALCULATE-MESSAGE-LENGTH.
  PERFORM ENCRYPT-PROCESS.
  DISPLAY "Encrypted Message: " ENCRYPTED-MSG.

  PERFORM DECRYPT-PROCESS.
```

```
    DISPLAY "Decrypted Message: " DECRYPTED-MSG.

    DISPLAY "Attempting to solve the cipher (Brute Force)...".
    PERFORM SOLVE.

    EXIT.

CALCULATE-MESSAGE-LENGTH.
    MOVE 1 TO MESSAGE-LENGTH.
    PERFORM UNTIL INPUT-MSG(MESSAGE-LENGTH:1) = SPACE
        ADD 1 TO MESSAGE-LENGTH
    END-PERFORM.
    SUBTRACT 1 FROM MESSAGE-LENGTH.

ENCRYPT-PROCESS.
    MOVE SPACES TO ENCRYPTED-MSG.
    MOVE 1 TO I.
    PERFORM UNTIL I > MESSAGE-LENGTH
        MOVE INPUT-MSG(I:1) TO CHAR
        IF CHAR >= "A" AND CHAR <= "Z"
            MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
            COMPUTE NEW-CODE = CHAR-CODE + SHIFT
            IF NEW-CODE > 90
                SUBTRACT 26 FROM NEW-CODE
            END-IF
            MOVE FUNCTION CHAR(NEW-CODE) TO ENCRYPTED-MSG(I:1)
        ELSE IF CHAR >= "a" AND CHAR <= "z"
            MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
            COMPUTE NEW-CODE = CHAR-CODE + SHIFT
            IF NEW-CODE > 122
                SUBTRACT 26 FROM NEW-CODE
            END-IF
            MOVE FUNCTION CHAR(NEW-CODE) TO ENCRYPTED-MSG(I:1)
        ELSE
```

```
        MOVE CHAR TO ENCRYPTED-MSG(I:1)
      END-IF
      ADD 1 TO I
    END-PERFORM.

DECRYPT-PROCESS.
    MOVE SPACES TO DECRYPTED-MSG.
    MOVE 1 TO I.
    PERFORM UNTIL I > MESSAGE-LENGTH
      MOVE ENCRYPTED-MSG(I:1) TO CHAR
      IF CHAR >= "A" AND CHAR <= "Z"
        MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
        COMPUTE NEW-CODE = CHAR-CODE - SHIFT
        IF NEW-CODE < 65
          ADD 26 TO NEW-CODE
        END-IF
        MOVE FUNCTION CHAR(NEW-CODE) TO DECRYPTED-MSG(I:1)
      ELSE IF CHAR >= "a" AND CHAR <= "z"
        MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
        COMPUTE NEW-CODE = CHAR-CODE - SHIFT
        IF NEW-CODE < 97
          ADD 26 TO NEW-CODE
        END-IF
        MOVE FUNCTION CHAR(NEW-CODE) TO DECRYPTED-MSG(I:1)
      ELSE
        MOVE CHAR TO DECRYPTED-MSG(I:1)
      END-IF
      ADD 1 TO I
    END-PERFORM.
SOLVE.
    PERFORM VARYING SHIFT FROM 1 BY 1 UNTIL SHIFT > 25
      MOVE SPACES TO RESULT-MSG
      MOVE 1 TO I
      PERFORM UNTIL I > MESSAGE-LENGTH
```

```
      MOVE ENCRYPTED-MSG(I:1) TO CHAR
      IF CHAR >= "A" AND CHAR <= "Z"
        MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
        COMPUTE NEW-CODE = CHAR-CODE - SHIFT
        IF NEW-CODE < 65
          ADD 26 TO NEW-CODE
        END-IF
        MOVE FUNCTION CHAR(NEW-CODE) TO RESULT-MSG(I:1)
      ELSE IF CHAR >= "a" AND CHAR <= "z"
        MOVE FUNCTION ORD(CHAR) TO CHAR-CODE
        COMPUTE NEW-CODE = CHAR-CODE - SHIFT
        IF NEW-CODE < 97
          ADD 26 TO NEW-CODE
        END-IF
        MOVE FUNCTION CHAR(NEW-CODE) TO RESULT-MSG(I:1)
      ELSE
        MOVE CHAR TO RESULT-MSG(I:1)
      END-IF
      ADD 1 TO I
    END-PERFORM
    DISPLAY "Shift " SHIFT ": " RESULT-MSG
END-PERFORM.
```

BASIC

```
' FreeBASIC Caesar Cipher - Project One
DIM message AS STRING
DIM shift AS INTEGER

FUNCTION toUpperCase (txt AS STRING) AS STRING
  DIM i AS INTEGER
  DIM newTxt AS STRING
  FOR i = 1 TO LEN(txt)
    newTxt += UCASE(MID(txt, i, 1))
  NEXT i
  RETURN newTxt
END FUNCTION

FUNCTION encrypt (txt AS STRING, shift AS INTEGER) AS STRING
  DIM i AS INTEGER, charCode AS INTEGER
  DIM encrypted AS STRING
  txt = toUpperCase(txt)

  FOR i = 1 TO LEN(txt)
    charCode = ASC(MID(txt, i, 1))

    IF charCode >= 65 AND charCode <= 90 THEN
      charCode = ((charCode - 65 + shift) MOD 26) + 65
    END IF

    encrypted += CHR(charCode)
  NEXT i
  RETURN encrypted
END FUNCTION

FUNCTION decrypt (txt AS STRING, shift AS INTEGER) AS STRING
  DIM i AS INTEGER, charCode AS INTEGER
  DIM decrypted AS STRING
```

```
  txt = toUpperCase(txt)

  FOR i = 1 TO LEN(txt)
    charCode = ASC(MID(txt, i, 1))

    IF charCode >= 65 AND charCode <= 90 THEN
      charCode = ((charCode - 65 - shift + 26) MOD 26) + 65
    END IF

    decrypted += CHR(charCode)
  NEXT i
  RETURN decrypted
END FUNCTION

SUB solve (txt AS STRING, shiftValue AS INTEGER)
  DIM i AS INTEGER
  PRINT "Solving (Showing all shifts from "; shiftValue; " to 0):"

  FOR i = shiftValue TO 0 STEP -1
    PRINT "Caesar "; i; ": "; decrypt(txt, i)  '
  NEXT i
END SUB

DIM encryptedText AS STRING
DIM decryptedText AS STRING

PRINT "Enter a message: ";
INPUT message
PRINT "Enter shift amount: ";
INPUT shift

encryptedText = encrypt(message, shift)
PRINT "Encrypted: "; encryptedText
```

```
decryptedText = decrypt(encryptedText, shift)
PRINT "Decrypted: "; decryptedText

solve(encryptedText, 25)

SLEEP
```

PASCAL

```
program CaesarCipher;

var
  message, encryptedMessage: string;
  shift: integer;

function EncryptChar(ch: char; shift: integer): char;
var
  charCode: integer;
begin
  if (ch >= 'A') and (ch <= 'Z') then
  begin
    charCode := Ord(ch);
    charCode := ((charCode - Ord('A') + shift + 26) mod 26) + Ord('A');
    EncryptChar := Chr(charCode);
  end
  else
    EncryptChar := ch;
end;

function DecryptChar(ch: char; shift: integer): char;
var
  charCode: integer;
```

```
begin
 if (ch >= 'A') and (ch <= 'Z') then
 begin
  charCode := Ord(ch);
  charCode := ((charCode - Ord('A') - shift + 26) mod 26) + Ord('A');
  DecryptChar := Chr(charCode);
 end
 else
  DecryptChar := ch;
end;

function Encrypt(text: string; shift: integer): string;
var
 i: integer;
begin
 text := UpCase(text);
 for i := 1 to Length(text) do
  text[i] := EncryptChar(text[i], shift);
 Encrypt := text;
end;

function Decrypt(text: string; shift: integer): string;
var
 i: integer;
begin
 text := UpCase(text);
 for i := 1 to Length(text) do
  text[i] := DecryptChar(text[i], shift);
 Decrypt := text;
end;

procedure Solve(text: string; maxShiftValue: integer);
var
 i: integer;
```

```
begin
  writeln;
  writeln('Solving for all shifts:');
  for i := maxShiftValue downto 0 do
    writeln('Caesar ', i:2, ': ', Encrypt(text, i));
end;

begin
  write('Enter a message: ');
  readln(message);

  write('Enter shift amount: ');
  readln(shift);

  message := UpCase(message);
  encryptedMessage := Encrypt(message, shift);

  writeln;
  writeln('Results:');
  writeln('Encrypted: ', encryptedMessage);

  Solve(message, 26);
end.
```

SCALA

```
import scala.io.StdIn

object CaesarCipher {
 def main(args: Array[String]): Unit = {
  println("Enter a string:")
  val text = StdIn.readLine().toUpperCase()
  println("Enter shift value:")
  val shift = StdIn.readInt()

  println("\nResults:")
  println(s"Encrypted: ${encrypt(text, shift)}")
  println("\nSolving for all shifts:")
  solve(text, 26)
 }

 // Encrypt function
 def encrypt(text: String, shift: Int): String = {
  shiftText(text, shift)
 }

 // Decrypt function
 def decrypt(text: String, shift: Int): String = {
  shiftText(text, -shift)
 }

 // Solving function
 def solve(text: String, maxShift: Int): Unit = {
  for (shift <- maxShift to 0 by -1) {
   println(f"Caesar $shift%2d: ${encrypt(text, shift)}")
  }
 }
```

```
  def shiftText(text: String, shift: Int): String = {
    val result = new StringBuilder()
    for (char <- text) {
      if (char.isLetter) {
        val base = if (char.isUpper) 'A' else 'a'
        result.append(((char - base + shift + 26) % 26 + base).toChar)
      } else {
        result.append(char)
      }
    }
    result.toString()
  }
}
```