CMPT 331 - Spring 2022 - Dr. Labouseur

# Functional Programming

Jake Vissicchio
Jake.Vissicchio1@Marist.edu

September 27, 2022

## 1 Log

Time Spent Prediction:
I'm assuming it will take me about **20 hours** since it took about that long on the Programming In The Past. Of course these will technically be harder to grasp in comparison however I will be able to save a lot of time from the fact that I have worked with Caesar Ciphers a lot now and I now have the math for it figured out so as long as I figure out the syntax of the language the shift should work as intended.

| Log | | |
|---|---|---|
| Date: | Hours Spent: | Tasks/Accomplishments/Issues/Thoughts: |
| 4/12 | 1 | I took some time downloading all that I need in order to do the project as well as taking the time to familiarize myself with the IDEs. I also chose to start with ML since we most recently went over it in class so it's kind of fresh in my mind. |
| 4/13 | 2 | I created a shiftHandler function to do all of the math for each character shifting which will be pivotal since I am using the implode, map, shift, explode idea that was talked about in class. |
| 4/14 | 4 | Created the encrypt function and works as intended with the test cases I used. Of course because encrypt works that means decrypt should work as well. |
| 4/14 | 4ish | Actually decrypt was not working because I forgot ML uses tilde over '-' when specifying a negative number. |
| 4/15 | 6 | I cannot really figure out how to do solve so I am going to come back to it later after looking at the other languages. |
| 4/16 | 7 | Started LISP and created a shiftHandler function. |
| 4/16 | 9 | Both encrypt and decrypt is complete for LISP. |
| 4/18 | 11 | Got solve to work using recursion. Moving onto Javascript |
| 4/18 | 12 | Created the shiftHandler function for Javascript. |
| 4/18 | 13 | While working on encrypt for Javascript I had a lot of issues with nothing printing but I noticed it was still running. Turns out I needed to put a "return" inside of my anonymous function calling shiftHandler for it to work correctly. |
| 4/18 | 14 | Javascript Caesar Cipher is complete. Was able to reuse my recursion strategy for Solve and everything works as intended. Onto Scala. |
| 4/19 | 16 | Scala was honestly pretty painless since I did Javascript before it. They shared a ton of similarities with only very few key differences that held me back but the Caesar Cipher is complete. If I finish Erlang and ML in 4 hours I'll actually be able to complete the assignment on schedule. |

| Log | | |
|---|---|---|
| Date: | Hours Spent: | Tasks/Accomplishments/Issues/Thoughts: |
| 4/19 | 20 | Took me forever to figure out just to create the shiftHandler for Erlang so unfortunately I went over my time. |
| 4/20 | 22 | I think I got encrypt to work to the best of my ability. It does not return a value but instead it prints out the encrypted string within the function |
| 4/20 | 24 | I did a lot of research on recursion in Erlang and I was able to handle solve with what I learned but I do not understand how to get the pretty output that shows each shift. |
| 4/20 | 26 | Finally got solve to work in ML using a similar strategy I used in Erlang so good thing I skipped it and came back. |

### 1.1 Discrepancy of Time

I was actually pretty close to my prediction until I reached Erlang, which ended up pushing me back a couple hours. I believe I underestimated how much more difficult coding becomes when you limit yourself from using loops since it is something you become accustomed to using from the start of your coding career. It really bends your brain but it is interesting to see how many lines of code you save due to it.

## 2 Commentary

This section includes random thoughts that came to my head while I was doing the assignment (sort of like a diary).

### 2.1 ML

- Got a good head start on learning the language in class which is good.

- It feels very cool to use because I've never really tried strictly functional programming before.

- The first function I wanted to make before dealing with encrypt, decrypt and solve was a shiftHandler function which basically will take a character and a shift amount and then do the shift accounting for going past 'Z'.

- So I was stuck on making the shiftHandler for a while until I looked up the error and realized I have been mismatching parentheses the whole time.

- Now that I have my shiftHandler I can make the Encrypt function also making use of implode, map and explode. shiftHandler will need to be an anonymous function so I can use it here. Matching up the parentheses is going to be very tedious though.

- My simple test case that I usually do is "za bc" with a shift of 1 since it handles lower to upper, spaces, and going past 'z'.

- So a big issue I had with decrypt was forgetting tilde specifies negative. I think this could be beneficial because of the fact that it allows you not to confuse negative with subtraction but due to me being used to every other language that usually uses '-' it makes it hard to remember.

- Solve is pretty difficult for me to figure out since this uses a simple for loop if I wasn't doing functional program. I thought about recursion but I do not know how I can keep track of the maximum shift while recursively calling the function.

- I am going to skip solve for now and see if I can find out something using other languages

- Came back after finishing everything. The issue I am currently having is the fact that I cannot have multiple lines of code within an if statement or I get an error.

- Turns out I actually CAN have multiple lines of code within an if statement, I was just not getting the right results on google before. Now solve works as intended.

- I forgot to mention this but as I was testing output on it I realized how much I love the output in ML. Its cool that it tells you almost everything about a variable when you put first declare it.

## 2.2 LISP

- One thing I can say from my first impressions is that the readability is absolutely awful. The massive amount of parentheses and operators being before the numbers rather than in between makes it really difficult to understand and there's usually a trade-off for writability but in this case its annoying to write as well as read.

- Gonna do a similar strategy I used for ML for LISP by starting with a shiftHandler function.

- Good thing there is a map function for LISP since I am able to reuse what I did for ML but just translate it to LISP. Also anonymous functions have the keyword lambda which is giving me scary flashbacks to the previous assignment (at least that assignment helped me understand more for this assignment).

- I was confused why my decrypt wasn't working as intended for a while until I realized I forgot to set my string to the new encrypted string. Silly mistake.

- I think the only thing that will work for solve is probably recursion since I unfortunately cannot loop.

- After a ton of trial and error I was able to make solve work. Now I am having some trouble getting it to print nicely because concatenation is weird.

## 2.3 Javascript

- Ahhh finally some familiarity. However I cannot use loops so what I am first going to do is make my shiftHandler for Javascript.

- After only using Typescript before, this feels like I am just using an extremely lenient Typescript so that saves some stress.

- The writability is really good but readability would probably take a hit because of it since you can write variables without having to specify its type.

- Thankfully there exists a map function and anonymous functions in Javascript which allows me to take a similar approach compared to the other languages for encrypt.

- For Javascript I can make use of split on "" to first split my string into a list of characters (technically substrings) and then after doing the shift on each element I can make use of join on "" to bring them all together again giving the encrypted string. Kind of similar to the implode then later explode that was done in ML.

- Decrypt was being annoying then I remembered I actually had a similar problem during the Programming In The Past assignment. I needed to add 26 to the shiftAmount when the shiftAmount is negative.

- Since I know how to use recursion to handle solve this part was not bad and on the plus side the concatenation was way easier.

## 2.4 Scala

- This should honestly not be too bad considering I am at least familiar with the syntax of Scala from Programming in the Past also Javascript is fairly similar to Scala so I can reuse a lot of my Javascript implementation.

- Its writability is a little worse than Javascript due to having to put var in front of variables and having to specify its type but it has better readability due to these factors.

- In Scala you cannot uppercase a char, only a string so instead of using a toUpperCase in the shiftHandler function I used it on the string before calling the encrypt function so that it is already all caps before any shifts are done on it.

- There isn't much to say about Scala that isn't similar to what I said in my Javascript diary but one weird thing I found was instead of join you use mk-String which does the same thing but its a weird keyword for sure.

## 2.5 Erlang

- This is going to take a while. Apparently there is no easy built in way to turn a single char to an ASCII code and vise versa meaning that I will need to try a different implementation.

- The error messages are super non descriptive so far. Very frustrating to deal with. It seems to be very finicky too.

- I hate that variables have to be uppercase it goes against every other language I've dealt with plus it's unsatisfying to look at.

- I think I might have as much difficulty on this as I had in COBOL, I can't wrap my head around it.

- I found a way to make a char to ascii code which is getting the head of the char, I'm upset with myself it took so long since it was right in front of my face haha. Also it's weird that Erlang uses rem rather than mod.

- Printing is really complicated compared to other languages. They have these different formats that you need to specify with a tilde. Also I noticed that I need to put brackets around my String when printing them or else I get an error. I guess because it treats it as a list.

- There is also no return operator which kind of complicates things. There is a way to get around this by using pattern matching from what I researched but honestly it looks extremely complicated and unfamiliar so I can just print out the string within encrypt. But the problem with that is that I cannot get the encrypted string for decrypt.

- I know Language Study goes over Erlang and honestly I think I would have really struggled. Of course the big difference is that for this assignment you kind of have to learn the basic syntax in a much faster rate while working on 4 other languages meanwhile that class focuses on one language but still. Erlang is not like any other language I have used before and almost feels as alien as when I was using COBOL (However I would use this over COBOL any day).

- I figured out how to handle solve after doing a ton of research on recursive functions in Erlang.

- Definitely my weakest code out of the 5 but at least it runs and handles the function calls correctly.

## 3 Code Listings

### 3.1 ML

```
fun shiftHandler(myChar, shiftAmount) =
    if myChar = #" " then
        #" "
    else
        chr(((ord (Char.toUpper myChar) - 65 + shiftAmount) mod 26) + 65);

fun encrypt (myString, shiftAmount) =
        String.implode(map (fn currChar =>
            shiftHandler (currChar, shiftAmount)) (String.explode(myString)));

fun decrypt(myString, shiftAmount) =
    encrypt(myString, shiftAmount * ~1);

fun solve(myString, currShift) =
        if currShift < 26 then (
                print ("Caesar (carrot) (Int.toString currShift)
                    ": "(carrot)encrypt(myString, currShift)(carrot)"\n");
        solve(myString, currShift + 1)
    )
    else
    print ("Caesar (carrot) (Int.toString currShift)
                    ": "(carrot)encrypt(myString, currShift)(carrot)"\n");
```

```
val str = "za bc";

val shift = 1;

val encryptedStr = encrypt(str, shift);

val decryptedStr = decrypt(encryptedStr, shift);

solve("HAL", 0);

(I apologize for the (carrot) it would not print the actual carrot character
in code listings)
```

### 3.2 LISP

```
(setq currentShift 0)

(defun shiftHandler(myChar shiftAmount)
        (if (= (char-code myChar) 32) (code-char 32)
                (code-char (+ (mod(+ (- (char-code myChar) 65)
                    shiftAmount) 26) 65))
        )
)

(defun encrypt(myString shiftAmount)
        (map 'string #'(lambda (char)(shiftHandler char shiftAmount))
            (string str))
)

(defun decrypt(myString shiftAmount)
        (encrypt myString (* shiftAmount -1))
)

(defun solve(myString maxShiftValue)
        (print (encrypt myString currentShift))
        (when (< currentShift maxShiftValue)
                (setq currentShift (+ currentShift 1))
                (solve myString maxShiftValue)
        )

)

(defun cipher()
        (setq str "za bc")
        (setq shift 1)
        (setq str (string-upcase str))
        (print str)
        (setq str (encrypt str shift))
        (print str)
        (setq decryptedStr (decrypt str shift))
        (print decryptedStr)
        (setq str "HAL")
        (setq maxShift 26)
        (solve str maxShift)
)
(cipher)
```

### 3.3 Javascript

```
function shiftHandler(myChar, shiftAmount)
{
    newChar = "";
    newShift = shiftAmount;
    if (myChar == " ")
    {
        newChar = " ";
    }
    else
    {
        if (shiftAmount < 0)
        {
                newShift = shiftAmount + 26;
        }
        newChar = String.fromCharCode((((myChar.toUpperCase().charCodeAt(0))
            - 65 + newShift) % 26) + 65);
    }
    return newChar;
}

function encrypt(myString, shiftAmount)
{
        newString = myString.split("");
        result = newString.map( function(myChar) {return shiftHandler(myChar, sh
        return result;
}

function decrypt(myString, shiftAmount)
{
        newString = "";
        newString = encrypt(myString, shiftAmount * -1);
        return newString;
}

currentShift = 0;
function solve(myString, maxShiftValue)
{
        console.log("Caesar " + currentShift + ": " +
            encrypt(myString, currentShift));
        if (currentShift < maxShiftValue)
        {
                currentShift = currentShift + 1;
                solve(myString, maxShiftValue);
        }
}

str = "za bc";
shift = 1;
console.log(str);
str = encrypt(str, shift);
console.log(str);
console.log(decrypt(str, shift));
solve("HAL", 26);
```

### 3.4 Scala

```
object Main
{
        var currentShift: Int = 0;
        def main(args: Array[String])
        {
                var str: String = "za bc";
                var upperStr: String = "";
                upperStr = str.toUpperCase();
                var shift: Int = 1;
                println(upperStr);
                println(this.encrypt(upperStr, shift));
                var encryptedStr: String = this.encrypt(upperStr, shift);
                println(this.decrypt(encryptedStr, shift));
                this.solve("HAL", 26);
        }

        def shiftHandler(myChar: String, shiftAmount: Int): String =
        {
                var newChar: String = "";
                var newShift: Int = shiftAmount;
                if (myChar == " ")
                {
                        newChar = " ";
                }
                else
                {
                        if (shiftAmount < 0)
                        {
                                newShift = shiftAmount + 26;
                        }
                        newChar = (((myChar.charAt(0).toInt - 65 + newShift)
                            % 26) + 65).toChar.toString;
                }
                return newChar;
        }

        def encrypt(myString: String, shiftAmount: Int): String =
        {
                var newShift: Int = shiftAmount;
                var newString = myString.split("");
                var result = (newString.map((newChar: String) =>
                    shiftHandler(newChar, newShift)).mkString(""));
                return result;
        }

        def decrypt(myString: String, shiftAmount: Int): String =
        {
                var inverseShift: Int = shiftAmount * -1;
                var result: String = "";
                result = this.encrypt(myString, inverseShift);
                return result;
        }

        def solve(myString: String, maxShiftValue: Int)
        {
                println("Caesar " + currentShift + ": " +
                    encrypt(myString, currentShift));
                if (currentShift < maxShiftValue)
                {
                        currentShift = currentShift + 1;
                        solve(myString, maxShiftValue);
                }
        }
}
```

### 3.5 Erlang

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

shiftChar(MyChar, ShiftAmount) ->
        if
                MyChar == " " ->
                        MyChar;
                true ->
                        ((hd(MyChar) - 65 + ShiftAmount) rem 26) + 65
        end.

encrypt(MyString, ShiftAmount) ->
        NewString = lists:map(fun(MyChar) ->
            shiftChar([(MyChar)], ShiftAmount) end, MyString),
        io:format("~s\n", [NewString]).

decrypt(MyString, ShiftAmount) ->
        encrypt(MyString, (ShiftAmount * -1)+ 26).

solve(MyString, CurrentShift) when CurrentShift == 26 ->
        encrypt(MyString, CurrentShift);
solve(MyString, CurrentShift) when CurrentShift < 26 ->
        encrypt(MyString, CurrentShift),
        PlusShift = CurrentShift + 1,
        solve(MyString, PlusShift).

main() ->
        Str = "za bc",
        ShiftAmount = 1,
        UpperStr = string:uppercase(Str),
        io:format("~s\n", [UpperStr]),
        encrypt(UpperStr, ShiftAmount),
        decrypt(UpperStr, ShiftAmount),
        solve(UpperStr, 0).
```

## 4 Output

### 4.1 ML

```
(*shift of 1*)
val str = "za bc" : string
val encryptedStr = "AB CD" : string
val decryptedStr = "ZA BC" : string

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

(*shift of 007*)
val str = "No google I am not searching for machine language" : string
val encryptedStr = "UV NVVNSL P HT UVA ZLHYJOPUN MVY THJOPUL SHUNBHNL" : string
val decryptedStr = "NO GOOGLE I AM NOT SEARCHING FOR MACHINE LANGUAGE" : string

Caesar 0: FBSUI
Caesar 1: GCTVJ
Caesar 2: HDUWK
Caesar 3: IEVXL
Caesar 4: JFWYM
Caesar 5: KGXZN
Caesar 6: LHYAO
Caesar 7: MIZBP
Caesar 8: NJACQ
Caesar 9: OKBDR
Caesar 10: PLCES
Caesar 11: QMDFT
Caesar 12: RNEGU
Caesar 13: SOFHV
Caesar 14: TPGIW
Caesar 15: UQHJX
Caesar 16: VRIKY
Caesar 17: WSJLZ
Caesar 18: XTKMA
Caesar 19: YULNB
Caesar 20: ZVMOC
Caesar 21: AWNPD
Caesar 22: BXOQE
Caesar 23: CYPRF
Caesar 24: DZQSG
Caesar 25: EARTH
Caesar 26: FBSUI
val it = () : unit
```

### 4.2 LISP

```
;shift of 1
"ZA BC"
"AB CD"
"ZA BC"

"HAL"
"IBM"
"JCN"
"KDO"
"LEP"
"MFQ"
"NGR"
"OHS"
"PIT"
"QJU"
"RKV"
"SLW"
"TMX"
"UNY"
"VOZ"
"WPA"
"XQB"
"YRC"
"ZSD"
```

```
"ATE"
"BUF"
"CVG"
"DWH"
"EXI"
"FYJ"
"GZK"
"HAL"

;shift of 5
"I CANNOT THINK OF ANYTHING FUNNY FOR THIS STRING"
"N HFSSTY YMNSP TK FSDYMNSL KZSSD KTW YMNX XYWNSL"
"I CANNOT THINK OF ANYTHING FUNNY FOR THIS STRING"

"GZZGIQ"
"HAAHJR"
"IBBIKS"
"JCCJLT"
"KDDKMU"
"LEELNV"
"MFFMOW"
"NGGNPX"
"OHHOQY"
"PIIPRZ"
"QJJQSA"
"RKKRTB"
"SLLSUC"
"TMMTVD"
"UNNUWE"
"VOOVXF"
"WPPWYG"
"XQQXZH"
"YRRYAI"
"ZSSZBJ"
"ATTACK"
"BUUBDL"
"CVVCEM"
"DWWDFN"
"EXXEGO"
"FYYFHP"
"GZZGIQ"
```

### 4.3 Javascript

```
//shift of 1
```

```
za bc
AB CD
ZA BC

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

//shift of 8
I dislike Cobol
Q LQATQSM KWJWT
I DISLIKE COBOL

Caesar 0: IFMMP
Caesar 1: JGNNQ
Caesar 2: KHOOR
Caesar 3: LIPPS
Caesar 4: MJQQT
Caesar 5: NKRRU
Caesar 6: OLSSV
Caesar 7: PMTTW
Caesar 8: QNUUX
Caesar 9: ROVVY
Caesar 10: SPWWZ
Caesar 11: TQXXA
Caesar 12: URYYB
Caesar 13: VSZZC
Caesar 14: WTAAD
Caesar 15: XUBBE
Caesar 16: YVCCF
Caesar 17: ZWDDG
Caesar 18: AXEEH
Caesar 19: BYFFI
Caesar 20: CZGGJ
Caesar 21: DAHHK
Caesar 22: EBIIL
Caesar 23: FCJJM
Caesar 24: GDKKN
Caesar 25: HELLO
Caesar 26: IFMMP
```

### 4.4 Scala

```
//shift of 1
ZA BC
AB CD
ZA BC

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

//shift of 20
AUSTIN POWERS
UOMNCH JIQYLM
AUSTIN POWERS

Caesar 0: KBLF
Caesar 1: LCMG
Caesar 2: MDNH
Caesar 3: NEOI
Caesar 4: OFPJ
Caesar 5: PGQK
Caesar 6: QHRL
Caesar 7: RISM
Caesar 8: SJTN
Caesar 9: TKUO
Caesar 10: ULVP
Caesar 11: VMWQ
Caesar 12: WNXR
Caesar 13: XOYS
Caesar 14: YPZT
Caesar 15: ZQAU
Caesar 16: ARBV
Caesar 17: BSCW
Caesar 18: CTDX
Caesar 19: DUEY
Caesar 20: EVFZ
Caesar 21: FWGA
Caesar 22: GXHB
Caesar 23: HYIC
Caesar 24: IZJD
Caesar 25: JAKE
Caesar 26: KBLF
```

## 4.5 Erlang

```
%shift of 1
```

```
ZA BC
AB CD
ZA BC

HAL
IBM
JCN
KDO
LEP
MFQ
NGR
OHS
PIT
QJU
RKV
SLW
TMX
UNY
VOZ
WPA
XQB
YRC
ZSD
ATE
BUF
CVG
DWH
EXI
FYJ
GZK
HAL

%shift of 15
COFFEE AND MORE COFFEE
RDUUTT PCS BDGT RDUUTT
COFFEE AND MORE COFFEE

UJSFE
VKTGF
WLUHG
XMVIH
YNWJI
ZOXKJ
APYLK
BQZML
CRANM
```

```
DSBON
ETCPO
FUDQP
GVERQ
HWFSR
IXGTS
JYHUT
KZIVU
LAJWV
MBKXW
NCLYX
ODMZY
PENAZ
QFOBA
RGPCB
SHQDC
TIRED
UJSFE
```

## 5 Official Ranking of Jake Vissicchio

1. JavaScript
- Never really used strictly Javascript before and only Typescript so using it kind of felt like a easier to use Typescript. I felt comfortable writing in it and it has okay readability, the only thing that hurts it is that you do not have to specify the data type of the variable.

2. Scala
- Scala takes the 2 spot again. Honestly I really like it for the same reasons I really like Java. The only reason I put Javascript above it was because of how easy it is to write in Javascript. Its huge negative is its slow runtime.

3. ML
- This language was one of the two that took the longest this project and honestly it was very difficult figuring out however it is interesting. Basically felt like a more readable LISP. A major thing that holds it back is its unfortunate acronym matching Machine Language making it annoying to find resources. My trick was searching SML rather than ML if I wanted to know more about something specific in the language such as "SML if statement".

4. LISP
- The only reason why ML is better than Lisp in my opinion is its horrendous readability and writability but it is kind of cool to figure out. I was counting parentheses on my screen a lot which did hurt its rating a lot.

5. Erlang
- After I finished it I can understand it was not as bad as my complaining made it out to be but it still has a lot of issues I wish it did not have. It is missing so many things that other languages have for convenience. If it had more convenient built in functions and more understandable error feedback it would easily be 3.

## 6 Resources

Here are a few resources that I used for this project (not including the notes).

- https://www.tutorialspoint.com/lisp

- http://clhs.lisp.se

- https://www.w3schools.com/jsref/default.asp - https://en.wikibooks.org/wiki/JavaScript/

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

- https://www.geeksforgeeks.org/anonymous-functions-in-scala/

- https://www.tutorialspoint.com/erlang

- https://www.erlang.org/doc/index.html

- https://stackoverflow.com/questions/12662858/multiple-statements-in-if-else-insml