Functional Programming

Daniel Gisolfi

LISP

Code

```
;; Daniel Nicolas Gisolfi
;; LISP Cipher
;; BEST RESOURCE => http://progopedia.com/implementation/steel-bank-common-lisp/

;; Create a function to do individual character shifting
(defun offset (ch key)
  ;; Here comes the "Lots of Irritating Superfluous Parentheses "
  (let* ((c  (char-code  ch)) (la (char-code #\a)) (ua (char-code #\A))
         (base (cond ((<= la c (char-code #\z)) la)
                     ((<= ua c (char-code #\Z)) ua)
                     (nil))))
    (if base (code-char (+ (mod (+ (- c base) key) 26) base)) ch)))

(defun encrypt (str key)
  ;; map is very useful and is used in a few of these languages actually
  ;; For each char in string call offset to shift the character
  (map 'string #'(lambda (c) (offset c key)) str))

;; lazy decryption...or efficient?
(defun decrypt (str key) (encrypt str (- key)))

;; I will admit this is a very concise version of the solve function
;; thanks to LISP....not worth it though
(defun solve (str num)
  ;; loop through the required number of test cases, encrypt as we go
  (loop for n from 0 to num
    do (format t "Caesar ~D: ~a~%" n (encrypt str n))))


;; Define the String to be encrypte and the base key
(let* ((og "HAL")
       (key 6)
       (encrypted_text (encrypt og key))
```

```
        (decrypted_text (decrypt encrypted_text key)))
        ;; Just Call Solve and let it do its thing
        (solve og 26)

  (format t "Original Text: ~a ~%" og)
  ;; ENCRYPT
  (format t "Encrypted: ~a ~%" encrypted_text)
  ;; DECRYPT
  (format t "Decrypted: ~a ~%" decrypted_text))
```

Output

Case 1

```
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
Original Text: HAL
Encrypted: NGR
Decrypted: HAL
```

Case 2

```
Caesar 0: Lots of Irritating Superfluous Parentheses
Caesar 1: Mput pg Jssjubujoh Tvqfsgmvpvt Qbsfouiftft
Caesar 2: Nqvu qh Kttkvcvkpi Uwrgthnwqwu Rctgpvjgugu
Caesar 3: Orwv ri Luulwdwlqj Vxshuioxrxv Sduhqwkhvhv
Caesar 4: Psxw sj Mvvmxexmrk Wytivjpysyw Tevirxliwiw
Caesar 5: Qtyx tk Nwwnyfynsl Xzujwkqztzx Ufwjsymjxjx
Caesar 6: Ruzy ul Oxxozgzotm Yavkxlrauay Vgxktznkyky
Caesar 7: Svaz vm Pyypahapun Zbwlymsbvbz Whyluaolzlz
Caesar 8: Twba wn Qzzqbibqvo Acxmzntcwca Xizmvbpmama
Caesar 9: Uxcb xo Raarcjcrwp Bdynaoudxdb Yjanwcqnbnb
Caesar 10: Vydc yp Sbbsdkdsxq Cezobpveyec Zkboxdrococ
Caesar 11: Wzed zq Tcctеletyr Dfapcqwfzfd Alcpyespdpd
Caesar 12: Xafe ar Uddufmfuzs Egbqdrxgage Bmdqzftqeqe
Caesar 13: Ybgf bs Veevgngvat Fhcresyhbhf Cneragurfrf
Caesar 14: Zchg ct Wffwhohwbu Gidsftzicig Dofsbhvsgsg
Caesar 15: Adih du Xggxipixcv Hjetguajdjh Epgtciwthth
Caesar 16: Beji ev Yhhyjqjydw Ikfuhvbkeki Fqhudjxuiui
Caesar 17: Cfkj fw Ziizkrkzex Jlgviwclflj Grivekyvjvj
Caesar 18: Dglk gx Ajjalslafy Kmhwjxdmgmk Hsjwflzwkwk
Caesar 19: Ehml hy Bkkbmtmbgz Lnixkyenhnl Itkxgmaxlxl
Caesar 20: Finm iz Cllcnuncha Mojylzfoiom Julyhnbymym
Caesar 21: Gjon ja Dmmdovodib Npkzmagpjpn Kvmziocznzn
Caesar 22: Hkpo kb Ennepwpejc Oqlanbhqkqo Lwnajpdaoao
Caesar 23: Ilqp lc Foofqxqfkd Prmbocirlrp Mxobkqebpbp
Caesar 24: Jmrq md Gppgryrgle Qsncpdjsmsq Nypclrfcqcq
Caesar 25: Knsr ne Hqqhszshmf Rtodqektntr Ozqdmsgdrdr
Caesar 26: Lots of Irritating Superfluous Parentheses
Original Text: Lots of Irritating Superfluous Parentheses
Encrypted: Ruzy ul Oxxozgzotm Yavkxlrauay Vgxktznkyky
Decrypted: Lots of Irritating Superfluous Parentheses
```

Log

2018-11-14

- Somehow this language is easier than ML....possibly because of the compiler is at least slightly more helpful.
- Was able to write my offset function to return a character shifted by the key!
- Using this offset function the encrypt and decrypt functions become easier, using map(just like in ML) I can pass each character of a string into offset....this took a while was LISP made to be confusing?

2018-11-21

- ML is pissing me off again....please save me LISP
- Most of the resources online aren't that helpful....this one is a savior http://progopedia.com/implementation/steel-bank-common-lisp/
- Using that Manuel the Solve function is now easy, using the `for loop` syntax found there
- Only took like an extra 1.5 hours to get solve working, LISP wasn't as hard as I anticipated, however, it was just as frustrating as expected.
- In Conclusion, you will not find me using the language unless forced to, and even then I will plead for mercy.
- Steel Bank Common LISP compiler was extremely fast....however, the errors were absolutely not readable, just a big mess

Haskell

Code

```haskell
-- Daniel Nicolas Gisolfi
-- Haskell Cipher

import Data.Char (ord, chr, isUpper, isAlpha)

-- For each character in the given string offset it by the key
encrypt :: Int -> String -> String
encrypt = (<$>) . offset

-- Negate the key and then call encrypt as per usual
decrypt :: Int -> String -> String
decrypt = encrypt . negate

-- Given a character and a key shift the char and return it
offset :: Int -> Char -> Char
offset key ch
  | isAlpha ch = chr $ intAlpha + mod ((ord ch - intAlpha) + key) 26
  | otherwise = ch
  where
    intAlpha =
      ord
        -- Check for Case of Char
        (if isUpper ch
           then 'A'
           else 'a')

--  Gotta use recursion...
--  No matter what Call encrypt with the given vals and
--  print the results
```

```haskell
--  Check if the given lim has been reached otherwise
--  increment the cur number and call the func again
solve :: String -> Int -> Int -> IO ()
solve str cur lim = do
    let encode = encrypt (cur) str
    let out = "Ceasar " ++ show cur ++ ": " ++ encode
    putStrLn out
    let c = cur + 1
    if cur /= lim
    then solve str c lim
    else putStrLn "Done"

main :: IO ()
main = do
    -- Define the string and key
    let og = "HAL"
    let key = 6

    -- Using the returned value build nice output for print statement
    let encoded = encrypt key og
    let encrypted_out = "Encrypt: " ++ og ++ " -> " ++ encoded
    putStrLn encrypted_out

    -- Using the returned value build nice output for print statement
    let decoded = decrypt key encoded
    let decrypted_out = "Decrypt: " ++ encoded ++ " -> " ++ decoded
    putStrLn decrypted_out

    -- Pass the og string as well as a 0 for the cur value, giving the
    -- fucntion a place to start and finally give it a limit to reach
    solve og 0 26
```

Output

Case 1

```
Encrypt: HAL -> NGR
Decrypt: NGR -> HAL
Ceaser 0: HAL
Ceaser 1: IBM
Ceaser 2: JCN
Ceaser 3: KDO
Ceaser 4: LEP
Ceaser 5: MFQ
Ceaser 6: NGR
```

```
Ceaser 7: OHS
Ceaser 8: PIT
Ceaser 9: QJU
Ceaser 10: RKV
Ceaser 11: SLW
Ceaser 12: TMX
Ceaser 13: UNY
Ceaser 14: VOZ
Ceaser 15: WPA
Ceaser 16: XQB
Ceaser 17: YRC
Ceaser 18: ZSD
Ceaser 19: ATE
Ceaser 20: BUF
Ceaser 21: CVG
Ceaser 22: DWH
Ceaser 23: EXI
Ceaser 24: FYJ
Ceaser 25: GZK
Ceaser 26: HAL
Done
```

Case 2

```
Encrypt: Dark Themes are Superior in almost every Case -> Jgxq Znksky gxk Yavkxoux ot grsuyz kbkxe Igyk
Decrypt: Jgxq Znksky gxk Yavkxoux ot grsuyz kbkxe Igyk -> Dark Themes are Superior in almost every Case
Ceaser 0: Dark Themes are Superior in almost every Case
Ceaser 1: Ebsl Uifnft bsf Tvqfsjps jo bmnptu fwfsz Dbtf
Ceaser 2: Fctm Vjgogu ctg Uwrgtkqt kp cnoquv gxgta Ecug
Ceaser 3: Gdun Wkhphv duh Vxshulru lq doprvw hyhub Fdvh
Ceaser 4: Hevo Xliqiw evi Wytivmsv mr epqswx izivc Gewi
Ceaser 5: Ifwp Ymjrjx fwj Xzujwntw ns fqrtxy jajwd Hfxj
Ceaser 6: Jgxq Znksky gxk Yavkxoux ot grsuyz kbkxe Igyk
Ceaser 7: Khyr Aoltlz hyl Zbwlypvy pu hstvza lclyf Jhzl
Ceaser 8: Lizs Bpmuma izm Acxmzqwz qv ituwab mdmzg Kiam
Ceaser 9: Mjat Cqnvnb jan Bdynarxa rw juvxbc nenah Ljbn
Ceaser 10: Nkbu Drowoc kbo Cezobsyb sx kvwycd ofobi Mkco
Ceaser 11: Olcv Espxpd lcp Dfapctzc ty lwxzde pgpcj Nldp
Ceaser 12: Pmdw Ftqyqe mdq Egbqduad uz mxyaef qhqdk Omeq
Ceaser 13: Qnex Gurzrf ner Fhcrevbe va nyzbfg rirel Pnfr
Ceaser 14: Rofy Hvsasg ofs Gidsfwcf wb ozacgh sjsfm Qogs
Ceaser 15: Spgz Iwtbth pgt Hjetgxdg xc pabdhi tktgn Rpht
Ceaser 16: Tqha Jxucui qhu Ikfuhyeh yd qbceij uluho Sqiu
Ceaser 17: Urib Kyvdvj riv Jlgvizfi ze rcdfjk vmvip Trjv
```

```
Ceaser 18: Vsjc Lzwewk sjw Kmhwjagj af sdegkl wnwjq Uskw
Ceaser 19: Wtkd Maxfxl tkx Lnixkbhk bg tefhlm xoxkr Vtlx
Ceaser 20: Xule Nbygym uly Mojylcil ch ufgimn ypyls Wumy
Ceaser 21: Yvmf Oczhzn vmz Npkzmdjm di vghjno zqzmt Xvnz
Ceaser 22: Zwng Pdaiao wna Oqlanekn ej whikop aranu Ywoa
Ceaser 23: Axoh Qebjbp xob Prmboflo fk xijlpq bsbov Zxpb
Ceaser 24: Bypi Rfckcq ypc Qsncpgmp gl yjkmqr ctcpw Ayqc
Ceaser 25: Czqj Sgdldr zqd Rtodqhnq hm zklnrs dudqx Bzrd
Ceaser 26: Dark Themes are Superior in almost every Case
Done
```

Log

2018-11-21

- The syntax is interesting and seems to suffer one of the problems alan mentioned about PHP, too many ways to do the same thing.
- Its also interesting that you must define basically the path that a value takes through the functions as data types, not sure how I feel about it
- I'm unsure if this will be true for all of the functional languages however the process of the program was very similar to that of lisp, using an offset function that changes char values rather than the entire string.
- some of the syntax is very strange, especially the pipe character making an appearance
- I was struggling to find documentation on how to write a while or for loop in the language and then laughed when I realized it probably wouldn't have loops and instead the only recursion, of course, it does Haskell discovered or created the Y Combinator.
- I was intimidated at first about writing the solve function as a recursive loop however it turned out to be easier and fewer lines than some of the procedural languages with loops
- strange choice of concatenation symbols in the syntax `++`, gave me some weird errors when I was attempting to increment an int by doing `cur++`.
- Haskell grew on me during the process(only took me an hour and a half for the whole thing), it wasn't too difficult mostly due to the great documentation readily available. Still probably would rather write a CLI application in Java, C or even python before Haskell.

ML

Code

Sorry SML is not supported for syntax highlighting :(

```
(* DANIEL NICOLAS GISOLFI *)
(* CAESAR CHIPHER *)

(* The following two functions exist to make the code more
```

```
readable as well as keep me sane... for now *)

(* given an integer return the char value of it *)
fun toChar(i:int):char =
    chr (ord #"a" + i)
;

(* given a char return the numeric value of it *)
fun toInt(ch:char):int =
    ord ch - ord #"a"
;

(* Using the prev two funcs, shift the lowercase char by the key *)
fun shiftChar key ch: char =
    if Char.isLower ch
        then toChar((toInt(ch) + key) mod 26)
    else ch
;

(*
There is no easy way to do this without the map function
the difficult task was to figure out how to use the map function while
the fn your passing through takes more than one arguement
The documentation is lacking.
 *)
fun encrypt(str:string, key:int): string =
    let
        val chars = explode(str)
        val shiftedChars = map (shiftChar key) chars
    in
        implode shiftedChars
    end
;

(* Just negate it and send it through *)
fun decrypt(str:string,key:int): string =
    encrypt(str, ~key)
;

(* Recursion is easier than a for loop for this *)
fun solve(str:string, cur:int, lim:int) =
    let
        val c = cur + 1
        val curStr = Int.toString(cur)
        val encrypted = encrypt(str, cur)
    in
```

```
        (* print the encryption for the given key *)
        print("Ceasar " ^ curStr ^ ": " ^ encrypted ^ "\n");
        (* if the limit has not been reached call the fn again*)
        if cur <> lim
            then solve(str, c, lim)
        (* are we there yet? *)
        else print("done\n")
    end
;

(* TEST IT ALL *)
val og = "hal";
val _ = print("ORIGINAL ------->  " ^ og ^ "\n");
val _ = print("ENCRYPTED -------> " ^ encrypt(og, 6) ^ "\n");
val _ = print("DECRYPTED -------> " ^ decrypt(og, 6) ^ "\n");
solve(og, 0, 26);
```

Output

Case 1

```
ORIGINAL ------->  hal
ENCRYPTED -------> ngr
DECRYPTED -------> buf
Ceasar 0: hal
Ceasar 1: ibm
Ceasar 2: jcn
Ceasar 3: kdo
Ceasar 4: lep
Ceasar 5: mfq
Ceasar 6: ngr
Ceasar 7: ohs
Ceasar 8: pit
Ceasar 9: qju
Ceasar 10: rkv
Ceasar 11: slw
Ceasar 12: tmx
Ceasar 13: uny
Ceasar 14: voz
Ceasar 15: wpa
Ceasar 16: xqb
Ceasar 17: yrc
Ceasar 18: zsd
Ceasar 19: ate
Ceasar 20: buf
```

```
Ceasar 21: cvg
Ceasar 22: dwh
Ceasar 23: exi
Ceasar 24: fyj
Ceasar 25: gzk
Ceasar 26: hal
done
```

Case 2

```
ORIGINAL ------->  ml redefines the word fragile
ENCRYPTED -------> sr xkjklotky znk cuxj lxgmork
DECRYPTED -------> gf lyxyzchym nby qilx zluacfy
Ceasar 0: ml redefines the word fragile
Ceasar 1: nm sfefgjoft uif xpse gsbhjmf
Ceasar 2: on tgfghkpgu vjg yqtf htciknq
Ceasar 3: po uhghilqhv wkh zrug iudjloh
Ceasar 4: qp vihijmriw xli asvh jvekmpi
Ceasar 5: rq wjijknsjx ymj btwi kwflnqj
Ceasar 6: sr xkjklotky znk cuxj lxgmork
Ceasar 7: ts ylklmpulz aol dvyk myhnpsl
Ceasar 8: ut zmlmnqvma bpm ewzl nzioqtm
Ceasar 9: vu anmnorwnb cqn fxam oajprun
Ceasar 10: wv bonopsxoc dro gybn pbkqsvo
Ceasar 11: xw cpopqtypd esp hzco qclrtwp
Ceasar 12: yx dqpqruzqe ftq iadp rdmsuxq
Ceasar 13: zy erqrsvarf gur jbeq sentvyr
Ceasar 14: az fsrstwbsg hvs kcfr tfouwzs
Ceasar 15: ba gtstuxcth iwt ldgs ugpvxat
Ceasar 16: cb hutuvydui jxu meht vhqwybu
Ceasar 17: dc ivuvwzevj kyv nfiu wirxzcv
Ceasar 18: ed jwvwxafwk lzw ogjv xjsyadw
Ceasar 19: fe kxwxybgxl max phkw yktzbex
Ceasar 20: gf lyxyzchym nby qilx zluacfy
Ceasar 21: hg mzyzadizn ocz rjmy amvbdgz
Ceasar 22: ih nazabejao pda sknz bnwceha
Ceasar 23: ji obabcfkbp qeb tloa coxdfib
Ceasar 24: kj pcbcdglcq rfc umpb dpyegjc
Ceasar 25: lk qdcdehmdr sgd vnqc eqzfhkd
Ceasar 26: ml redefines the word fragile
done
```

Log

2018-11-13

- okay...so ML is frustrating as the compiler errors are not so helpful.
- However I was able to create a function to convert from a char to an int and vice versa. Using that I can now shift a single character...Now just to get it to do that for a whole list of chars.
- Alrighty so the compiler likes to say `caser.sml:24.39 Error: syntax error: inserting EQUALOP` but that's not the actual error, guess and check it is then.

2018-11-14

- I hate LISP, I'm returning to ML and its "bitchy" compiler...in more than just type declaration.
- One of my favorite parts of ML so far is the Syntax, I usually find semicolons ugly however the way ML implements them, (using them a bit more sparingly) is satisfying and actually helpful. (Possibly a good addition to python...final project maybe?)

2018-11-25

- Why am I still not done with ML I started this one first!?
- The error I was getting had something to do with simply using 'offset' as the function name..... I tried googling if its a keyword, doesn't look like it, possibly a reserved word.
- Thanks to the not so helpful compiler that bug took a few hours to discover.

2018-11-26

- The compiler is so finicky that I resorted to opening a small environment and typing in the commands one by one.
- This compiler is worse than LISP's
- Finally I have made progress, encrypt and decrypt work. Now time for solve, it may be easier to do it recursively as ML is hard to work with.
- It's an odd thing when recursion is easier to use than a for loop...I like it
- I will not be returning to this language unless I use a different compiler, im a fan of the syntax though.

Erlang

Code

```
1  -module(ceasar).
2  % Define all functions and how many parama they take
3  -export([main/0, offset/2, encrypt/2, decrypt/2, solve/3]).
4
5  % offset a single char by the key, if the key is out of range
```

```
% just return back the given char
offset(Char,Key) when (Char >= $A) and (Char =< $Z) or
                      (Char >= $a) and (Char =< $z) ->
  Offset = $A + Char band 32,
  N = Char - Offset,
  Offset + (N + Key) rem 26;
offset(Char, _Key) ->
  Char.

% Using Basically what I learned from ML, conver the string to chars
% and using map pass each element of the list to the offset function
% one at a time.
encrypt(Str, Key) ->
  lists:map(fun(Char) -> offset(Char, Key) end, Str).

% negate the key and call encrypt
decrypt(Str, Key) ->
  encrypt(Str, -Key).

% Base case -> if the limit has been reached stop looping...
solve(Str, Cur, Lim) when Cur == Lim+1 ->
  io:fwrite("Done\n");
% Otherwise call encrypt, print the result and keep looping
solve(Str, Cur, Lim) ->
  C = Cur + 1,
  Encrypted = encrypt(Str, Cur),
  io:format("Ceasar ~p: ~s~n", [Cur ,Encrypted]),
  solve(Str, C, Lim).

main() ->
  OG = "Rush's self titled EP is their best album",
  Key = 6,

  Encrypted = encrypt(OG, Key),
  Decrypted = decrypt(Encrypted, Key),

  % Printing stuff is quite ugly :(
  io:format("Original  ---> ~s~n", [OG]),
  io:format("Encrypted ---> ~s~n", [Encrypted]),
  io:format("Decrypted ---> ~s~n", [Decrypted]),
  solve(OG, 0, 26).
```

Output

Case 1

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

Case 2

```
Original  ---> Rush's self titled EP is their best album
Encrypted ---> Xayn'y ykrl zozrkj KV oy znkox hkyz grhas
Decrypted ---> R[sh's self titled EP is their best alb[m
Ceasar 0: Rush's self titled EP is their best album
Ceasar 1: Svti't tfmg ujumfe FQ jt uifjs cftu bmcvn
Ceasar 2: Twuj'u ugnh vkvngf GR ku vjgkt dguv cndwo
Ceasar 3: Uxvk'v vhoi wlwohg HS lv wkhlu ehvw doexp
Ceasar 4: Vywl'w wipj xmxpih IT mw xlimv fiwx epfyq
Ceasar 5: Wzxm'x xjqk ynyqji JU nx ymjnw gjxy fqgzr
Ceasar 6: Xayn'y ykrl zozrkj KV oy znkox hkyz grhas
Ceasar 7: Ybzo'z zlsm apaslk LW pz aolpy ilza hsibt
```

```
Ceasar 8: Zcap'a amtn bqbtml MX qa bpmqz jmab itjcu
Ceasar 9: Adbq'b bnuo crcunm NY rb cqnra knbc jukdv
Ceasar 10: Becr'c covp dsdvon OZ sc drosb locd kvlew
Ceasar 11: Cfds'd dpwq etewpo PA td esptc mpde lwmfx
Ceasar 12: Dget'e eqxr fufxqp QB ue ftqud nqef mxngy
Ceasar 13: Ehfu'f frys gvgyrq RC vf gurve orfg nyohz
Ceasar 14: Figv'g gszt hwhzsr SD wg hvswf psgh ozpia
Ceasar 15: Gjhw'h htau ixiats TE xh iwtxg qthi paqjb
Ceasar 16: Hkix'i iubv jyjbut UF yi jxuyh ruij qbrkc
Ceasar 17: Iljy'j jvcw kzkcvu VG zj kyvzi svjk rcsld
Ceasar 18: Jmkz'k kwdx laldwv WH ak lzwaj twkl sdtme
Ceasar 19: Knla'l lxey mbmexw XI bl maxbk uxlm teunf
Ceasar 20: Lomb'm myfz ncnfyx YJ cm nbycl vymn ufvog
Ceasar 21: Mpnc'n nzga odogzy ZK dn oczdm wzno vgwph
Ceasar 22: Nqod'o oahb pephaz AL eo pdaen xaop whxqi
Ceasar 23: Orpe'p pbic qfqiba BM fp qebfo ybpq xiyrj
Ceasar 24: Psqf'q qcjd rgrjcb CN gq rfcgp zcqr yjzsk
Ceasar 25: Qtrg'r rdke shskdc DO hr sgdhq adrs zkatl
Ceasar 26: Rush's self titled EP is their best album
Done
```

Log

2018-11-24

- I have so many questions for Joe Armstrong, here are a few...
    - Why on earth would you force variables to start with Uppercase characters?
    - Why would you not implement better errors for detecting variables with lowercase characters?
    - Why the commas? I'm not a fan of semicolons but I have to admit they look better than commas.
- Instead of telling you that you've used a lowercase variables erlang just mentions that the left and right side of the assignment does not match, which leads one to believe you're returning the wrong type rather than just breaking syntax with your variable name.
- I'm not a fan of the almost assumed return value in many of these languages like Erlang and ML, instead of saying return you just kinda leave it there, it's not very readable.
- The only thing I have working is a function to offset the char given a key, progress but not much...

2018-11-27

- I'm back with a vengeance!
- I have the encrypt function working now at least, this compiler is very paticular, some of the code was only working in main and not encrypt

- Also decrypt is done cuz passing a negative key is easy
- Finally its time for solve, gonna do it recursively again to embrace function programming(also I figured the logic of it out back in the Haskell one so I'm just kind rewriting the same function)
- Trying to have an if check in the Solve function for my base case, it will not compile here it is...

```
% I put this snippet in as this is not how the final version looks
solve(Str, Cur, Lim) ->
  C = Cur + 1,
  Encrypted = encrypt(Str, Cur),
  io:format("Ceasar ~p: ~s~n", [Cur ,Encrypted]),
  if
      Cur == Lim ->
        io:fwrite("Done\n");
      ture ->
        solve(Str, C, Lim);
  end.
```

- So looking at this above snippet there should be no error, this is a simple if expression, the true acts as an else clause. However this will not compile, I used multiple resources to find this syntax, why doesnt it work?
- The reason I later found out is that apparantly the conditions must return a value(neither does) as well as that each return value must be of the same type. This limits if expressions so much that I just refused to use them.
- The solution I found was to use what I can only describe as function cases. (also used in the offset function)
- Erlang may be powerful, but I would need a very good reason to use it at all.

Scala - Functionaly

Code

```
// Daniel Nicolas Gisolfi
// Ceasar Cipher in Scala done functionaly
object Ceasar {
    // Given a char shift it by the key and convert
    // back to char to return
    def offset(char:Char, key:Int):Char =  {
        // check if the letters are going out of bounds
        if (char.toInt + key > 'Z'.toInt) {
            return (char.toInt + key -26).toChar
        } else {
            return (char.toInt + key).toChar
```

```
        }
    }

    // Iterate through each character of the string and
    // for each char, send it to offset() and then concat it
    def encrypt(str:String, key:Int):String = {
        var encoded:String = "";
        str.foreach((char: Char) => encoded = encoded.concat((offset(char, key).toString)));
        return encoded
    }

    // Just call encrypt but negate the key first
    def decrypt(str:String, key:Int):String = {
        return  Ceasar.encrypt(str, -key)
    }

    // for the desired limit solve the cipher for all values
    def solve(str:String, cur:Int, lim:Int) {
        println("Ceasar " + cur + ": " + Ceasar.encrypt(str, cur));
        // Base case, check if the limits been reached,
        // otherwise call the function again
        if(cur != lim) {
            solve(str, cur+1 , lim);
        } else {
            println("Done");
        }
    }

    def main(args: Array[String]): Unit = {
        val og = "HAL";
        val key = 6;

        val encrypted = Ceasar.encrypt(og, key);
        val decrypted = Ceasar.decrypt(encrypted, key);

        println("Original  --> " + og);
        println("Encrypted --> " + encrypted);
        println("Decrypted --> " + decrypted);
        Ceasar.solve(og,0,26);
    }

}
```

Output

Case 1

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

Case 2

```
Original  --> ORANGUTAN
Encrypted --> UXGTMAZGT
Decrypted --> ORANGUTAN
Ceasar 0: ORANGUTAN
Ceasar 1: PSBOHVUBO
Ceasar 2: QTCPIWVCP
Ceasar 3: RUDQJXWDQ
Ceasar 4: SVERKYXER
Ceasar 5: TWFSLZYFS
```

```
Ceasar 6: UXGTMAZGT
Ceasar 7: VYHUNBAHU
Ceasar 8: WZIVOCBIV
Ceasar 9: XAJWPDCJW
Ceasar 10: YBKXQEDKX
Ceasar 11: ZCLYRFELY
Ceasar 12: ADMZSGFMZ
Ceasar 13: BENATHGNA
Ceasar 14: CFOBUIHOB
Ceasar 15: DGPCVJIPC
Ceasar 16: EHQDWKJQD
Ceasar 17: FIREXLKRE
Ceasar 18: GJSFYMLSF
Ceasar 19: HKTGZNMTG
Ceasar 20: ILUHAONUH
Ceasar 21: JMVIBPOVI
Ceasar 22: KNWJCQPWJ
Ceasar 23: LOXKDRQXK
Ceasar 24: MPYLESRYL
Ceasar 25: NQZMFTSZM
Ceasar 26: ORANGUTAN
Done
```

Log

2018-11-27

- I remember nothing of Scala from 2 months ago, back to google!
- I saw "def" and got excited, sadly this is not python...woulda been faster to test it. The only thing I remember from my time with Scala was how horribly slow the compiler was... it's still very slow, even for a 30 line program.
- For how slow the compiler it is, it's my favorite compiler of all 5. The error messages are actually helpful(unless a runtime error occurs then its back to hard to read java errors)
- I'm starting to enjoy programming functionally, it makes everything look very condense and clean...however readability suffers greatly.
- Recursion is so satisfying to use(I wrote the same function for solve in 4 of the languages)
- Somehow I found functional programming to be easier than Fortran and COBOL even with the lacking documentation for a lot of the languages
- The suggestion to save Scala till last was great advice, using the same flow of the ciphers done in ML and Erlang especially I was able to create what I believe is a functionaly programed Cipher in around 1 to 2 hours