Functional Programming

Aidan Carr
Aidan.Carr1@marist.edu

April 11, 2025

1 Introduction

1.1 Directions

Develop a set of functions that will allow you to **encrypt** a string using a Caesar cipher. Develop a set of functions that will allow you to **decrypt** a string using a Caesar cipher. Develop a set of functions that will help you to **solve** (break) a Caesar cipher.

Implement the above functions for five *functional* programming languages.

1.2 Languages and Time Predictions

Before the assignment, I predicted how long it would take to write the Caesar Cipher programs in each language.

1. JavaScript, predicted 1:30
2. Scala, predicted 1:30
3. ML, predicted 1:30
4. LISP, predicted 2:30
5. Erlang, predicted 2:30

Note: This is the order I wrote my programs in.

2 JavaScript

2.1 JavaScript Code

Woah JavaScript? This one is so new to me! Let's see how I did it!

```
1 //encrypt
2 console.log(encrypt("the snack that smiles back", 16));
3 //decrypt
4 console.log(decrypt("JPMT XLi WGVMTX", 4));
5 //solve
6 solve("oetpu wjz oqjju", 7);
```

```
function encrypt(text, value){
    //base case, no more characters
    if (text.length == 0) {
        return "";
    }

    //recursive, encrypt character and the next character(s)
    else {
        //fix value
        value = value % 26;
        //get head (in caps)
        headASCII = text.toUpperCase().charCodeAt(0)

        //letter
        if ('A'.charCodeAt(0) <= headASCII && headASCII <= 'Z'.charCodeAt(0)) {
            //shift
            shiftedASCII = headASCII + value;

            //loop back after Z
            if (shiftedASCII > 'Z'.charCodeAt(0)) {
                shiftedASCII -= 26;
            }
        }
        //not a letter
        else {
            shiftedASCII = headASCII;
        }

        //this and next character(s)
        return String.fromCharCode(shiftedASCII) + encrypt(text.substring(1), value);
    }
}

function decrypt(text, value) {
    return encrypt(text, 26-value);
}

function solve(text, value) {
    //base case, final value 0
    if (value <= 0) {
        console.log("Caesar 0: " + encrypt(text, 0));
    }
    //recursive, this and next value(s)
    else {
        console.log("Caesar " + value + ": " + encrypt(text, value));
        solve(text, value-1);
    }
}
```

2.2 JavaScript Test Output

```
JXU IDQSA JXQJ ICYBUI RQSA
FLIP THE SCRIPT
Caesar 7: VLAWB DQG VXQQB
Caesar 6: UKZVA CPF UWPPA
Caesar 5: TJYUZ BOE TVOOZ
Caesar 4: SIXTY AND SUNNY
Caesar 3: RHWSX ZMC RTMMX
Caesar 2: QGVRW YLB QSLLW
Caesar 1: PFUQV XKA PRKKV
Caesar 0: OETPU WJZ OQJJU
```

2.3 JavaScript Thoughts

I have had a lot of experience coding in JavaScript, so this language was easy to write a functional Caesar cipher in. I love JavaScript's readability, there are minimal keywords, and a good use of symbols. Braces allow `if, else`, and function statements to be easily separated in terms of scope.

Writeability is also no issue. Lines always end in a semi colon, functions are all called with parenthesis and a period for class methods. The equals operator is a bit overused and we see `=`, `==`, and `===` being used which can be difficult to understand as a beginner. But this is one of the few languages I can assign `value % 26` to `value` itself.

A downside to JavaScript is the lack of type enforcement. Type does not have to be declared which can both help and hurt writeability, making programming easier, but unsafe at times (TypeScript solves this). Additionally, some functions like `charCodeAt(0)` are a bit long compared to other ASCII operators in the other languages.

And finally, JavaScript has one of the highest online presences. Several sites have in-depth lessons on any piece of coding in JavaScript. This includes Google's AI– asking a quick question on Google quickly gives you an easy explanation and example of how to use the function. Overall, high readability, writeability, and community/cost.

3 Scala

3.1 Scala Code

Hey I remember this one.

```
object CaesarCipher {
  def main(args: Array[String]): Unit = {


    //Encrypt Function
    def encrypt(text: String, rawValue: Int): String = {

      //base case, no more characters
      if (text.length == 0) {
        return ""
      }

      //recursive, encrypt char and the next char(s)
      else {
        //fix the value
        var value = (rawValue % 26)
        //get head (in caps)
        var headASCII = text.toUpperCase.head.toInt
        var shiftedASCII = headASCII

        //letter
        if ('A'.toInt <= headASCII && headASCII <= 'Z'.toInt) {
          //shift
          shiftedASCII = headASCII + value

          //wrap around after Z
          if (shiftedASCII > 'Z'.toInt) {
            shiftedASCII = shiftedASCII - 26
          }
        }

        //not a letter
        else {
          //keep default
        }

        //return this and next character(s)
        var head = shiftedASCII.toChar
        //println(head)
        return head.toString + encrypt(text.substring(1), value)
      }
    }
```

```
45      //Decrypt Function
46      def decrypt(message: String, caesarValue: Int): String = {
47        //Modify the value
48        var value = (caesarValue % 26)
49        //Encrypt with the compliment
50        var decryptedText = encrypt(message, 26-value)
51        return decryptedText
52      }
53
54
55      //Solve Function
56      def solve(text: String, value: Int): Unit = {
57        //base case, final value 0
58        if (value <= 0) {
59          println("Caesar 0: " + encrypt(text, 0))
60        }
61        //recursive, this and the next value(s)
62        else {
63          println("Caesar " + value + ": " + encrypt(text, value))
64          solve(text, value-1)
65        }
66      }
67
68
69      //Encrypt
70      println(encrypt("Scala bowl of cereal", 58))
71      //Decrypt
72      println(decrypt("YiGRG HUCR ul iKXKGR", 58))
73      //Solve
74      solve("PKWOP", 11)
75
76    }
77  }
```

3.2 Scala Test Output

```
YIGRG HUCR UL IKXKGR
SCALA BOWL OF CEREAL
Caesar 11: AVHZA
Caesar 10: ZUGYZ
Caesar 9: YTFXY
Caesar 8: XSEWX
Caesar 7: WRDVW
Caesar 6: VQCUV
Caesar 5: UPBTU
Caesar 4: TOAST
Caesar 3: SNZRS
Caesar 2: RMYQR
Caesar 1: QLXPQ
Caesar 0: PKWOP
```

3.3 Scala Thoughts

I enjoyed coming back to Scala because of its quick familiarity. The braces are prominent in this language and keep the readability high. Print statements are obvious and hide detail which future print statements will not. We still have variables and they still do not require a variable type which hurts Scala's reliability. However, variables can be declared with types which I took advantage of on line 6.

I like seeing the word `return`, other languages lack this and I think it helps readability, but this is mainly because Scala is a procedural language, and the following functional language don't feel like they are missing out on the `return` keyword.

Functions require return values, good for readability and reliability, so that's one point over JavaScript. Although `Unit` might be a bit less readable than `void` it still works.

Strings are not mutable in Scala, so I had to go back to how we originally learned to program, not too much of a drawback, just a little algorithm edit. Also, capitalization mattered here, forced uniformity which I like.

Overall readability and writeability are very high, there are less strict rules to follow here compared to many languages later.

4 ML

4.1 ML Code

```
fun encrypt (text:string) (rawValue:int) =
  (*base case, no more characters*)
  if size text = 0 then
    ""

  (*recursive, encrypt character and the next*)
  else
    let
      (*fix value*)
      val value = rawValue mod 26

      (*convert to list and behead it*)
      val textList = explode text
      val headChar = hd textList

      (*get head in caps*)
      val headASCII = Char.ord (Char.toUpper headChar)

      (*shift letter*)
      val shiftedASCII =
        if Char.ord(#"A") <= headASCII andalso headASCII <= Char.ord(#"Z") then
          let
          in
              if headASCII + value <= Char.ord(#"Z") then
              headASCII + value
            else (*loop back after Z*)
              headASCII + value -26
          end

        (*no letter means no shift *)
        else
          headASCII

      (*convert to char*)
      val shiftedChar = Char.chr shiftedASCII

    in
      (*return this and future encryptions*)
      str shiftedChar ^ encrypt (implode (tl textList)) value
    end;

fun decrypt (text:string) (value:int) =
  encrypt text (26-value);

fun solve (text:string) (value:int) =

  (*last solve value 0*)
  if value <= 0 then
    print("Caesar 0: " ^ encrypt text 0 ^ "\n")

  (*print current solve value and then the next*)
  else
    let
        val _ = print("Caesar " ^ Int.toString value ^ ": " ^ encrypt text value ^"\n")
    in
        solve text (value-1)
    end;

(*lets test them out*)
encrypt "crux" 6;
decrypt "Uf pfl befn kyv Dlwwze Dre" 17;
solve "HLRXDZIV" 11;
```

4.2 ML Test Output

```
"IXAD"
"DO YOU KNOW THE MUFFIN MAN"
Caesar 11: SWCIOKTG
Caesar 10: RVBHNJSF
Caesar 9: QUAGMIRE
Caesar 8: PTZFLHQD
Caesar 7: OSYEKGPC
Caesar 6: NRXDJFOB
Caesar 5: MQWCIENA
Caesar 4: LPVBHDMZ
Caesar 3: KOUAGCLY
Caesar 2: JNTZFBKX
Caesar 1: IMSYEAJW
Caesar 0: HLRXDZIV
```

4.3 ML Thoughts

Pretty difficult to write in. Not a traditional programming language from what I am used to, but a command line so I cant go back and edit lines or retry bits of code. Semi colons exist but they were not very consistent, hurting writeability. I wasn't not sure if you use them inside of functions. With a ton of nested if statement, the language got tricky to read and write. Keywords like `let` and `in` had to be used because if statements only executed one statement (unless these words were used).

No clear line ending made this functional language harder to write. However, it showed itself to be versatile, allowing me to write an if block INSIDE of an assignment statement, no function or `return` value needed when assigned `shiftedASCII`.

ChatGPT helped once again because I would constantly get errors for the longer code. These errors included missing else statements, illegal to reassign with writing `val`, and wrong way of function calling (that took some time to get used to). I wasn't sure how to do a print statement inside an `else` statement, so it recommended I declare a variable `let val _ = print(... ^..)` on lines 53 and 54. It also introduced me to the `^` concatenation operator, which I think is one of the true orthogonality flaws of the language when `+` or `,` work in other languages.

The more I wrote in this functional language, the more I realized I could keep shrinking down the code. Variables did not need to be declared because many would only be used once, therefore it could mostly all be in one line, reminding me of lambda calculus. However, to maintain readability, I kept it close in style to procedural programming. Using anonymity could allow a language like this to lack readability.

Resources online helped with many aspects of learning the language so there is still a community with answers. The overall readability was low for a procedural language but high in terms of the next two because of the simplicity. No big confusing semi colons or multiple symbols, only symbols with operators, keywords like `then`, `let`, `in`, `if`, `else`, `end`, `val`.

5 LISP

5.1 LISP Code

LISP? Where are you bud? Oh next page.

```
;ENCRYPT FUNCTION
(defun encrypt (text rawValue)

  ;base case, no more characters
  (if (= (length text) 0)
    ""

  ;else
    (let
      (
        ;fix value
        (value (mod rawValue 26))

        ;get head in caps
        (headChar (char-upcase (aref (subseq text 0 1) 0)))
        (shiftedHead " ")

        ;ASCIIs
        (headASCII 0)
        (shiftedASCII 0)
      )
        (setf headASCII (char-code headChar))
        (setf shiftedASCII headASCII)
        (setf shiftedHead headChar)

          ;letter
          (if (and (<= (char-code #\A) headASCII)
                   (<= headASCII (char-code #\Z)))
            (progn
              ;shift
              (setf shiftedASCII (+ headASCII value))

              ;loop back after Z
              (if (> shiftedASCII (char-code #\Z))
                (setf shiftedASCII (- shiftedASCII 26))
              )
            nil)

          ;else ()
          ;not a letter, leave as original
          )


      (setf shiftedHead (code-char shiftedASCII))
      (concatenate 'string

          (format nil "~A" shiftedHead)
          (encrypt (subseq text 1) value)
      )
    )
  )
)


;DECRYPT FUNCTION
(defun decrypt (text rawValue)
  (encrypt text (- 26 rawValue))
)
```

```lisp
;SOLVE FUNCTION
(defun solve (text value)

  (if ( <= value 0)
    ;base case, final value = 0
    (concatenate 'string
      (format nil "Caesar 0: ")
      (encrypt text 0)
    )
    ;else, recursive, do this value and next value(s)
    (concatenate 'string
      (format nil "Caesar ~A: " value)
      (encrypt text value)
      (format nil "~%")
      (solve text (- value 1))
    )
  )
)


;MAIN FUNCTION
;encrypt this
(format t "~A" (encrypt "Chicken Teriyaki" 33))
(write-line "")
;decrypt that
(format t "~A" (decrypt "M HS RSX LEZI E PMXLT" 4))
(write-line "")
;try solving this one julius!
(format t "~A" (solve "PXFKQ ABKFP, IBJLVKB" 16))
```

5.2 LISP Test Output

```
JOPJRLU ALYPFHRP
I DO NOT HAVE A LITHP
Caesar 16: FNVAG QRAVF, YRZBLAR
Caesar 15: EMUZF PQZUE, XQYAKZQ
Caesar 14: DLTYE OPYTD, WPXZJYP
Caesar 13: CKSXD NOXSC, VOWYIXO
Caesar 12: BJRWC MNWRB, UNVXHWN
Caesar 11: AIQVB LMVQA, TMUWGVM
Caesar 10: ZHPUA KLUPZ, SLTVFUL
Caesar 9: YGOTZ JKTOY, RKSUETK
Caesar 8: XFNSY IJSNX, QJRTDSJ
Caesar 7: WEMRX HIRMW, PIQSCRI
Caesar 6: VDLQW GHQLV, OHPRBQH
Caesar 5: UCKPV FGPKU, NGOQAPG
Caesar 4: TBJOU EFOJT, MFNPZOF
Caesar 3: SAINT DENIS, LEMOYNE
Caesar 2: RZHMS CDMHR, KDLNXMD
Caesar 1: QYGLR BCLGQ, JCKMWLC
Caesar 0: PXFKQ ABKFP, IBJLVKB
```

5.3 LISP Thoughts

Didn't like LISP I didn't like LISP. I took me about 30 minutes to even figure out what I was doing because SBCL wasn't working right and online examples wouldn't work in the online IDE I was using. But then it started to work and I asked ChatGPT for some basic tips:

- im writing code in commonlisp for the first time with coding experience already. give me the basics (variable types, functions, printing)
- how can i get the tail or head substring of a string
- why isnt this working: [code]

The error messages for LISP were way too much for me, not helpful in telling me where parentheses should and should not be. This was an error for one misplaced parenthesis:

```
Unhandled SB-C::INPUT-ERROR-IN-LOAD in thread #<SB-THREAD:THREAD "main thread" RUNNING
                          {10005184C3}>:
READ error during LOAD:

  end of file on #<SB-SYS:FD-STREAM for "file /box/script.lisp" {1001532ED3}>

  (in form starting at line: 1, column: 0, file-position: 0)

Backtrace for: #<SB-THREAD:THREAD "main thread" RUNNING {10005184C3}>
0: (SB-DEBUG::DEBUGGER-DISABLED-HOOK #<SB-C::INPUT-ERROR-IN-LOAD {1001542213}> #<unused argument> :QUIT
     T)
1: (SB-DEBUG::RUN-HOOK *INVOKE-DEBUGGER-HOOK* #<SB-C::INPUT-ERROR-IN-LOAD {1001542213}>)
2: (INVOKE-DEBUGGER #<SB-C::INPUT-ERROR-IN-LOAD {1001542213}>)
3: (ERROR #<SB-C::INPUT-ERROR-IN-LOAD {1001542213}>)
4: (SB-C:COMPILER-ERROR SB-C::INPUT-ERROR-IN-LOAD :CONDITION #<END-OF-FILE {10015421E3}> :POSITION 0 :
    LINE/COL NIL :STREAM #<SB-SYS:FD-STREAM for "file /box/script.lisp" {1001532ED3}>)
5: (SB-C::%DO-FORMS-FROM-INFO #<CLOSURE (LAMBDA (SB-KERNEL:FORM &KEY :CURRENT-INDEX &ALLOW-OTHER-KEYS)
    :IN SB-INT:LOAD-AS-SOURCE) {10015416CB}> #<SB-C::SOURCE-INFO {1001541693}> SB-C::INPUT-ERROR-IN-
    LOAD)
6: (SB-INT:LOAD-AS-SOURCE #<SB-SYS:FD-STREAM for "file /box/script.lisp" {1001532ED3}> :VERBOSE NIL :
    PRINT NIL :CONTEXT "loading")
7: ((FLET SB-FASL::THUNK :IN LOAD))
8: (SB-FASL::CALL-WITH-LOAD-BINDINGS #<CLOSURE (FLET SB-FASL::THUNK :IN LOAD) {7F374143F69B}> #<SB-SYS:
    FD-STREAM for "file /box/script.lisp" {1001532ED3}>)
9: ((FLET SB-FASL::LOAD-STREAM :IN LOAD) #<SB-SYS:FD-STREAM for "file /box/script.lisp" {1001532ED3}>
    NIL)
10: (LOAD #<SB-SYS:FD-STREAM for "file /box/script.lisp" {1001532ED3}> :VERBOSE NIL :PRINT NIL :IF-DOES
    -NOT-EXIST T :EXTERNAL-FORMAT :DEFAULT)
11: ((FLET SB-IMPL::LOAD-SCRIPT :IN SB-IMPL::PROCESS-SCRIPT) #<SB-SYS:FD-STREAM for "file /box/script.
    lisp" {1001532ED3}>)
12: ((FLET SB-UNIX::BODY :IN SB-IMPL::PROCESS-SCRIPT))
13: ((FLET "WITHOUT-INTERRUPTS-BODY-2" :IN SB-IMPL::PROCESS-SCRIPT))
14: (SB-IMPL::PROCESS-SCRIPT "script.lisp")
15: (SB-IMPL::TOPLEVEL-INIT)
16: ((FLET SB-UNIX::BODY :IN SAVE-LISP-AND-DIE))
17: ((FLET "WITHOUT-INTERRUPTS-BODY-7" :IN SAVE-LISP-AND-DIE))
18: ((LABELS SB-IMPL::RESTART-LISP :IN SAVE-LISP-AND-DIE))
```

I had to ask ChatGPT what the issue was and the problem was quickly found and a solution was given with an explanation. It explained why and where the parentheses were out of order.

Additonal notes I had: After an hour, I had some progress finally getting to ASCII values. I was not liking the readability of all the function names like `format nil` or `concatenate 'string` or `setf` or `aref` and seeing `let` again. There were WAYYY too many parentheses and stupid little rules to everything. The order of writing a function or any operation first before the parameter got old real quick.

This language had very low writeability. Lines 27 and 28 had been much shorter in other languages, but two conditionals with an `and` operator makes this section look difficult. Also the print statements look kinda stupid. I like the idea of writing `"Caesar Ã: " value` to insert `value` inside the string, but writing `format t` or `format nil` just doesn't look good at all.

ChatGPT helped in terms of returning strings, I did not get to learn what `nil` meant, but it was needed apparently. The readability of this is not too bad. The language follows a certain set and rules and truly sticks to them: `(operator parameter parameter)`.

6 Erlang

6.1 Erlang Code

Kinda sounds like a dude's name, something like Erlang Weaselman.

```
1 %MAIN FUNCTION
2 main(_) ->
3
4     io:format("~s~n", [encrypt("Toejam and Earlang", 1991)]),
5     io:format("~s~n", [decrypt("AZBNM ZMC DFFR", 51)]),
6     io:format("~s~n", [solve("JXU idqsa JXQJ ICYBUI rqsa!", 14)]).
```

```
%ENCRYPT FUNCTION
encrypt(Text, RawValue) ->

    % base case, empty string, done
    if length(Text) =< 0 ->
      "";

    %else, encrypt char + the next
    true ->

      %fix value
      Value = RawValue rem 26,
      %head in caps
      HeadString = string:to_upper(string:substr(Text, 1, 1)),
      %head as ascii int
      [HeadASCII] = HeadString,

      %tail
      Tail = string:substr(Text, 2, length(Text)-1),

      ShiftedASCII =
        %letter
        if HeadASCII >= $A, HeadASCII =< $Z ->
          %shift
          Shift = HeadASCII + Value,

          %pull it back if shift went too far
          if Shift > $Z ->
            Shift - 26;
          true ->
            Shift
          end;

        %not a letter
        true ->
          HeadASCII
        end,

    %return this and next character(s)
    [ShiftedASCII] ++ encrypt(Tail, Value)
    end.


%DECRYPT FUNCTION
decrypt(Text, RawValue) ->
  Value = RawValue rem 26,  %because remainder doesnt work on negative numbers
  encrypt(Text, 26-Value).


%SOLVE FUNCTION
solve(Text, Value) ->
  if Value =< 0 ->
    "Caesar 0: " ++ encrypt(Text, 0);
  true ->
    "Caesar " ++ integer_to_list(Value) ++ ": " ++
    encrypt(Text, Value) ++ "\n" ++
    solve(Text, Value-1)
  end.
```

6.2 Erlang Test Output

```
IDTYPB PCS TPGAPCV
BACON AND EGGS
Caesar 14: XLI WREGO XLEX WQMPIW FEGO!
Caesar 13: WKH VQDFN WKDW VPLOHV EDFN!
Caesar 12: VJG UPCEM VJCV UOKNGU DCEM!
Caesar 11: UIF TOBDL UIBU TNJMFT CBDL!
Caesar 10: THE SNACK THAT SMILES BACK!
Caesar 9: SGD RMZBJ SGZS RLHKDR AZBJ!
Caesar 8: RFC QLYAI RFYR QKGJCQ ZYAI!
Caesar 7: QEB PKXZH QEXQ PJFIBP YXZH!
Caesar 6: PDA OJWYG PDWP OIEHAO XWYG!
Caesar 5: OCZ NIVXF OCVO NHDGZN WVXF!
Caesar 4: NBY MHUWE NBUN MGCFYM VUWE!
Caesar 3: MAX LGTVD MATM LFBEXL UTVD!
Caesar 2: LZW KFSUC LZSL KEADWK TSUC!
Caesar 1: KYV JERTB KYRK JDZCVJ SRTB!
Caesar 0: JXU IDQSA JXQJ ICYBUI RQSA!
```

6.3 Erlang Thoughts

Also was not a big fan of Erlang, really saving the worst for last here. The arrow symbol `->` was kinda cool so I'll give them a point there. The readability is slightly improved from LISP because operators go in order again:

```
if length(Text) =< 0 ->
```

Readability is once again tarnished by the multiple line ending punctuations. Punctuation included `, ; .  ->` and also sometimes nothing at all. `if else` statements are replaced with `if` statements followed by another `if` or (how ChatGPT recommended) `true ->` to get everything *else* that was left.

Speaking of ChatGPT, let's see what I asked it:

- im learning erlang. i already know how to code in other langauges. give me the basics. print, funtoiins, variables, if statemetns, and the quirks of erlang (nice spelling)
- defeine the substring method (nice spelling again)
- whats the problem here: [code]

Again, ChatGPT worked well with helping solve errors and fixing code that was confusing to write. One tiny issue I ran into was forgetting to name variables with a Capital letter. My variable `shift` was not working for the longest time, but the error was not helpful. I can see why this is implemented, capital letters allow all those words to be reserved for only variables, which makes sense, so good job on readability there.

The substring function wasn't the greatest, based on the way I learned it, a bit of a verbose tool. I did like the `++` symbol for string concatenation. Also, behind the scenes, strings are just list of ASCII numbers, super helpful for writing Caesar Cipher code. I only had to write `[ShiftedASCII]` to get the string version.

Overall, readability is high up, writeability was a little difficult at first, but I can see myself warming up to it, not having to write 10 Caesar Cipher programs before learning this one. Community for Erlang was somewhat there, some sites like tutorialspoint.com helped and mostly ChatGPT gave the best pointers for this language and the others.

7 Rankings

My overall rankings in terms of fun, readability, writeability, and vibezzz:

1. JavaScript
    a) The language I am most used to, I like the simplicity of most of the functions and the clean look of it all.
    b) Time taken: **0:47** (Predicted 1:30)
2. Scala
    a) Fun language to come back to, high readability and writeability. Good use of symbols.
    b) Time taken: **0:58** (Predicted 1:30)
3. ML
    a) Annoying and difficult to learn but a good gateway into functional programming.
    b) Time taken: **2:49** (Predicted 1:30)
4. Erlang
    a) Did not like the errors reporting, inconsistent symbols, and silly rules.
    b) Time taken: **2:11** (Predicted 2:30)
5. LISP
    a) Hated the excessive amount of parentheses and the overly consistent order of functions followed by parameters.
    b) Time taken: **2:26** (Predicted 2:30)

Overall, it took me 9 hours and 11 minutes of coding when I predicted 9:30. Pretty close I'd say. A lot of timer pauses to cool off.