

## Lisp in Summer Projects Submission

<b>Submission Date</b>	2013-10-04 15:57:28
<b>Full Name</b>	Jaeden Amero
<b>Country</b>	USA
<b>Project Name</b>	A Random QSO Generator
<b>Type of software</b>	other
<b>General category</b>	library
<b>LISP dialect</b>	Racket
<b>GitHub URL</b>	<a href="https://github.com/Patater/qso-generator">https://github.com/Patater/qso-generator</a>
<b>Did you start this project?</b>	Yes, all the code is written by me
<b>Project Description</b>	I want to describe my project in this form.
<b>Purpose</b>	Existing Koch-method Morse code training software waits until all symbols are learned before giving a Morse code student practice with Morse code conversations (known QSOs in amateur radio parlance). The point of this program is to generate random QSOs given a subset of symbols, so that the student can gain practice with the QSO format even before they know all the Morse code symbols.
<b>Function</b>	Using functions from this program, we can generate random callsigns; word-level similar texts; letter-level similar texts; texts that resemble the original, but are created with a subset of the original alphabet; and random QSOs given partial alphabets.
<b>Motivation</b>	I wanted to learn Lisp. I heard Lisp was good at natural language processing (NLP) tasks, so I decided to do a project that involves NLP.
<b>Audience</b>	I wrote this for myself, for later use as part of a Morse code training program that uses the Koch method of learning.
<b>Methodology</b>	The program works by following a set of production rules to produce a random QSO.  A QSO usually contains the callsigns of those involved in

the QSO. The first thing the program does, when asked to produce a random QSO via `generate-random-qso`, is generate two callsigns, one for each participant in the conversation.

Callsigns are generated according to another set of production rules. They are made up of a country prefix, a separating numeral, and a suffix. As best as can be done given a partial alphabet, we attempt to create a callsign-looking string according to rules imposed upon us by the official callsign format.

After the callsigns are generated, we generate the body (main text) of the QSO. We construct a hierarchical Markov chain from our QSO corpus. The hierarchical Markov chain is composed of a 2-word Markov chain, a 1-word Markov chain, a 3-letter Markov chain, a 2-letter Markov chain, and a 1-letter Markov chain.

We use a hierarchical Markov chain so that if there is no valid transition (i.e. a transition to a new state that remains within our alphabet) at the highest level Markov chain, we will fall back onto the lower level chain to see if there is a valid transition there. If none of the chains have a valid transition, we will select a random symbol from our alphabet to be used to compose the next state. This allows us to make a best effort to generate text that looks like a QSO while remaining within our alphabet.

In order to combat repetitious looking output and to give more exposure to all symbols within the alphabet (for the sake of practice), we will, with a 20% chance, fall back to a lower level chain, even when the higher level chain contains a valid transition.

It is also possible for us to progress back up from a lower level chain to a higher level chain. The program keeps track of a history for each level of the hierarchical chain, no matter which chain is actually used to transition to the next state. The next time the hierarchical Markov chain is used, it checks for valid transitions starting from the highest level chain, which allows for getting back up into the higher order chains.

The hierarchical Markov chain history is generated as follows:

- \* For word-level chains when a word is generated:

The new history for the chain is the new word appended to the previous history with the first word removed.

- \* For letter-level chains when a word is generated:

The new history for the chain is last  $n$  letters of the previous history with all letters of the new word appended to it, where  $n$  is the size of the previous history (which is only one state back, no more).

- \* For word-level chains when a letter is generated:

The new history for the chain is the previous history with the new letter appended to the last word of the previous history. If the new letter is a space, then a new blank word is appended to the previous history with the first word removed.

- \* For letter-level chains when a letter is generated:

The new history for the chain is the new letter appended to the previous history with the first letter removed.

After the hierarchical Markov chain is created, we the value of generate-random-qso as a new string composed from the introduction of the QSO, the main body of the QSO, and the conclusion of the QSO.

## Conclusion

Letter substitutions would also help to create a more similar text. For instance, instead of emitting a disallowed letter C, we could allow a K where the C would have been used. The issue with this is that the history and context information would need to be maintained as if a C were actually emitted. This sounds a bit tricky.

We have some structure to the QSO, but we could have done better.

The generated text sounds like somebody schizophrenic with very short term memory loss. They say, "GOOD COPY EDWARD. MY NAME IS VICKY. MY NAME IS LARRY. HOW COPY JOHNNY." messing up names so much. Markov chains are famous for being bad at this sort of thing, so I might have to help it out a bit similar to how I helped with the callsigns. To do this, I could use hierarchical Markov chains not for the entire text, but for chunks, where the chunks are approximately: "the radio I am using", "my name", "you name", "where I live", "what the weather is like", "how well I am copying you", "asking how I am being copied".

I really liked describing my program as a set of production rules. As a C programmer with experience using Bison, it was a breath of fresh air to be have my main source code resemble EBNF as opposed to only the source code consumed by Bison resembling EBNF. It's nice to have the ability to express the higher level ideas in the main language of the program, instead of scattered about in various other languages.

I made an attempt to make the Markov functions as general as possible and not limited to just strings, but for the sake of time I just implemented the hierarchical Markov chain history code to work with lists that contain only strings. Everything else was intended to be general enough to work with heterogeneous lists

I started out writing unit tests for most of my functions, but after I started creating functions that used variables from their parent scope, I found it much more difficult to write tests for everything. I also didn't like having to choose between running the unit tests all the time in my main program, and having to export internal-use-only functions for my test program to use. There must be a better way to unit test that I'm not yet aware of.

## Build Instructions

The program runs interactively with DrRacket, so all that needs to be done is to press the run button in DrRacket.

## Test Instructions

Run any of the \*-test.rkt files in DrRacket to test \*.

## Execution Instructions

Here are some fun things you can do with it. Open "qso-generate.rkt" and run it. Then run some of these commands.

Generate new, 20-word long, text in the style of the included QSO corpus, using a typical word-level Markov-chain (size

2).

```
(display (string-join (generate-similar-corpus 2 (string-split
(file->string
"corpora/qso.txt") " ") 20) " "))
```

Generate new text in the style of three different texts using a typical word-level Markov-chain (size 2).

```
(display (string-join (generate-similar-corpus 2 (string-split
(string-append
(file->string "corpora/navy-seal-copypasta.txt") (file->string
"corpora/dont-copy-that-floppy.txt") (file->string
"corpora/bel-air.txt"))) " ")
200) " ")
```

Generate a line that is similar to any line from a text file, but without using the letter E or e, using hierarchical Markov chains.

```
(display (generate-random-text
(build-hierarchical-markov-chain-from-file "corpora/king-
lear.txt")
'(#A #B #C #D #F #G #H #I #J #K #L #M #N #O #P #Q #R
#S #T
#U #V #W #X #Y #Z #a #b #c #d #f #g #h #i #j #k #l #m #n
#o #p #q #r #s #t #u #v #w #x #y #z #. #, #' #; #? #!
#space)
200
))
```

Aside from all those other fun things, here's the main point of this program. Generate a random QSO-like message with a few differently sized alphabets. The alphabets are in N1IRZ's learning order.

```
(display (generate-random-qso '(#K #M)))
```

```
(display (generate-random-qso '(#K #M #R #S #U #A #P #T
#L #O)))
```

```
(display (generate-random-qso '(#K #M #R #S #U #A #P #T
#L #O #W #I
#. #N #J #E #F # #Y #V)))
```

```
(display (generate-random-qso '(#K #M #R #S #U #A #P #T
#L #O #W #I
#. #N #J #E #F # #Y #V #, #G #5 #/ #Q #9 #Z #H #3 #8)))
```

```
(display (generate-random-qso '(#K #M #R #S #U #A #P #T
#L #O #W #I
#. #N #J #E #F # #Y #V #, #G #5 #/ #Q #9 #Z #H #3 #8 #B
#?
#4 #2 #7 #C #1 #D #6 #X)))
```

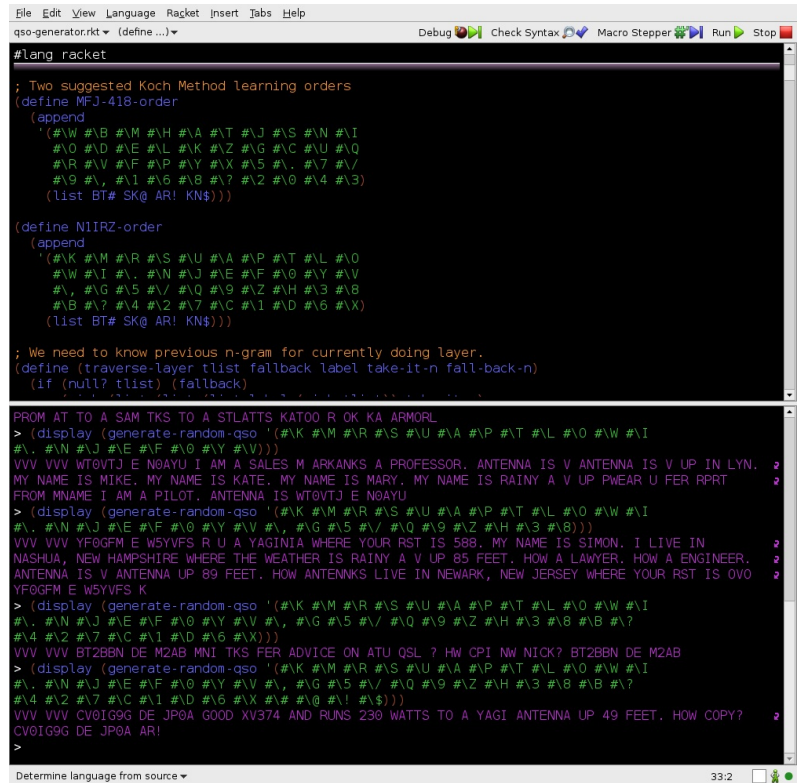
```
(display (generate-random-qso '(#K #M #R #S #U #A #P #T
#L #O #W #I
#. #N #J #E #F # #Y #V #, #G #5 #/ #Q #9 #Z #H #3 #8 #B
#?
#4 #2 #7 #C #1 #D #6 #X ## #@ #! #$$$)))
```

### Describe any bugs or caveats

The random initial transition list, that decided which word to start with, might only be using the first word of the last

example QSO in qso.txt. This is fixed in a newer version of the program, too new to be included in Lisp in Summer Projects.

## Screen shots



```
file Edit View Language Racket Insert Tabs Help
qso-generator.rkt (define ...) Debug Check Syntax Macro Stepper Run Stop
#lang racket

; Two suggested Koch Method learning orders
(define MFJ-418-order
  (append
    (#W #\B #\M #\H #\A #\T #\J #\S #\N #\I
     #\O #\D #\E #\L #\K #\Z #\G #\C #\U #\Q
     #\R #\V #\F #\P #\Y #\X #\5 #\ . #\7 #\ /
     #\9 #\ , #\1 #\6 #\8 #\? #\2 #\0 #\4 #\3)
    (List BT# SK@ AR! KN$)))

(define NIIRZ-order
  (append
    (#K #\M #\R #\S #\U #\A #\P #\T #\L #\O
     #\W #\I #\ . #\N #\J #\E #\F #\0 #\Y #\V
     #\ , #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8
     #\B #\? #\4 #\2 #\7 #\C #\1 #\D #\6 #\X)
    (List BT# SK@ AR! KN$)))

; We need to know previous n-gram for currently doing layer.
(define (traverse-layer tlist fallback label take-it-n fall-back-n)
  (if (null? tlist) (fallback)

      (let ([n (take-it-n)])
        (display (generate-random-qso '(#K #\M #\R #\S #\U #\A #\P #\T #\L #\O #\W #\I
                                       #\ . #\N #\J #\E #\F #\0 #\Y #\V #\ ,
                                       #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8 #\B #\?
                                       #\4 #\2 #\7 #\C #\1 #\D #\6 #\X))))))

PROM AT TO A SAM TKS TO A STLATTS KATOO R OK KA ARMORL
> (display (generate-random-qso '(#K #\M #\R #\S #\U #\A #\P #\T #\L #\O #\W #\I
                                  #\ . #\N #\J #\E #\F #\0 #\Y #\V #\ ,
                                  #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8 #\B #\?
                                  #\4 #\2 #\7 #\C #\1 #\D #\6 #\X))))
VVV VVV WTVOTJ E NOAYU I AM A SALES M ARKANKS A PROFESSOR. ANTENNA IS V ANTENNA IS V UP IN LYN.
MY NAME IS MIKE. MY NAME IS KATE. MY NAME IS MARY. MY NAME IS RAINY A V UP PWEAR U FER RPRT
FROM MNAME I AM A PILOT. ANTENNA IS WTVOTJ E NOAYU
> (display (generate-random-qso '(#K #\M #\R #\S #\U #\A #\P #\T #\L #\O #\W #\I
                                  #\ . #\N #\J #\E #\F #\0 #\Y #\V #\ ,
                                  #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8 #\B #\?
                                  #\4 #\2 #\7 #\C #\1 #\D #\6 #\X))))
VVV VVV YFOGFM E WSYVFS R U A YAGINIA WHERE YOUR RST IS 588. MY NAME IS SIMON. I LIVE IN
NASHUA, NEW HAMPSHIRE WHERE THE WEATHER IS RAINY A V UP 85 FEET. HOW A LAWYER. HOW A ENGINEER.
ANTENNA IS V ANTENNA UP 89 FEET. HOW ANTENNKS LIVE IN NEWARK, NEW JERSEY WHERE YOUR RST IS OVO
YFOGFM E WSYVFS K
> (display (generate-random-qso '(#K #\M #\R #\S #\U #\A #\P #\T #\L #\O #\W #\I
                                  #\ . #\N #\J #\E #\F #\0 #\Y #\V #\ ,
                                  #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8 #\B #\?
                                  #\4 #\2 #\7 #\C #\1 #\D #\6 #\X))))
VVV VVV BT2BBN DE M2AB MNI TKS FER ADVICE ON ATU QSL ? HW CPI NW NICK? BT2BBN DE M2AB
> (display (generate-random-qso '(#K #\M #\R #\S #\U #\A #\P #\T #\L #\O #\W #\I
                                  #\ . #\N #\J #\E #\F #\0 #\Y #\V #\ ,
                                  #\G #\5 #\ / #\Q #\9 #\Z #\H #\3 #\8 #\B #\?
                                  #\4 #\2 #\7 #\C #\1 #\D #\6 #\X #\# #\@ #\! #\$))))
VVV VVV CV0IG9G DE JP0A GOOD XV374 AND RUNS 230 WATTS TO A YAGI ANTENNA UP 49 FEET. HOW COPY?
CV0IG9G DE JP0A AR!
>
```

[qso-generator.png](#)

## Official

I have read rules and have abided by them.  
I am 18 years of age or older.  
I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.