

# 1 Introduction

This language is used to write sheet music for guitar tabs. Maybe in the future it'll be able to write normal sheet music as well!

I've always found GUI music writing software to be annoying to use, and it's always difficult to find all the buttons and features you need. I've felt like it would be easier and faster to simply type out music, since typing something like 'e4' is faster than looking for the right buttons and finding the right placement. This language would also be very customizable, and the user could adjust widths, spacing, etc.

To run the program, first type `dotnet run ;file;`, and then run the `score.ps` postscript file that is created by the `f#` code. You will need to have Ghostscript downloaded, and you can call `gs score.gs` to run the code. `ps2pdf score.gs` will open a pdf version, which is generally more useful.

# 2 Design Principles

I want the language to be as simple and intuitive as possible. The syntax is very simple, and a lot of the formatting is taken care of by the evaluator, which means that there are default values for a lot of parameters, making life easier for the user. Aesthetically, I'm working to make sure the output sheet music follows music engraving standards to look professional.

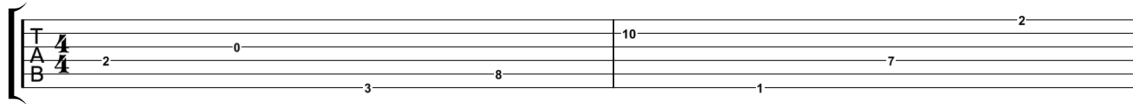
# 3 Example Programs

Example 1:

```
-type tab
-time 4-4
-key c
```

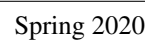
```
1:
    3e4
    4g
    1g
    2f
2:
    5a
    1f
    3a
    6f#
```

Output:

**Example 2:**

```
-type tab  
-time 3-4  
-key c
```

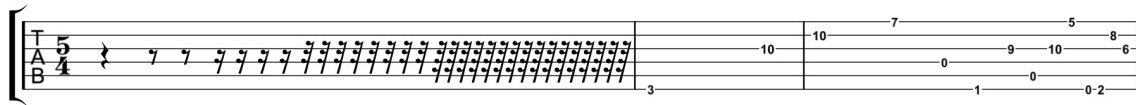
```
1:  
  r  
  6e16  
  1f  
  2g#  
  4ab  
  r8  
  r  
2:  
  r4  
  3fn  
  5g#  
3:  
  5a2  
  3bb4
```



```

r
r
2:
  1g1
  4f4
3:
  5a2
  6b4
  3d8
  1f
  4e16
  2a
  4f32
  6a
  1e64
  1f#
  5gn
  4db

```



## 4 Language Concepts

Data types:

There are several data types useful to the user:

Notes - consist of a string, pitch, rhythm, and possible properties

Measure numbers

Specifiers, such as key and time signature

Grammar:

```

<expr>          ::= <option>
                  | <measure>
<option>        ::= <type>

```

```

    | <time>
    | <key>
<key>      ::= c | cm | c# | c#m | cb | d | dm | db | d#m | e | em | eb | ebm | f |
<time>     ::= <num>-<num>
<num>      ::= x
<type>     ::= tab
<measure>  ::= <note>+
<note>     ::= <simple>
              | <complex>
              | <group>
              | <tuplet>
<simple>    ::= <singlesimple>
              | <restsimple>
<complex>  ::= <singlecomplex>
              | <restcomplex>
<singlesimple> ::= <string><pitch><property>*
<restsimple>   ::= r
<singlecomplex> ::= <string><pitch><rhythm><property>*
<restcomplex>  ::= r<rhythm>
<group>       ::= (<singlesimple>+)
              | (<singlesimple>+)<rhythm>
<tuplet>      ::= t<num>o<num> {<simple>+}
<string>      ::= 1 | 2 | 3 | 4 | 5 | 6
<pitch>       ::= | A | ASharp | AFlat | ANat | B | BSharp | BFlat | BNat | C | CSharp
<rhythm>      ::= <rhythmnumber><dot>*
<rhythmnumber> ::= 1 | 2 | 4 | 8 | 16 | 32 | 64
<dot>         ::= .
<property>    ::= /sls | /sle | /stu | /std | /p | /plu | /pld | /gr | /har | /sl | /s

```

## 5 Syntax

A program begins with optional specifiers as to the type of music being written (only tab supported as of now), time signature, and key. They begin with a dash, followed by the specifier, and then the actual string

e.g.

-type tab

-key f#

-time 12-8

Measures always begin with an int, which is the measure number, followed by a colon

e.g.

4:

Notes start with a string number (1-6), and then a pitch

e.g. 5gb

Pitches can be: a, a#, ab, an, b, b#, bb, bn etc.

A rhythm can be specified as: 1 2 4 8 16 32 64

It can be followed by dots as well

e.g.

32..

If no rhythm is given, it defaults to 4

Properties can be specified to change or add qualities. They begin with a / and then the abbreviation  
e.g.

/gr means grace note

Notes can also be rests, which are an r followed by an optional rhythm

Examples of valid notes:

5f

4g64../gr/sl

r

r4.

This language doesn't really have associativity issues

## 6 Semantics

The table may have floated to another page

Table 1: Data and Operations

Syntax	Abstract Syntax	Type	Meaning
-type tab	ScoreOption of string *	string * string	This is an option type, where it begins with a dash, followed by the specifier and the data
1:	measure number	string	A number followed by a colon is the measure number which must precede each measure
r	RestSimple	Note	An r alone is a simple rest, which uses the default rhythm
5g#	SingleSimple of int * Pitch * Property List	Note	This is a simple note, with no rhythm specified, so it uses the default rhythm, which begins as a quarter note and afterwards is whatever the last rhythm specified was.
4e32/gr	SingleComplex of int * Pitch * Rhythm * PropertyList	Note	This is a complex note, comprised of all the parts that could make up a note
r32	RestComplex of Rhythm	Note	This represents a rest, which is complex since it has a rhythm specified

## 7 Remaining Work

Features that I hope to implement:

Dotted rhythms

Multiple notes at once

Beaming to make rhythms easier to read

Decorations, such as slurs, staccatos, slides etc

Ability to add title, author name, other text