Decoding and playing notes

In the following exercises, we will see how to generate and play some sounds. We will use the tuneR package.

In R, several packages deals with music analysis. tabr, tuneR , seewave, chorrrds are dedicated to sound and music processing. Package tabr is dedicated to music analysis and generation.

Here are the weblink describing functions :
tuneR :   https://cran.r-project.org/web/packages/tuneR/tuneR.pdf

1-   Install and load libraries

Install libraries

```
install.packages( c('tuneR', 'seewave'))
```

Load Library   (tuneR and seewave)

```
library(tuneR)
library(seewave)
```

Define your local path where your music files are located.

```
path="your path to store music files"
```

2- Play a file - use the command **play** of tuneR and play the file *allobates_femoralis.wav*

```
> play(paste0(path, "Allobates_femoralis.wav"))
```

3- Generate a sound

The following command defines the times in seconds if a sample has a duration of 3 seconds and a frequency of 8000 Hz :

```
t = seq(0, 3, 1/8000)
```

440 Hz sine wave that lasts t length seconds (here, 3 seconds) :

```
u = (2^15-1)*sin(2*pi*440*t)
```

Finally we generate the wav variable :

```
w = Wave(u, samp.rate = 8000, bit=16)
```

Look at u and w . How many values? What are the min, max, and mean values?
(tip: use summary function)

Play w.
(tip: use play function, refer to tuneR.pdf)

4- Play a series of notes

Let's consider different parameter and a playing function that generate the wav variable:
The sampling rate used in the playing function:

```
sr = 8000;
```

The length of time (secs) of the smallest musical time unit:

```
tatum= .1;
```

The playing function below (called *playit*) can play a melody. The melody is expressed using a midi pitch for each spaced tatum value :

```
playit = function(f){
    h = NULL;
    bits= 16 # bit depth
    f[f<1]= f[f<1]+ 8
    for (i in 1:length(f)){
           fr = 440*2^((f[i]-69)/12);
            h = c(h,rep(fr,sr*tatum))
      }
      y = sin(2*pi*cumsum(h/sr))
      u = Wave(2^(bits-4)*y,samp.rate= sr, bit=bits)
      tuneR::play(u)
}
```

We will generate a series of 50 notes. With some variation with the previous note. We define 5 parameters and a function that will correct the values of the notes. The array of notes is called *s*. *lo* is the lowest possible midi pitch. *hi* is the highest possible midi pitch. *p* is the probability of keeping the same direction. The variables notes are the number of notes (here 50). *s[1]* is initialized by *lo*. *up=1* is a start with an upward chromatic direction. *1-p* is the probability of switching directions:

```
lo     = 80;
hi     = 100;
p          = .8
notes      = 50
s = rep(0,notes);
s[1]  = lo;
up         = 1

for (i in 2:notes){
            if (up){ s[i]= min(s[i-1]+1,hi);}  # if up move upward if possible
            else{ s[i]= max(lo,s[i-1]-1);}
            if (runif(1)> p) up = 1-up;
}
```

Hence *s* is given as a parameter to function *playit* .
>playit(s)

As variation of notes is generated with *runif* probability distribution. The loop will generate a different series of notes each time you run it.

Modify function playit so that it can save the series of notes in a wav file. Change parameters and save the series of notes in a wav file.

```
playit = function(f, fname){
    h = NULL;    bits= 16 # bit depth
f[f<1]= f[f<1]+ 8
for (i in 1:length(f)){
            fr = 440*2^((f[i]-69)/12);
             h = c(h,rep(fr,sr*tatum))
      }
      y = sin(2*pi*cumsum(h/sr))
      u = Wave(2^(bits-4)*y,samp.rate= sr, bit=bits)
      tuneR::play(u)
     savewave(u, filename=fname)
}
```

5- Find equivalence between American notation of notes and midi values (from 68 to 100)
(You need to fill in the following table.)

Midi value	American note	Midi value	American note	Midi value	American note
68		79		90	
69	A	80		91	
70		81		92	
71		82		93	
72		83		94	
73		84		95	
74		85		96	
75		86		97	
76		87		98	
77		88		99	
78		89		100	

6- play first notes of Beethoven 5th symphony: sol sol sol mi fa fa fa re
find midi values for **sol sol sol mi fa fa fa re**
then call above playit() function.

Hint: You can find out the corresponding American note using the following table, and then use the above table you have finished to find out Midi value for each note.

Note	C	D	E	F	G	A	B	C
Syllable	DO	RE	MI	FA	SOL	LA	SI	DO

Using c() function in R to create a vector of a series of midi values.
For example, we have a list of midi values 88 88 87 93 93 88, you can create a vector notes by using c function

```
>notes =c(88,88,1,88,88,1,87,87,1,93,93,1,93,93,1,88,88)
```

Then, call playit and save it in the test.wav file.

```
>playit(notes, paste0(path,"test.wav"))
```

7- write a function you call *transcribeMusic* that will convert the sound file into a sequence of notes
(we will using periodogram and FF (fast fourier) functions from tuneR, we will settle width of the periodogram to 4096)

```
transcribeMusic <- function(wavFile, widthSample = 4096) {
   # add your code below

   # use periodogram function to obtain time series info from wave file
   # the output is an object of class Wspec, for example, myWspec
   myWspec =

   #use FF function to estimate fundamental frequencies from myWspec, to get freqWav
    freqWav =

   # use noteFromFF function to derive notes from given frequencies in freqWav, to get noteNum

   noteNum =

   # get rid of null value from notes in the above object noteNum, to get noteNum2 without nulls
   noteNum2 =

   #print note numbers of noteNum2
   print(noteNum2)

   # use notenames to generate note names (noteTxt) from noteNum2

   # return noteTxt

}
```

8- Using *transcribeMusic* guess the sequence of notes of the bird (file Allobates_femoralis.wav). Save the series of notes in a wav file.

```
originalSound <- readWave( paste0(path, "Allobates_femoralis.wav") )
noteTxt = transcribeMusic(originalSound)
print(notesTxt)
```

```
playit(notes , paste0(path,"bird.wav"))
```

9- Read the music files with mp3 format *champagne.mp3* and *Vonstroke.mp3*

Convert the files in wav format
(hint: use readMp3 function to read mp3 file, use writeWave function to save to wav file format)

```
wmp1 = readMP3( paste0(path, "champagne.mp3") )
wmp2 = readMP3( paste0(path, "electro.mp3") )
writeWave(wmp1, filename = paste0(path, "champagne.wav") )
writeWave(wmp2, filename = paste0(path, " Vonstroke.wav") )
```

Concatenate the two music in a single one, and export it as a wav file.
(hint: use bind function to concatenate two music objects)

```
w2 = bind( wmp1 , wmp2 )
writeWave(w2, filename = paste0(path, "two.wav"))
```

Concatenate by range time alternatively first 3 second range time of one followed by 3 seconds of the second and so one (6 samples or more).
(Hint: use extractWave function to extract a range of time of music from a wav file, use *bind* function to concatenate several music objects).

```
wav1 = readWave( paste0(path, "champagne.wav") )
wav2 = readWave( paste0(path, "Vonstroke.wav") )

# extract from 0 to 3 seconds music
ch1 = extractWave(wav1, from = 0, to = 3, xunit = "time")
ch2 = extractWave(wav1, from = 3, to = 6, xunit = "time")
ch3 = extractWave(wav1, from = 6, to = 9, xunit = "time")

vo1 =extractWave(wav2, from = 0, to = 3, xunit = "time")
vo2 =extractWave(wav2, from = 3, to = 6, xunit = "time")
vo3 = extractWave(wav2, from = 6, to = 9, xunit = "time")

combi = bind( ch1 , vo1, ch2, vo2, ch3, vo3)
```

10- To stack two pieces of music we use arithmetic of tuneR method '+'

Load champagne.mp" and electro.mp3, convert to wav and select the same quantity of samples for addition.

```
Wav1 = readMP3( paste0(path, "champagne.mp3") )
Wav2 = readMP3( paste0(path, "electro.mp3") )
writeWave(Wav1, filename = paste0(path, "champagne.wav") )
writeWave(Wav2, filename = paste0(path, "electro.wav") )
```

```
Wav_ch = readWave( paste0(path, "champagne.wav") )
Wav_el = readWave( paste0(path, "electro.wav") )
Wav_ch1 = extractWave(Wav_ch, from = 1, to = 500000)
Wav_ch2 = extractWave(Wav_el, from = 1, to = 500000)

# www = Wav_ch1+Wav_ch2 # if both are stereo

www = Wave(Wav_ch1@left+Wav_ch2@left, samp.rate= 44100, bit=16) #if
one is mono

tuneR::play(normalize(www, unit=c("16")))
```

Now we want to create something more harmonious

Try to mix music bass-lounge.mp3 and satie_gymnopedie_no_1.wav.
For the second you will choose samples from 400001 to 1400000 from satie_gymnopedie_no_1.mp3 file. And export the wav to a file called: satie-lounge.wav.

Here is the partial code

```
 wav_sa_mp = readMP3( paste0(path, "satie_gymnopedie_no_1.mp3") )
wav_lo_mp = readMP3( paste0(path, "bass-lounge.mp3") )
writeWave(wav_sa_mp, filename = paste0(path, "satie_gymnopedie_no_1.wav") )
writeWave(wav_lo_mp, filename = paste0(path, "bass-lounge.wav") )

wav_sa = readWave( paste0(path, "satie_gymnopedie_no_1.wav") )
wav_lo = readWave( paste0(path, "bass-lounge.wav") )
```

You have to extract partial wave from the wav object using extractWave function, and save the wave to a file called satie-lounge.wav by using writeWave function.
Then, use play function to play it.

11- choose two music of your preference and make a mix as in question 9/10