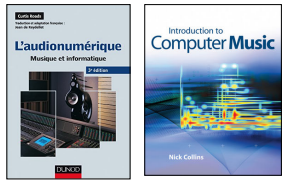
Université Nice Sophia Antipolis

Programmer avec des sons

(Introduction à l'**audio numérique**)

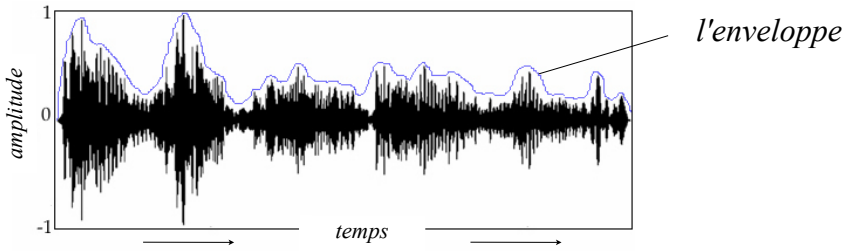


- Partie 1 -

Généralités sur l'audio

Le son comme fonction du temps

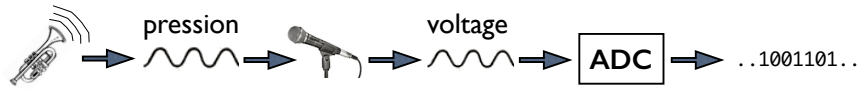
- Un **son** (*sound*) est représenté par une fonction $p=p(t)$ où l'abscisse t est le **temps** et l'ordonnée la **pression d'air** (ou **amplitude** du signal) qui va s'exercer sur le tympan (ou la membrane du microphone).
- La pression peut être positive (compression) ou négative (dépression) suivant le mouvement des particules d'air. La pression nulle correspond au silence.



- Le temps étant discrétisé, la figure précédente représente un grand nombre d'**échantillons** (*samples*) du son à intervalles réguliers. Un échantillon isolé n'indique rien sur le son.

De l'Analogique au ~~Digital~~ Numérique

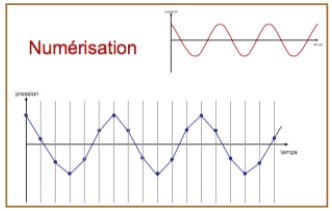
- Le monde **analogique** est **continu**, ses courbes mathématiques sont lisses. Le monde **numérique** est **discret**, une courbe est remplacée par une suite de points, les échantillons (*samples*).



- Le **microphone** convertit grâce à une membrane toute variation de pression d'air en un signal électrique continu : un voltage v = v(t).
- Le **convertisseur** ADC va ensuite **discrétiser** la courbe du voltage.

- Le résultat sera donc un grand nombre d'**échantillons** prélevés sur la courbe du voltage, à une certaine **fréquence d'échantillonnage** F_e, le plus souvent F_e = 44100 Hz (1 Hz = 1/sec). C'est la norme **qualité CD**.

• Le **son numérisé** est une suite de nombres donnant l'amplitude du signal sonore. Ces nombres sont sur 8, **16**, 24 bits ou plus suivant la précision voulue.



• **Occupation mémoire** :

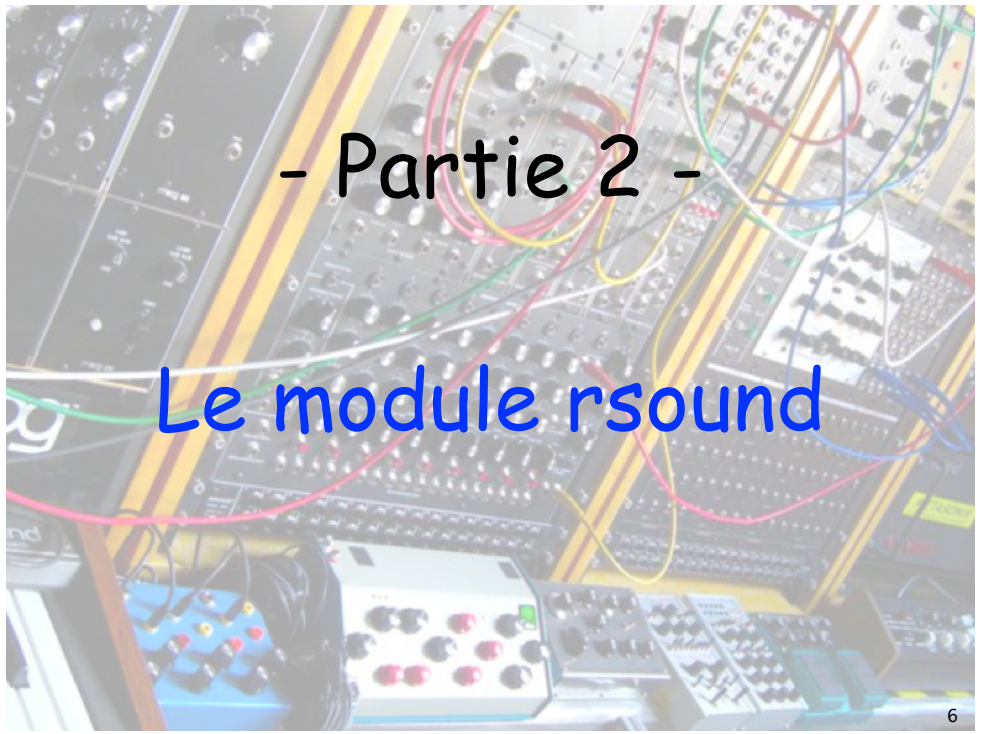
mem = nb-sec x nb-octets x nb-canaux x F_e

- 10 sec de parole humaine, 11 kHz, 8 bits, mono : 110 Ko
- 1 sec de musique CD, 44.1 kHz, 16 bits, mono : 88 Ko
- *Skyfall* (4'50), 44.1 kHz, 16 bits, stéréo : 49 Mo

• Plus la fréquence maximale F_{max} du signal à échantillonner est elevée, plus F_e doit être élevée, pour ne pas perdre trop d'information.

• Le **théorème de Nyquist** précise qu'il faut choisir $F_e \geq 2F_{max}$ pour reconstruire le signal.

ce qui explique le choix de 44100 pour la qualité CD…



- Partie 2 -

Le module rsound

Programmer avec des sons en Scheme

• Nous utiliserons le module expérimental **rsound** de John Clements de l'équipe Racket.

```
(require rsound)
```

• Le module **rsound** va nous permettre de charger en mémoire des fichiers **.wav 16 bits PCM** sous la forme d'objets sonores. Nous pourrons construire ou transformer des objets sonores, puis les sauver sur disque au format **.wav 16 bits PCM**.

• Le format audio **.wav 16 bits PCM** est une format sans perte, contrairement au MP3 qui perd du signal en le compactant. Les professionnels travaillent en studio sans perte avec du 48 bits ou plus…

• Le logiciel **Audacity** (gratuit) permet entre autres de convertir des fichiers pour les avoir au bon format wav PCM 16 bits (cf TP).

• Nous opterons pour **44.1 kHz** comme fréquence d'échantillonnage (la qualité CD). Vérifiez que votre carte audio Windows est bien réglée pour échantillonner à 44.1 kHz (44100 échantillons par seconde).

Quelques sons de percussion prédéfinis de rsound

snare

bassdrum

ding

crash-cymbal

c-hi-hat-1

```
; essai-rsound.rkt
(require rsound)
(play crash-cymbal)
(sleep 1)   ; 1 sec
(play ding)
```

Transformation .wav *(16 bits PCM)* → rsound

- Une valeur Racket de type rsound est une **structure** :

```
(define-struct rsound (data start stop sample-rate))
```

un vecteur contenant les échantillons — *numéro d'échantillon de début et de fin* — *fréquence d'échantillonage (44100)*

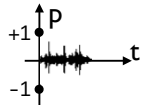
```
> (define foo (rs-read "5th.wav"))
> foo                          ; un son est une struct
#(struct:rsound #<s16vector> 0 1043006 44100)
> (play foo)                   ; play est asynchrone !
> (rsound-sample-rate foo)     ; fréquence Fe
44100
> (rs-frames foo)              ; nombre d'échantillons
1043006
> (duration foo)               ; durée en secondes, cf TP
23.650929705215418
> (stop)                       ; couper l'audio !
```

- Erreur *I give up !* par **rs-read** si le fichier n'est pas au bon format…

Visualisation d'un son

- On peut visualiser la courbe d'amplitude (volume) pour chaque canal :

```
(require rsound/draw)
(rs-draw foo)
```



- Accès à l'amplitude d'un échantillon entre -1 et 1 en coût $O(1)$:

```
> (rs-ith/right foo 100000)    ; l'échantillon
0.12253181554612873            ; n°100000 du canal droit
```

Bof…

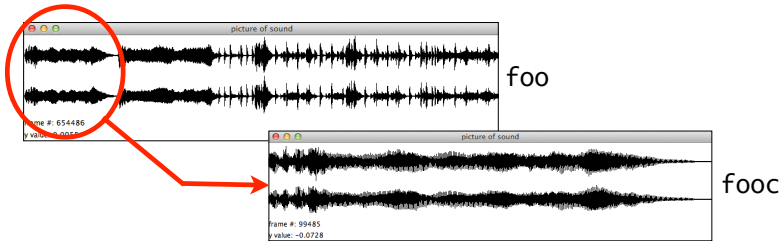
- En passant la souris sur le graphique du son, on visualise le numéro approximatif de l'échantillon à un certain endroit du son (pour faire des clips par exemple).

Extraction d'un clip audio

`(clip snd start end)`

- On peut extraire un morceau (*clip*) d'un son, en donnant le numéro d'échantillon (*frame*) de début et de fin du clip :

```
> foo
#(struct:rsound #<s16vector> 0 1043006 44100)
> (define fooc (clip foo 3000 205000))
> fooc
#(struct:rsound #<s16vector> 3000 205000 44100)
> (rs-draw fooc)
```



N.B. La partie data de foo et de fooc est un unique objet en mémoire. Sinon la mémoire serait vite pleine !

Sauvegarde d'un son sur disque

`(rs-write snd wav-file)`

- La fonction rs-read permettait de charger sous forme de son un fichier au format .wav 16 bits PCM stéréo.
- La fonction rs-write permet inversement de sauver sur disque un objet rsound construit en Scheme.

```
(rs-write fooc "fooc.wav")
```

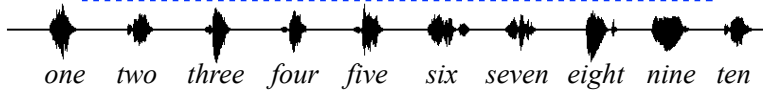
- L'objet de type rsound peut provenir du clip d'un son déjà existant, ou bien d'une suite d'opérations sur plusieurs sons, voire d'un son purement synthétisé de manière mathématique ou à partir du **bruit blanc** aléatoire (noise nb-frames).

Enregistrement d'un son

`(record-sound nbframes)`

```
(define voice (record-sound (* 44100 10)))
(play voice)
```

← *10 sec*



• Intéressons-nous maintenant aux **OPERATIONS SUR LES SONS**.

• Comme avec les images, nous allons juxtaposer, superposer, agrandir… bref **combiner des sons pour obtenir de nouveaux sons**.

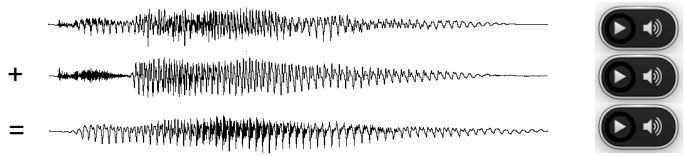
(*opération* son_1 son_2) ☞ son_3

Superposition de plusieurs sons

(rs-overlay snd_1 snd_2)

• ATTENTION : les amplitudes s'ajoutant, risquent de sortir de [-1,+1] !

```
> (define caco (rs-overlay* (list one two three)))
> (play caco)
```



• On utilise rs-overlay* pour superposer une liste de sons :

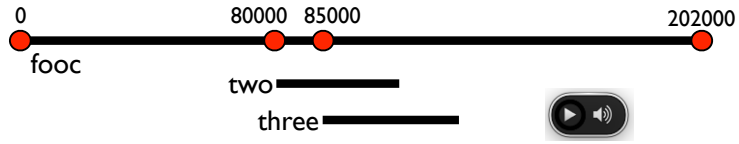
(rs-overlay one two) ⇔ (rs-overlay* (list one two))

Assemblage d'une liste de sons

(assemble L)

• Variante de rs-overlay dans laquelle **les sons peuvent se chevaucher**. Il suffit de préciser à quel moment (numéro de *frame*) installer un son.

```
(define caco1
   (assemble (list (list fooc 0) (list two 80000) (list three 85000))))
(play caco1)
```



Amplification d'un son

(rs-scale k snd)

```
(play (rs-append one (rs-scale 2 one)))
```

ATTENTION AUX OREILLES !

```
> (rs-equal? (rs-scale 2 one) (rs-overlay one one))
#t
```


Le streaming avec les pstreams

• L'inconvénient de play est de demander au système de bas niveau (*portaudio*) d'ouvrir un **flot sonore** (*stream*) pour chaque son. Pour y remédier, rsound propose d'ouvrir une **pstream** (un tube sonore) dans laquelle on peut **insérer un son à un moment donné (passé, présent ou futur)**. Avec *moment* = nombre de frames (44100 frames/sec).

```
(define ps (make-pstream))      ; au temps 0 de ps
```

• On peut **injecter un son dans la pstream** pour lecture immédiate (superposée aux sons en train d'être joués). Par exemple juste au moment où une balle rebondit sur le sol !

```
(pstream-play ps ding)          ; tout de suite !
```

• Il est possible de connaître à un moment donné **l'heure dans la pstream** (en nombre de frames) :

```
> (pstream-current-frame ps)
951389
```

on en est à un peu plus de 25 sec depuis la construction de la pstream…

• La fonction sans résultat pstream-queue permet d'injecter dans la pstream un son destiné à être joué à un moment donné (depuis t=0) :

```
(pstream-queue ps ding 88200)   ; au temps t = 2 sec
```

un numéro de frame

```
(pstream-queue ps ding
  (+ (pstream-current-frame ps) 88200)) ; dans 2 sec
```

```
(pstream-queue ps ding 2000)            ; à t = 2000 frames
(pstream-queue ps crash-cymbal 100000)  ; à t = 100000 frames
(pstream-queue ps ding 50000)           ; à t = 50000 frames
(printf "ps time is ~a\n" (pstream-current-frame ps))
```

• Plus compliquée, la fonction pstream-queue-callback permet d'activer une fonction d'arité 0 à une heure donnée de la pstream :

```
(pstream-queue-callback ps
  (lambda () (stop))                ; fermer l'audio
  441000)                           ; à t = 10 sec
```

• On peut utiliser pstream-queue-callback pour jouer un son en boucle par exemple… Cf TP.

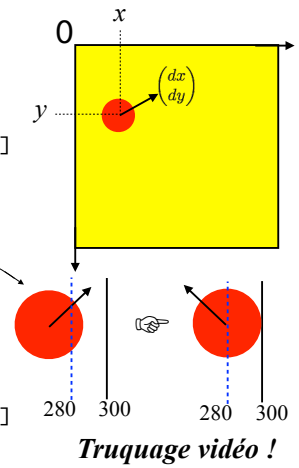
- Partie 3 -

Application aux Animations

Une petite animation…

• Prenons l'exemple d'une balle qui se déplace dans un carré à vitesse constante. Elle rebondit sur les murs. Ni frottement ni pesanteur !

```
(define (anim-balle)
  (local [(define FOND (rectangle 300 300 'solid "yellow"))
          (define BALLE (circle 20 'solid "red"))
          (define-struct balle (x y dx dy))
          ; le Monde est une balle b
          (define INIT (make-balle 150 150 -5 -6))
          (define (suivant b)    ; Monde → Monde
            (local [(match-define (balle x y dx dy) b)
                    (define xs (+ x dx)) (define ys (+ y dy))]
              (cond ((< xs 20)   ; rebond sur mur gauche
                     (make-balle 20 y (- dx) dy))
                    ((> xs 280)  ; rebond sur mur droit
                     (make-balle 280 y (- dx) dy))
                    ((< ys 20)   ; rebond sur plafond
                     (make-balle x 20 dx (- dy)))
                    ((> ys 280)  ; rebond sur plancher
                     (make-balle x 280 dx (- dy)))
                    (else (make-balle xs ys dx dy)))))
          (define (dessiner b)   ; Monde → Scène
            (place-image BALLE (balle-x b) (balle-y b) FOND))]
    (big-bang INIT
              (on-tick suivant)
              (on-draw dessiner))))
```



Truquage vidéo !

… et un exemple de **sonorisation**

• Nous voulons jouer le petit son boing.wav trouvé sur le Web (wavsource ou freesound) à chaque collision de la balle avec un mur.

• Exemple lorsque xs > 280 : collision mur droit. Il nous faudra rendre le résultat de l'expression (make-balle 20 y (- dx) dy) mais **quasi en même temps** jouer le son boing. Nous allons **sonifier** cette expression :

```
(define (sonifier expr)
  (both (pstream-play ps boing) expr))
```

• La fonction both va évaluer ses deux arguments de gauche à droite et retourner la valeur du second en jouant sur le fait que pstream-play est **asynchrone** !

```
(define (both expr1 expr2)    ; à la fois…
  expr2)
```

• Ainsi both réalise le **séquencement** d'évaluations (noté `e1 ; e2` dans les langages impératifs). La véritable primitive Scheme se nomme begin.

https://fr.wikipedia.org/wiki/Sonification


```
(define (anim-balle)
  (local [(define FOND (rectangle 300 300 'solid "yellow"))
          (define BALLE (circle 20 'solid "red"))
          (define-struct balle (x y dx dy))
          ; le Monde est une balle
          (define INIT (make-balle 150 150 -5 -6))
          (define (suivant b)    ; Monde → Monde
            (local [(match-define (balle x y dx dy) b)
                    (define xs (+ x dx)) (define ys (+ y dy))]
              (cond ((< xs 20)   ; rebond sur mur gauche
                     (sonifier (make-balle 20 y (- dx) dy)))
                    ((> xs 280)  ; rebond sur mur droit
                     (sonifier (make-balle 280 y (- dx) dy)))
                    ((< ys 20)   ; rebond sur plafond
                     (sonifier(make-balle x 20 dx (- dy))))
                    ((> ys 280)  ; rebond sur plancher
                     (sonifier (make-balle x 280 dx (- dy))))
                    (else (make-balle xs ys dx dy)))))
          (define (dessiner b)   ; Monde → Scène
            (place-image BALLE (balle-x b) (balle-y b) FOND))]
    (big-bang INIT
              (on-tick suivant)
              (on-draw dessiner))))
```

- Partie 4 -

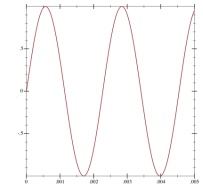
Compléments optionnels...

Les sons purs (sinusoïdaux)

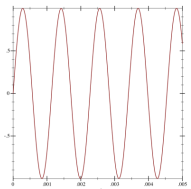
• A la base des sons (signaux audio) se trouvent les **sons purs**, produits par des **oscillateurs sinusoïdaux** : $p = sin(2\pi\omega t)$ de fréquence ω. Un tel signal est **périodique** de période $T = 1/\omega$.

• La **fréquence** mesure la **hauteur** du son (grave ou aigu). Le signal $sin(2\pi\ 440\ t)$ a une fréquence de 440Hz, c'est **le "LA" fondamental** des musiciens. La fréquence est mesurée en **Hertz** (Hz = 1/sec).

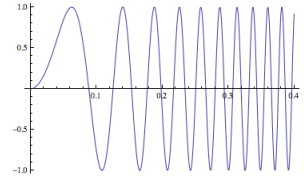
• La plage de fréquences perceptibles par **l'oreille humaine** va du grave 20 Hz à l'aigu 20000 Hz (20 kHz). La **voix** humaine va de 80 Hz à 240 Hz (homme) ou de 140 Hz à 500 Hz (femme).



freq = 440



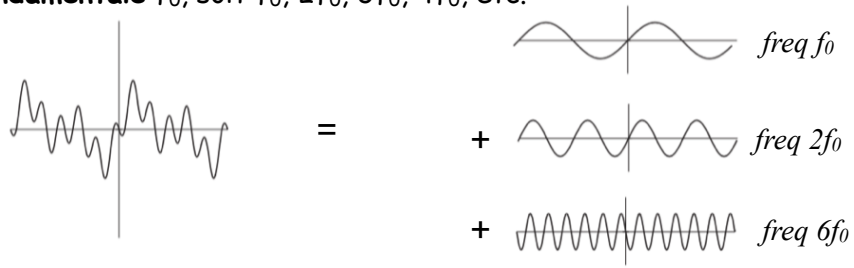
freq = 880



(sin (* 400 t t)))

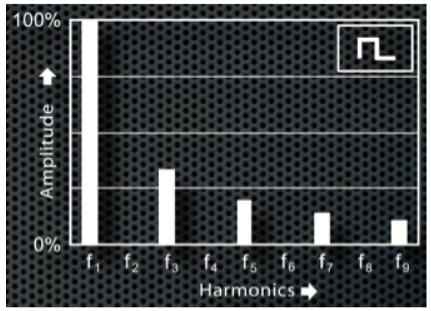
Les sons presque purs (avec harmoniques)

• **Le son fondamental est le son sinusoïdal**. A partir de lui, le grand mathématicien **Joseph Fourier** a pu décomposer (1807) tout signal périodique en une somme (infinie ?) de signaux sinusoïdaux dont les fréquences se trouvent parmi les multiples entiers d'une fréquence **fondamentale** f_0, soit f_0, $2f_0$, $3f_0$, $4f_0$, etc.



• Supposons que f_0 = 440 Hz (le "LA" fondamental). L'ajout des harmoniques 880 Hz, 1320 Hz, 1760 Hz, etc. (ou seulement certaines d'entre elles) produira un "LA" sur un autre **timbre**.

• Exemple : le **signal carré** *square(t)* qui est périodique, doit suivant Fourier être décomposable en somme d'oscillateurs. En maths, c'est l'**Analyse de Fourier** (étudiée en L2-L3).



$$\text{square}(t) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots} \frac{1}{n} \sin(n f_0 t)$$

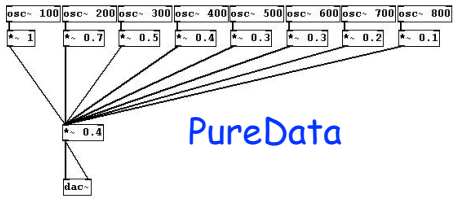
• Le signal en **dent de scie** *(sawtooth)* comporte toutes les harmoniques : le son est **plus riche**...

• Tous ces signaux sont utilisés dans les **synthétiseurs** utilisés sur scène pour modifier des timbres d'instruments.

http://pages.uoregon.edu/emi/11.php

..et tous les autres sons (signaux audio)

- Les sons généraux ne sont pas périodiques. Ils sont obtenus par mélange de sources diverses : instruments de musique, bruits de percussions, voix humaine, hurlement du vent, bruit de moteur, etc.
- L'ingénieur du son utilise des logiciels, des appareils mais aussi des **langages de programmation spécialisés** (*SuperCollider*, *Common Music*, *PureData*, etc) pour construire de nouveaux sons, de la musique algorithmique, et les effets spéciaux du cinéma par ex.

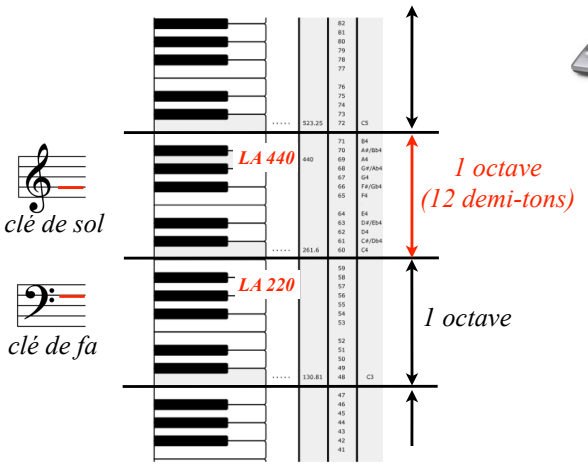


PureData

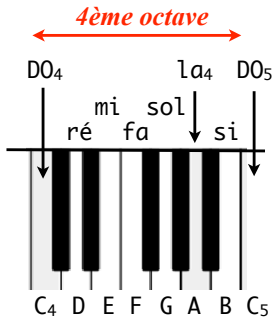
```
// par défaut 440 Hz, amplitude  1.0
{Saw.ar}.play

//volume abaissé à 0.2
{SinOsc.ar(mul: 0.2)}.play
```

SuperCollider

L'échelle musicale occidentale



fréquence multipliée par 2



- L'**échelle musicale occidentale** divise un octave en **12 intervalles** égaux correspondant à 12 notes au clavier . Le rapport entre deux intervalles consécutifs est donc $\sqrt[12]{2} = 2^{1/12}$

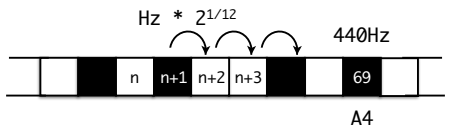
Le système de notation musicale MIDI

- Les notes de musique sont numérotées de 0 à 127.

FR	DO3	DO#	RE	MI ♭	MI	FA	FA#	SOL	SOL#	LA3	SI ♭	SI
USA	C4	C#	D	D#	E	F	F#	G	G#	A4	A#	B
Hz	261.6	277.2	293.7	311.1	329.6	349.2	370	392	412.5	440	466.2	493.9
MIDI	60	61	62	63	64	65	66	67	68	69	70	71

- Un **clavier MIDI** envoie à l'ordinateur une suite d'informations MIDI (le numéro de la note, la pression sur la touche, etc). Pour transiter entre un numéro de note MIDI et une fréquence (*pitch*) :

```
(midi-note-num->pitch 69) -> 440    (Hz)
(pitch->midi-note-num 440) -> 69    (MIDI)
```



Comment jouer des notes de piano ?

```
(require rsound)
(require rsound/piano-tones)

(play (piano-tone 60))
(sleep 3)
(play (assemble                                ; Beethoven 5th
        (list (list (piano-tone 67) 0)         ; G  ==  SOL
              (list (piano-tone 67) 10000)     ; G  ==  SOL
              (list (piano-tone 67) 20000)     ; G  ==  SOL
              (list (piano-tone 63) 30000))))  ; Eb  ==  MI-bémol
(sleep 3)
(play (assemble                                ; accord DO majeur
        (list (list (piano-tone 60) 0)         ; C == DO
              (list (piano-tone 64) 0)         ; E == MI
              (list (piano-tone 67) 0))))      ; G == SOL
```

- Trouver des algorithmes générant des suites de notes agréables à l'oreille n'a rien d'évident. Il s'agit de **composition algorithmique** (ex: utilisation de *chaînes de Markov*)... L'aléatoire est-il beau ?

L'exemple de Clements à RacketCon 2014

- Notes de piano avec chevauchement, tous les 1/10 sec.

```
(require rsound)
(require rsound/draw)

(define LS                          ; une liste de notes à assembler
  (build-list 100 (λ (i) (list (piano-tone (+ 25 (random 30)))
                               (*  i 4410)))))   ; 4410 = 0.1 sec

(define SND (assemble LS))          ; le son obtenu
(rs-draw SND)
```


```
(define SNDC (clip SND 0 819200))
(rs-write SNDC "racketcon.wav")
(play SNDC)
```

Google : "Youtube Clements RacketCon 2014"