introduction à la programmation (avec Processing)

introduction à la programmation (avec Processing)

Erg (École de Recherche Graphique) - Bruxelles.
Arts numériques Bach 1, 2, & 3.

Professeur: Marc Wathieu.

Page de téléchargement de ce fichier :
http://www.multimedialab.be/cours/logiciels/processing.htm

Mise à jour: 15 février 2008.

Ce livret PDF a été conçu comme un diaporama destiné à être projeté et commenté.
Pour un affichage optimisé, je vous recommande une résolution 1024 X 768,
une visualisation avec Acrobat Reader
et le raccourci ctrl+l (Windows) ou pomme+l (Mac OSX).

Télécharger ici Acrobat Reader.

---

http://www.multimedialab.be

Introduction à la programmation (avec Processing)

Sommaire.

.algorithme

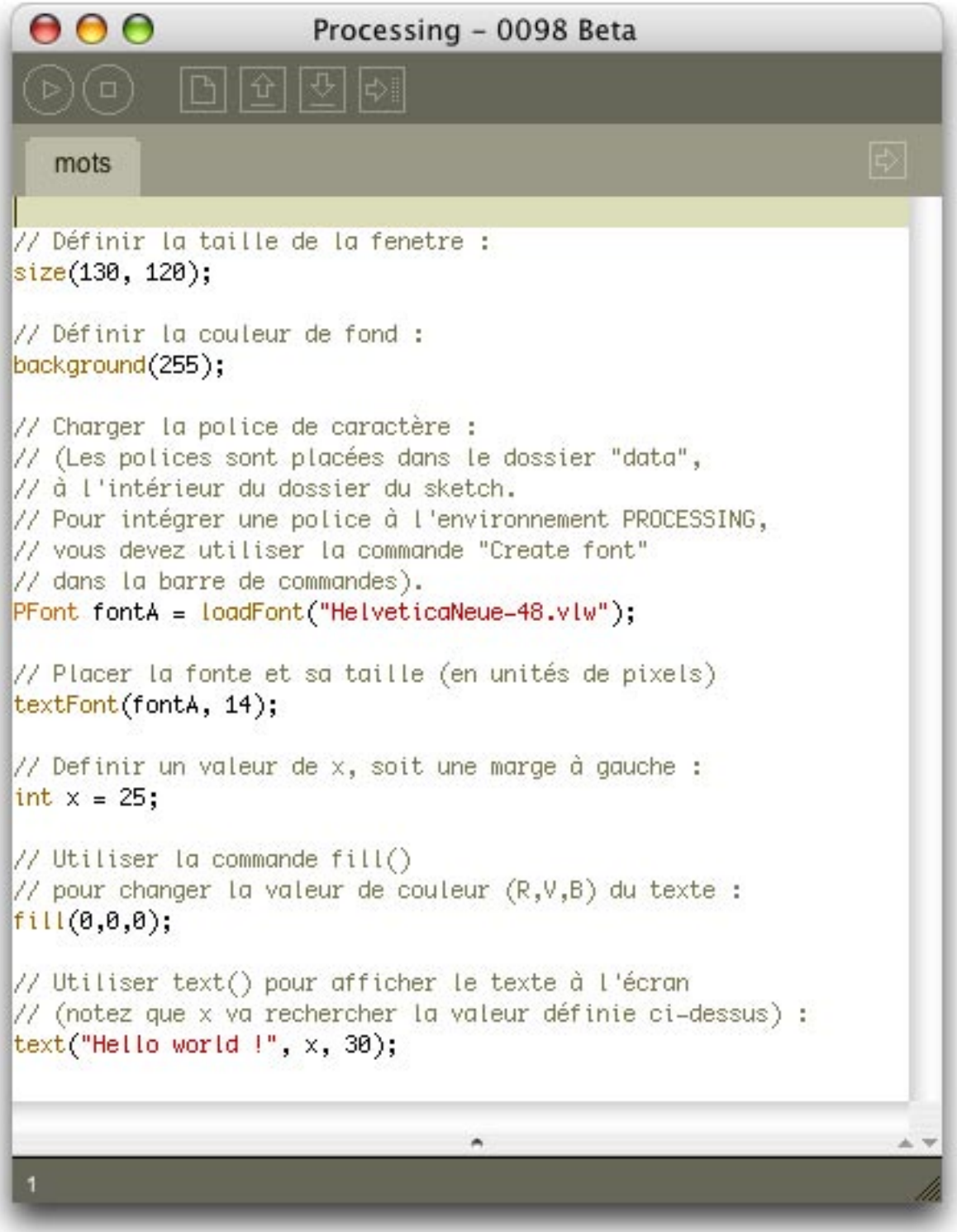
L’ordinateur est un **outil**,
un **exécutant** pointilleux,
méticuleux,
très rapide,
mais totalement dénué d’imagination et d’initiative.

La machine ne fait rien par elle-même;
elle ne fait qu’**appliquer** à la lettre des ordres simples et précis.

L’ordinateur reçoit des **instructions**
énoncées clairement,
sans équivoque,
dans un langage strict
qui qualifie l’action souhaitée.

Il **exécute** ces instructions
séquentiellement,
sans erreur,
sans rechigner,
et sans intelligence.

Une suite de telles instructions est un **programme**.



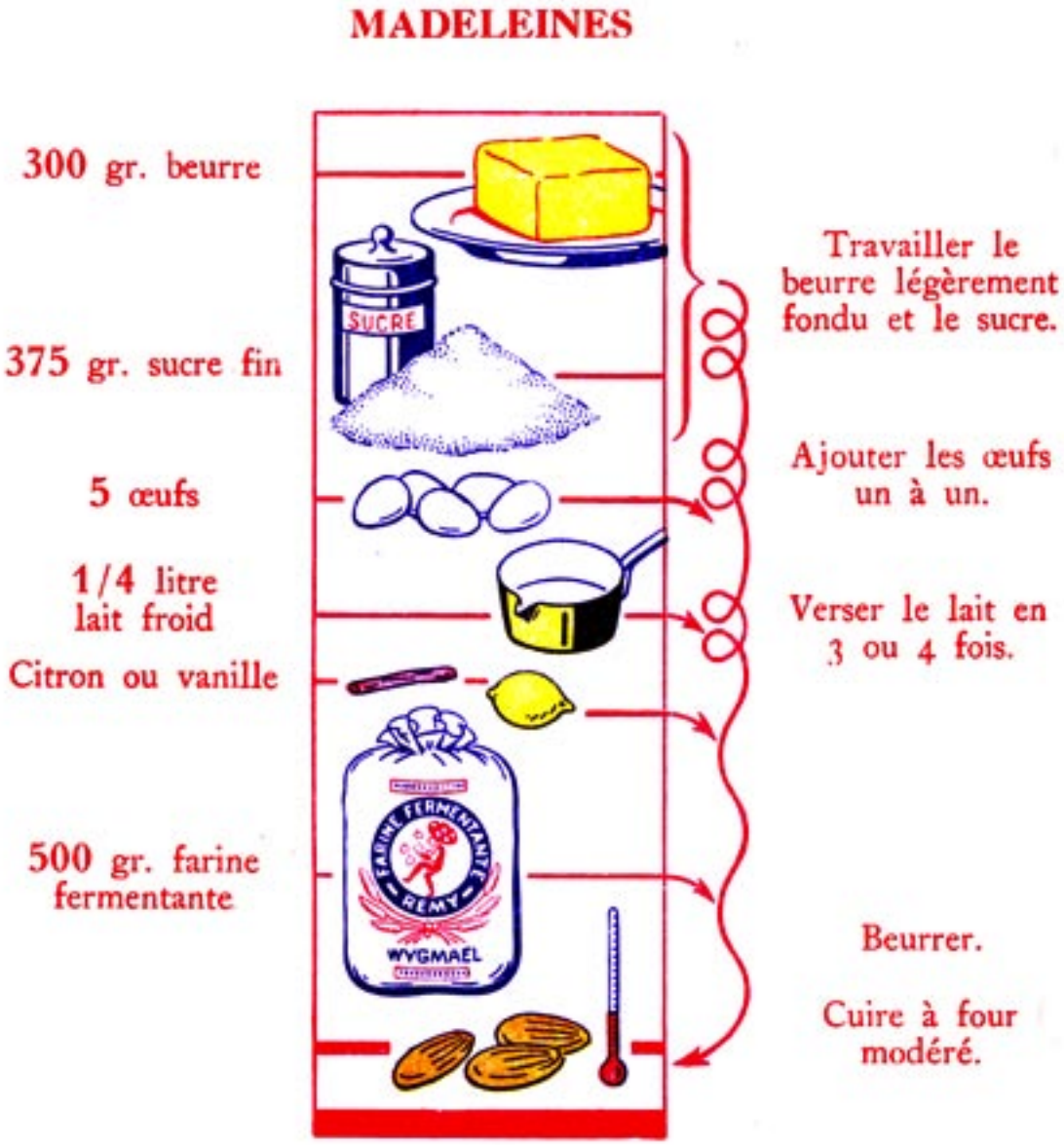
http://processing.org/


Hello
Hello world !

La **programmation** consiste à déterminer la démarche permettant d'obtenir, à l'aide d'un ordinateur, la solution à un problème donné.

Cette marche à suivre s'appelle un **algorithme**.

Une recette de cuisine est un algorithme.



L’explication d’un chemin est un algorithme.

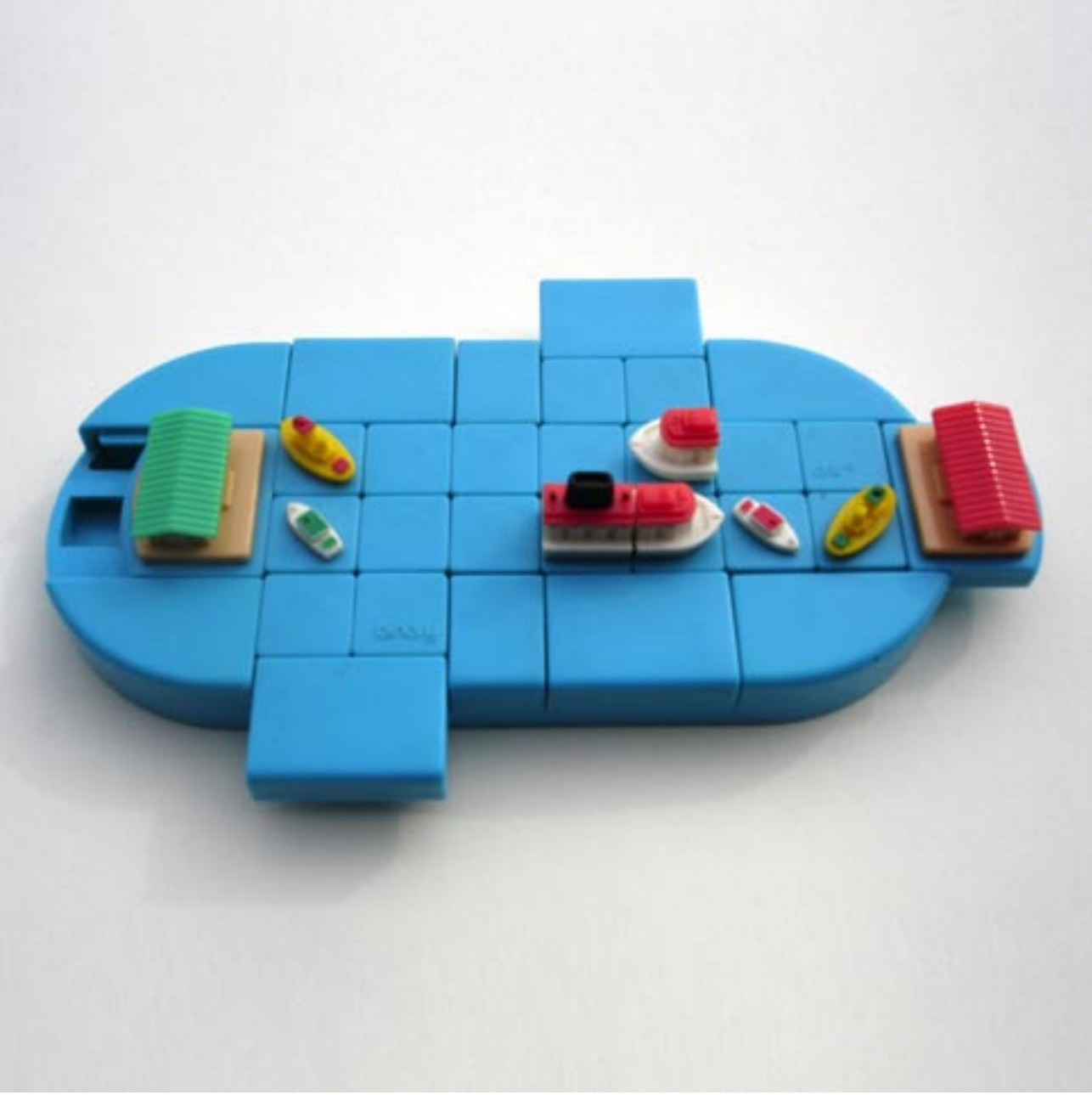
Cumul	Temps		Feuille de route
0 m	0H00		**87 Rue du Page Bruxelles (Belgique)**
			Continuer sur **Rue du Page [27m]**
27 m			Prendre à gauche **Rue Fourmois [120m]**
150 m	0H02		Prendre à droite **Rue du Tabellion [240m]**
390 m	0H06		**10 Rue du Tabellion Bruxelles (Belgique)**

Un mode d’emploi est un algorithme.

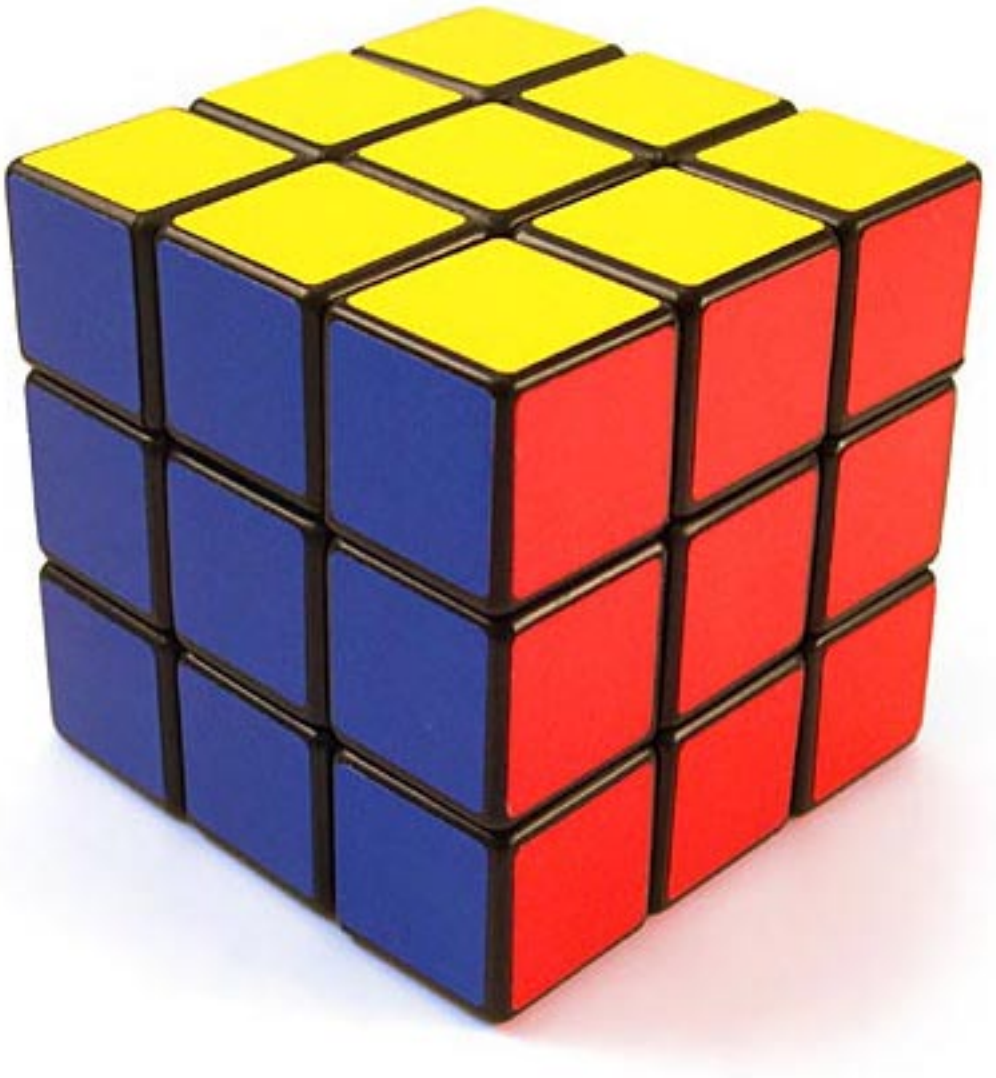
Vous avez encore perdu votre bouton de col : la colère vous fait lever les bras au ciel. Votre poing (**A**) heurte la poire (**B**), projetant sur le ménate (**C**) un jet d'eau pointu. Momentanément aveuglé, l'oiseau (**D**) quitte son perchoir (**E**), tombe dans le chariot de montagnes russes (**F**) qui glisse le long du rail (**G**), tendant la corde (**H**), laquelle active le levier (**I**). La main en bois (**J**) appuie sur la poupée qui parle (**K**). Celle-ci couine : «PLAY BALL!». Le lanceur liliputien de l'équipe des géants (**L**) attrape la balle (**M**) qui est collée au bras du phono (**N**), le mettant en marche. Le disque dit «Où qu'c'est-y qu'il a passé ?». Le père du lanceur (**O**), un penseur encore plus petit que son fils, est intrigué par la question, et marche de long en large pour y réfléchir. Absorbé dans sa réflexion, il passe sous le bureau (**P**), se cogne au bouton de col (**Q**) et crie «Ouille!», vous mettant ainsi sur la trace.

Les cartoons de Rube Goldberg sont de véritables algorithmes.
http://www.rubegoldberg.com/

La résolution d’un «Fifteen Puzzle» est algorithmique.



La reconstitution du bateau d’un côté à l’autre d’un «Port-to-Port» est algorithmique.

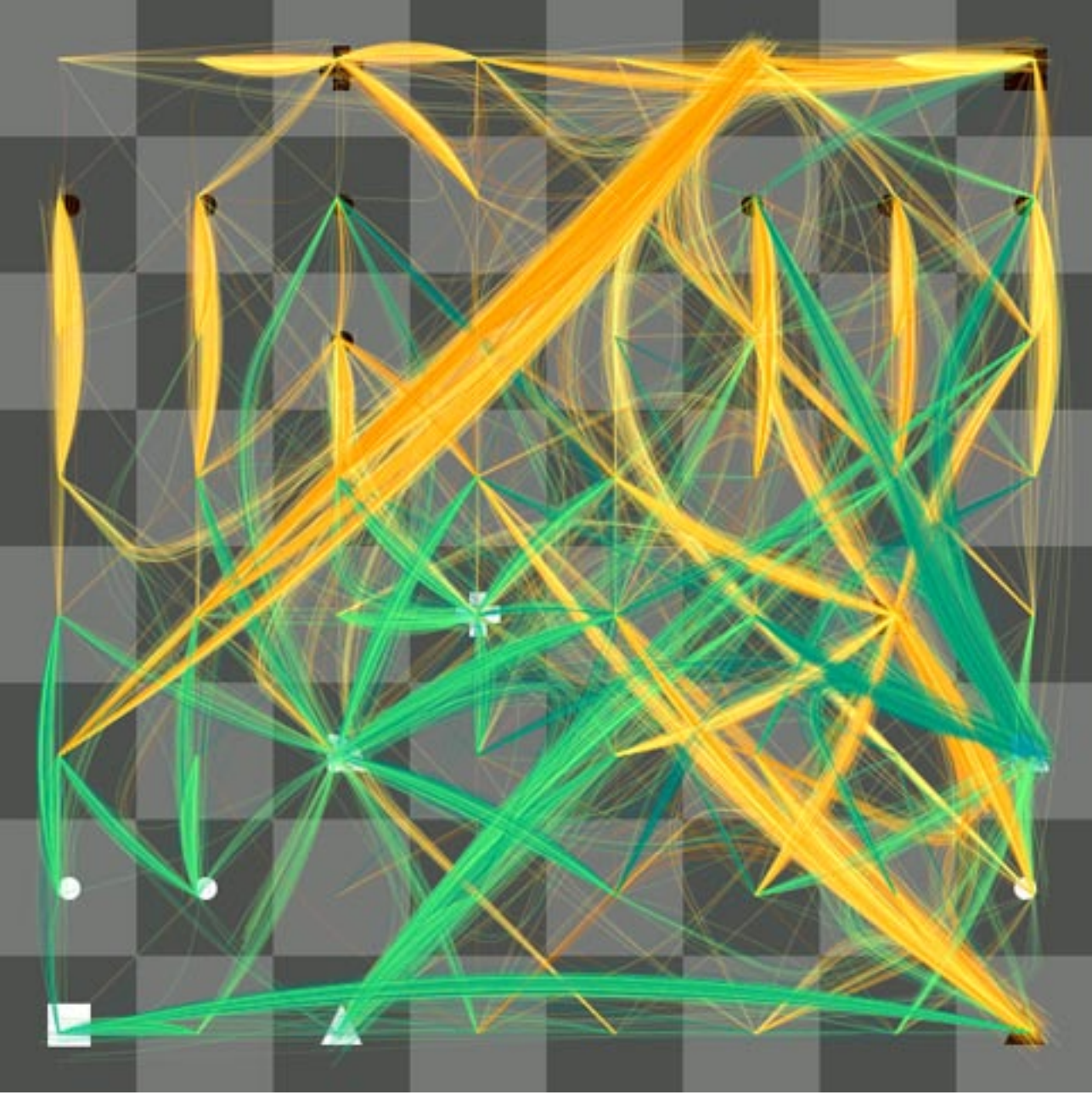


La résolution d’un Rubik’s cube est algorithmique.
Il a été inventé en 1974 par Ernö Rubik,
sculpteur et architecte hongrois.

http://fr.wikipedia.org/wiki/Cube_de_Rubik

Les échecs ont constitué l'un des premiers défis pour les développeurs informatiques et les chercheurs en intelligence artificielle. L' «arbre de coups» d'une partie d'échec est un exemple d'heuristique. L'heuristique définit les critères, méthodes, ou principes permettant de décider laquelle parmi plusieurs actions possibles promet d'être la plus adaptée pour parvenir à un but donné. Ici, Man Ray et Marcel Duchamp disputant une partie.

http://fr.wikipedia.org/wiki/Jeu_d%27échecs
http://www.marcelduchamp.net/



Martin Wattenberg a programmé un jeu d'échec avec PROCESSING.
Le programme simule graphiquement les différents coups possibles.

http://www.turbulence.org/spotlight/thinking/chess.html

Page suivante :

Planche extraite du livre «Les noms des fleurs trouvés par la méthode simple» (1904) de Gaston Bonnier.

Ce véritable livre-machine permet de trouver le nom d'une fleur ou d'une plante en répondant progressivement à un questionnaire. Les hypothèses inutiles s'excluent naturellement : le nom de la fleur ou de la plante est trouvé par élimination.

1 + Plante ***ayant des fleurs*** (Ces fleurs peuvent être quelquefois très petites, ou vertes, ou peu visibles)..... **2**

1 + Plante ***n'ayant jamais de fleurs,*** c'est-à-dire plante dont on ne voit jamais que les feuilles ou les tiges feuillées, comme les Fougères par exemple (Voir les figures aux nos 1092 à 1104).... **1092**

1092 (*vient de 1*). × Rameaux ou écailles ***verticillés*** (figures AR); ou feuilles

réduites à des collerettes dentées au sommet (figure A) et placées les unes au-dessus des autres; sporanges (c'est-à-dire petits sacs contenant les spores ou germes de la plate) ***groupés au sommet de la tige en une masse ovale*** (figure AV)..... **1104**

× Plante ***n'ayant pas à la fois*** ces caractères..... **1093**

△ Gaines de ***plus d'un centimètre et demi*** de largeur (figure T, grandeur naturelle) bordées de ***30 à 40 dents.*** → **Prêle élevée** [*Equisetum maximum*]. — Figurée en couleurs (tiges vertes) : 3, planche 64.

1104 (*vient de 1092*). △ Gaines de ***moins d'un centimètre et demi*** de largeur, bordées

de ***moins de 20 dents*** (figure A, grandeur naturelle; les figures AR et AV représentent deux formes de tiges de la même plante). → **Prêle des champs** (Queue-de-cheval, Queue-de-rat) [*Equisetum arvense*]. — **industrielle; médicinale.** — Figurée en couleurs : 2 et 2 *bis*, planche 64.

20Q.net devine à quoi vous penser en 20 questions maximum.
Compte-rendu d'une partie :

Pensez à un objet et l'intelligence artificielle essaiera de deviner à quoi vous pensez en posant des questions simples. L'objet que vous choisissez doit être connu de la plupart des gens et ne jamais être une personne, un lieu ou une chose spécifique.

Q01. Est-il Animal, Végétal, Minéral, Autre, ou Inconnue? **Autre.**
Q02. Peut-on le brûler? **Oui.**
Q03. Peut-on le trouver dans un paquet? **Non pertinent.**
Q04. Peut-on le contrôler? **Oui.**
Q05. Peut-on l'aimer? **Oui.**
Q06. Cela a-t-il des ailes? **Non pertinent.**
Q07. Cela peut-il remonter le moral? **Parfois.**
Q08. Cela contient-il un liquide? **Non.**
Q09. Est-ce fragile? **En partie.**
Q10. Est-ce mécanique? **Oui.**
Q11. Est-ce fait de métal? **En partie.**
Q12. Est-ce spontané? **Non.**
Q13. Est-ce très gros? **Non.**
Q14. Peut-on le porter sur soi? **Oui.**
Q15. Cela a-t-il un manche? **Oui.**
Q16. Y a-t-il un trou dedans? **Oui.**
Q17. Est-ce marron? **Parfois.**
Q18. Cela a-t-il des boutons? **Parfois.**
Q19. Peut-on en devenir dépendant? **Peut-être.**
Q20. Je devine que c'est une guitare électrique? **Oui.**

20Q a gagné !

http://y.20q.net/
http://www.20q.net/

Le jeu «Cluedo» : enquêter et déduire un scénario
en éliminant une à une les possibilités invraisemblables.

Arthur Ganson : Faster !
http://www.arthurganson.com/

Peter Fischli & David Weiss : Der Lauf Der Dinge, 16 mm (1987).
http://www.tcfilm.ch/lauf_txt_e.htm

Concevoir un algorithme, c'est avant tout :

analyser un problème
afin de **définir** et d'**énoncer**
l'ensemble des actions et des objets à manipuler
pour obtenir un **résultat**.

donc :

trouver le **cheminement logique**
des tâches à fournir à l'ordinateur
pour qu'il les exécute.

Pour aborder la programmation,
prenons un exemple tangible
que nous allons traduire en **algorithme**.

Comment préparer un jus d'orange ?

Comme si nous nous adressions à un extra-terrestre (l'ordinateur), nous allons :

- **Définir** les objets et les ingrédients nécessaires.
- **Lister** les opérations.
- **Ordonner** la liste des opérations.

Définir les objets et les ingrédients nécessaires.

Nous ne savons pas si nous disposons d'un presse-fruit ou non.
Mais pour résoudre notre problème,
nous devons prendre certaines décisions
qui auront une influence sur l'allure générale de notre algorithme.

Supposons que, pour préparer notre jus d'orange,
nous soyons en possession
des ustensiles et ingrédients suivants :

- un couteau
- une planche à découper
- un presse-fruit
- un verre
- deux oranges
- une paille

En fixant la liste des ingrédients et des ustensiles, nous définissons un environnement, une base de travail. Nous sommes ainsi en mesure d'établir une liste de toutes les actions à mener pour résoudre le problème et de construire la marche à suivre permettant d'obtenir un jus d'orange.

Lister les opérations.

Presser les quatre demi-oranges.
Poser une orange sur la planche à découper.
Prendre un presse-fruit.
Prendre deux oranges.
Verser le jus d'orange dans le verre.
Prendre une planche à découper.
Brancher le presse-fruit.
Prendre un couteau.
Placer une paille dans le verre.
Prendre un verre.
Découper la première orange en deux avec le couteau.
Découper la deuxième orange en deux avec le couteau.

Cette énumération est une **description** de toutes les actions nécessaires à la préparation d'un verre de jus d'orange.

Il faut faire **une chose à la fois** :
chaque action élémentaire est une **étape incompressible**
du problème donné.

En définissant l'ensemble des actions possibles, nous utilisons un **langage** minimal qui permettra de préparer le jus d'orange : ce langage est composé de différents types de mots :

verbes : prendre, verser, découper, presser, poser, brancher, placer.
objets : presse-fruit, couteau, planche à découper, verre, paille.
matières : jus d'orange.

L'étendue du langage (le nombre de mots qu'il contient)
est déterminée par l'environnement qu'on s'est fixé*.

* Pour cet exemple, l'environnement a été volontairement restreint. Nous aurions pu décrire des tâches comme «contracter les muscles de la main pour saisir le couteau», «prendre un contrat chez un fournisseur d'électricité» ou «planter un oranger», etc.
Considérons que ces tâches dépassent le cadre utile de notre objectif pédagogique.

Telle que nous l’avons décrite,
la liste des opérations ne nous permet cependant pas encore
de préparer un jus d’orange.

En suivant cette liste, tout semble y être, mais dans le désordre.
Pour arriver au jus d’orange, il reste à ordonner la liste.

Ordonner la liste des opérations.

01. Prendre un presse-fruit.
02. Prendre deux oranges.
03. Prendre une planche à découper.
04. Prendre un couteau.
05. Prendre un verre.
06. Poser une orange sur la planche à découper.
07. Découper la première orange en deux avec le couteau.
08. Poser une orange sur la planche à découper.
09. Découper la deuxième orange en deux avec le couteau.
10. Brancher le presse-fruit.
11. Presser les quatre demi-oranges.
12. Verser le jus d'orange dans le verre.
13. Placer une paille dans le verre.

L'exécution de l'ensemble **ordonné** de ces tâches
permet bel et bien de préparer un jus d'orange.

Le programme (ou plutôt l'algorithme) fonctionne !

Cet algorithme va nous servir à préciser
quatre notions de base importantes en programmation :

- La notion de **variable**.
- La notion de **bifurcation**.
- La notion de **boucle**.
- La notion de **fonction**.

variable.

bifurcation.

boucle.

fonction.

.la notion de variable

Nous avons vu que pour arriver à préparer ce fameux jus d'orange,
nous avons d'abord déterminé les objets et ingrédients
dont nous avions besoin : **prendre** une orange, un verre, etc.

De la même manière, pour concevoir un programme,
il va nous faloir non pas «prendre» des données numériques,
mais **définir** ces données, ainsi que tout objet dont nous aurons besoin.

Cette définition consiste à **nommer** ces objets
et à **décrire** leur contenu,
afin qu'ils puissent être stockés en mémoire.

Nous ne dirons plus :

prendre une **orange** *mais bien* **définir** une **variable**.

=

```
int d = 40;
int p1 = d;
int p2 = p1+d;
int p3 = p2+d;
int p4 = p3+d;
```

```
int largeur;
largeur = 200;
```

```
size (300,300);
stroke(255);
framerate(30);
```

```
stroke(random(255),random(255),random(255));
```

```
line(pmouseX,pmouseY,mouseX,mouseY);
```

Une **variable** est définie par deux éléments :

- Un **nom**
  qui sert à désigner un **emplacement** donné
  de la mémoire centrale : une **adresse**.

- Un **type**
  qui détermine la façon dont est traduite la valeur en code binaire.

exemple :

- **int** : nombre entier (integer).
- **float** : nombre à virgule (nombre réel), pour plus de précision.

.la notion de bifurcation

L’ordinateur va se poser une **question** pendant le déroulement du programme :

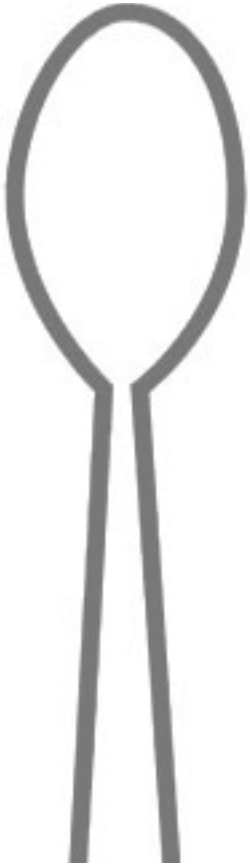
jus d’orange **sucré ou non** ?

En fonction de la **réponse** à cette question, il va bifurquer dans son déroulement et choisir de faire autre chose.

Revenons à l’algorithme du jus d’orange...

L'énoncé ainsi transformé (**sucré ou non**) nous oblige à modifier
la liste des objets manipulés,
ainsi que celle des opérations à réaliser.

Pour obtenir du jus d'orange sucré,
nous devons ajouter à notre liste un nouvel ingrédient, le **sucre** fin,
et deux nouveaux ustensiles, un **sucrier** et une **cuillère**.



De la même façon, nous devons modifier la **liste des opérations**, pour qu'elle prenne en compte les nouvelles données :

Presser les quatre demi-oranges.
Poser une orange sur la planche à découpe.
Prendre un presse-fruit.
Prendre deux oranges.
Verser une dose de sucre dans le verre.
Prendre une cuillère.
Placer une cuillère dans le verre.
Verser le jus d'orange dans le verre.
Prendre une planche à découper.
Brancher le presse-fruit.
Prendre un couteau.
Prendre un sucrier.
Placer une paille dans le verre.
Prendre un verre.
Découper la première orange en deux avec le couteau.
Découper la deuxième orange en deux avec le couteau.

Ainsi modifiée, la liste des opérations doit être **réordonnée**
afin de rechercher le moment le mieux adapté
pour ajouter les nouvelles opérations :

01. Prendre un presse-fruit.
02. Prendre deux oranges.
03. Prendre une planche à découper.
04. Prendre un couteau.
05. **Prendre un sucrier**.
06. **Prendre une cuillère**.
07. Prendre un verre.
08. Poser une orange sur la planche à découper.
09. Découper la première orange en deux avec le couteau.
10. Poser une orange sur la planche à découper.
11. Découper la deuxième orange en deux avec le couteau.
12. Brancher le presse-fruit.
13. Presser les quatre demi-oranges.
14. Verser le jus d'orange dans le verre.
15. **Verser une dose de sucre dans le verre**.
16. Placer une paille dans le verre.
17. **Placer une cuillère dans le verre**.

Ainsi ordonnée, cette marche à suivre
nous permet d'obtenir un jus d'orange sucré,
MAIS elle ne nous autorise pas à choisir entre **sucré** ou **non** !

Pour cela, nous devons introduire un test,
en posant une **condition** devant chaque instruction
concernant la prise du sucre :

01. Prendre un presse-fruit.
02. Prendre deux oranges.
03. Prendre une planche à découper.
04. Prendre un couteau.
05. **Si (jus d'orange sucré)** Prendre un sucrier.
06. **Si (jus d'orange sucré)** Prendre une cuillère.
07. Prendre un verre.
08. Poser une orange sur la planche à découper.
09. Découper la première orange en deux avec le couteau.
10. Poser une orange sur la planche à découper.
11. Découper la deuxième orange en deux avec le couteau.
12. Brancher le presse-fruit.
13. Presser les quatre demi-oranges.
14. Verser le jus d'orange dans le verre.
15. **Si (jus d'orange sucré)** Verser une dose de sucre dans le verre.
16. Placer une paille dans le verre.
17. **Si (jus d'orange sucré)** Placer une cuillère dans le verre.

Ainsi, nous obtenons du jus d'orange sucré ou non, selon notre choix.

Notons que le test **Si (jus d'orange sucré)**
est identique pour les instructions 5, 6, 15 et 17.

Nous allons **regrouper** les instructions en deux blocs distincts :

01. Prendre un presse-fruit. 02. Prendre deux oranges. 03. Prendre une planche à découper. 04. Prendre un couteau. 05. Prendre un verre. 06. Poser une orange sur la planche à découper. 07. Découper la première orange en deux avec le couteau. 08. Poser une orange sur la planche à découper. 09. Découper la deuxième orange en deux avec le couteau. 10. Brancher le presse-fruit. 11. Presser les quatre demi-oranges. 12. Verser le jus d'orange dans le verre. 13. Placer une paille dans le verre.	Bloc II : Préparer le jus d'orange

01. Si (jus d'orange sucré) Prendre une cuillère. 02. Si (jus d'orange sucré) Prendre un sucrier. 03. Si (jus d'orange sucré) Verser une dose de sucre fin dans le verre. 04. Si (jus d'orange sucré) Placer une cuillère dans le verre.	Bloc II : sucrer le jus

La réalisation du **Bloc I «Préparer le jus d'orange»** nous permet d'obtenir du jus d'orange.

Ensuite, en exécutant le test **Si (jus d'orange sucré)**, deux solutions sont possibles:

1- La proposition (jus d'orange sucré) est **vraie**, alors les instructions 1 à 4 du **Bloc II «Préparer le sucre»** sont exécutées. Nous obtenons du jus d'orange sucré.

2- La proposition (jus d'orange sucré) est **fausse**, et les instructions qui suivent ne sont pas exécutées. Nous obtenons un jus d'orange non sucré.

L'instruction if/else (si/sinon) découle de ce principe.

Elle permet de programmer un choix,
en plaçant une condition derrière le terme **if**,
comme nous avons placé une condition derrière le terme **si**
de l'algorithme du jus d'orange, sucré ou non.

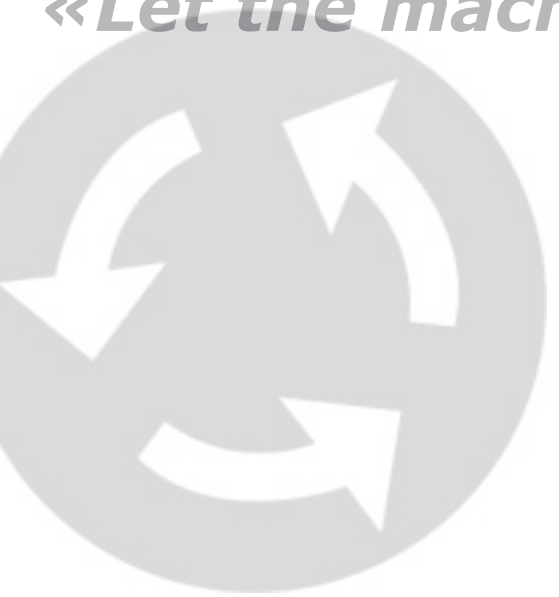
```
if (condition) // si la condition est VRAIE
{ // faire
ceci ou cela; // instructions
}
else // sinon (si la condition ci-dessus est FAUSSE)
{ // faire
ceci ou cela; // instructions
}
```

.la notion de boucle

Combien de dose de sucre fin dans votre jus d'orange ?

Pour aborder cette notion de répétition ou de boucle,
nous allons améliorer l'algorithme du jus d'orange sucré,
de sorte que le programme demande à l'utilisateur
de prendre une dose de sucre fin
autant de fois qu'il le souhaite.

«Let the machine do the job !»



Pour cela, reprenons uniquement
le bloc d'instructions II : **Préparer le sucre.**

01. Si (jus d'orange sucré) Prendre une cuillère.
02. Si (jus d'orange sucré) Prendre un sucrier.
03. Si (jus d'orange sucré) Verser une dose de sucre fin dans le verre.
04. Si (jus d'orange sucré) Placer une cuillère dans le verre.

Bloc II :
sucrer le jus

L'exécution du bloc d'instructions II **Préparer le sucre**
nous permet de verser une seule dose de sucre dans le verre.

Si nous désirons mettre **plus** de sucre,
nous devons exécuter l'instruction 3
autant de fois que nous souhaitons de dose de sucre fin.

La marche à suivre devient alors :

Prendre une cuillère.
Prendre un sucrier.
Début répéter:
02. Verser une dose de sucre dans le verre.
03. Poser la question: «Souhaitez-vous une autre dose de sucre ?»
04. Attendre la réponse.
Tant que la réponse est **OUI**, retourner à **Début répéter**.
Si la réponse est **NON**, on sort de cette boucle infernale !

.la notion de fonction

En donnant un nom à un bloc d'instructions,
nous pouvons définir un sous-programme,
appelé **fonction** ou **méthode**.

Une fonction ressemblerait à une petite usine
(on y entre des matières premières qui ressortent transformées...)
ou plutôt à une **machine** à l'intérieur d'une usine (le programme...)

Enfin, bon : préparons à présent **un jus de citron** !

Certains algorithmes peuvent être appliqués à des problèmes voisins
en modifiant simplement les données pour lesquels ils ont été construits.

En faisant varier certaines valeurs,
le programme fournit un résultat différent du précédent.

Ces valeurs, caractéristiques du problème à traiter,
sont appelées **paramètres** du programme.

Pour comprendre concrètement ce concept,
nous allons reprendre notre algorithme du jus d'orange
pour le transformer en un algorithme
qui nous permettra de faire du **jus d'orange OU du jus de citron.**

Préparer un jus d'orange ou préparer un jus de citron
est une opération à peu près semblable.

En reprenant la liste de toutes les opérations nécessaires
à la réalisation d'un jus d'orange,
nous constatons qu'en remplaçant simplement
le mot **orange** par le mot **citron**,
nous obtenons du jus de citron.

01. Prendre un presse-fruit.
02. Prendre deux **oranges/citrons**.
03. Prendre une planche à découper.
04. Prendre un couteau.
05. Prendre un verre.
06. Poser **une orange/un citron** sur la planche à découper.
07. Découper **la 1e orange/le 1e citron** en deux avec le couteau.
08. Poser **une orange/un citron** sur la planche à découper.
09. Découper **la 2e orange/le 2e citron** en deux avec le couteau.
10. Brancher le presse-fruit.
11. Presser les quatre **demi-oranges/demi-citrons**.
12. Verser le jus **d'orange/de citron** dans le verre.
13. Placer une paille dans le verre.

Définir les paramètres.

Pour éviter d'avoir à recopier plusieurs fois
des marches à suivre qui ne diffèrent que sur un détail,
l'idée est de construire un **algorithme général**.

Cet algorithme ne varie qu'en fonction d'**ingrédients déterminés**,
qui font que le programme donne un résultat différent.

En **généralisant** l'algorithme du jus de citron ou du jus d'orange,
on exprime une marche à suivre permettant de réaliser un **jus de fruits**.

Pour obtenir un résultat différent (oranges ou citrons),
il suffit de définir l'**ingrédient** comme paramètre de l'algorithme.

La marche à suivre s'écrira donc en remplaçant
les mots **orange** ou **citron** par le mot **ingrédient**.

Préparer un jus de (ingrédient**)**

01. Prendre un presse-fruit.
02. Prendre deux **(ingrédient)**.
03. Prendre une planche à découper.
04. Prendre un couteau.
05. Prendre un verre.
06. Poser un **(ingrédient)** sur la planche à découper.
07. Découper le 1e **(ingrédient)** en deux avec le couteau.
08. Poser un **(ingrédient)** sur la planche à découper.
09. Découper le 2e **(ingrédient)** en deux avec le couteau.
10. Brancher le presse-fruit.
11. Presser les quatre demi-**(ingrédient)**.
12. Verser le jus de **(ingrédient)** dans le verre.
13. Placer une paille dans le verre.

Bloc :
Préparer
(ingrédient)

En paramétrant un algorithme,
nous n'avons plus besoin de recopier plusieurs fois
les instructions qui le composent pour obtenir un résultat différent.

En donnant un nom au bloc d'instructions
correspondant à l'algorithme général **Préparer()**,
nous définissons un sous-programme (une fonction)
capable d'être exécuté autant de fois que nécessaire.

Il suffit pour cela d'appeler la **fonction** ou **méthode** par son nom.

Un algorithme, ensemble **ordonné** des tâches à accomplir
est donc la description d'un programme.

Il reste à traduire tout cela
dans un langage compréhensible par la machine...

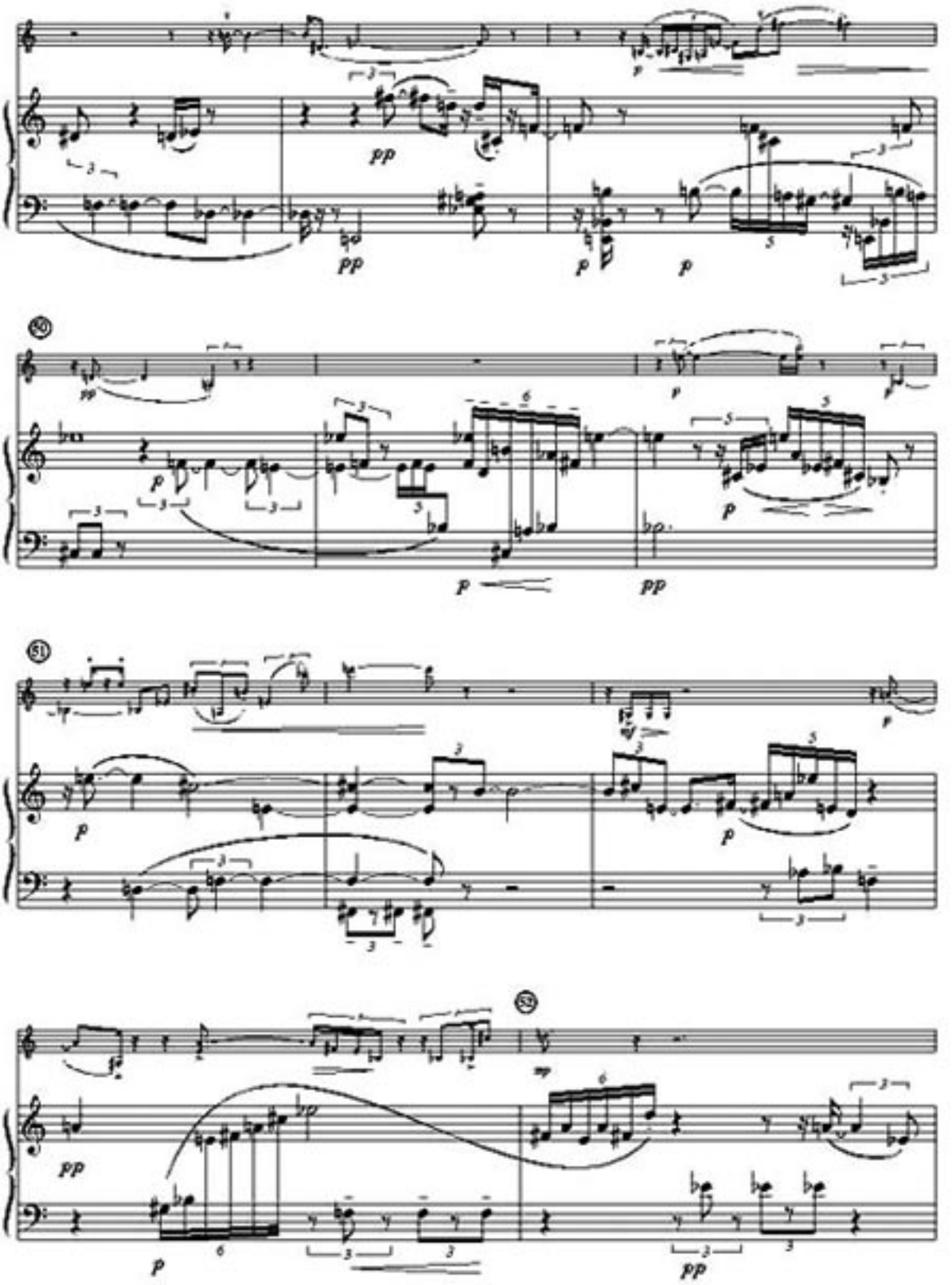
En savoir plus sur les algorithmes :
http://www.commentcamarche.net/algo/algointro.php3
http://fr.wikipedia.org/wiki/Algorithme

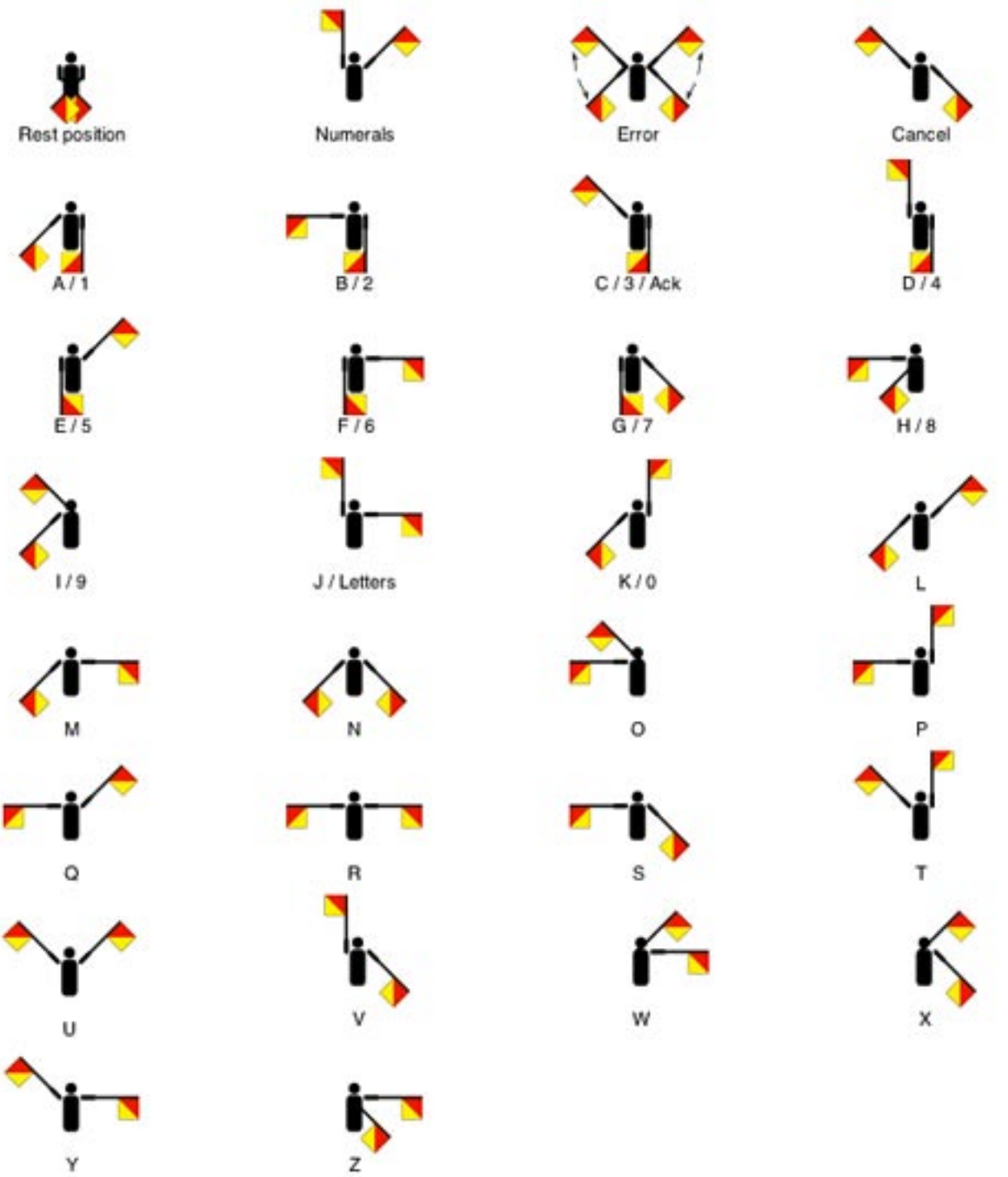
.langage

Le terme **langage** désigne un système spécifique d'expression de la pensée, au moyen de **signes**, comportant un **vocabulaire** et une **grammaire** définis.

La Pierre de Rosette, découverte en 1799,
décryptée par Jean-François Champollion après huit ans de travail !

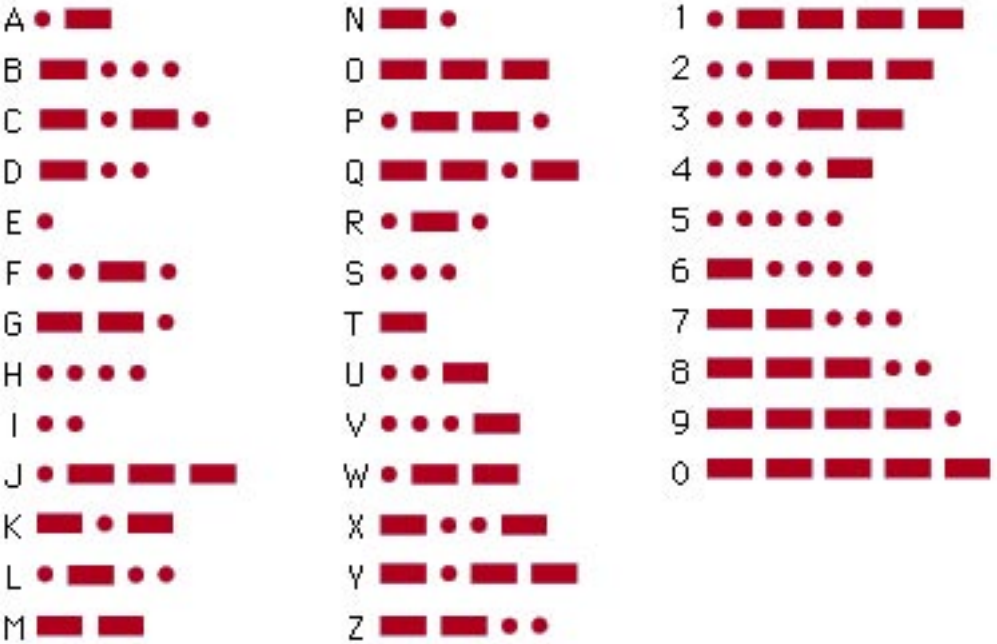
http://fr.wikipedia.org/wiki/Pierre_de_Rosette


50
51
52



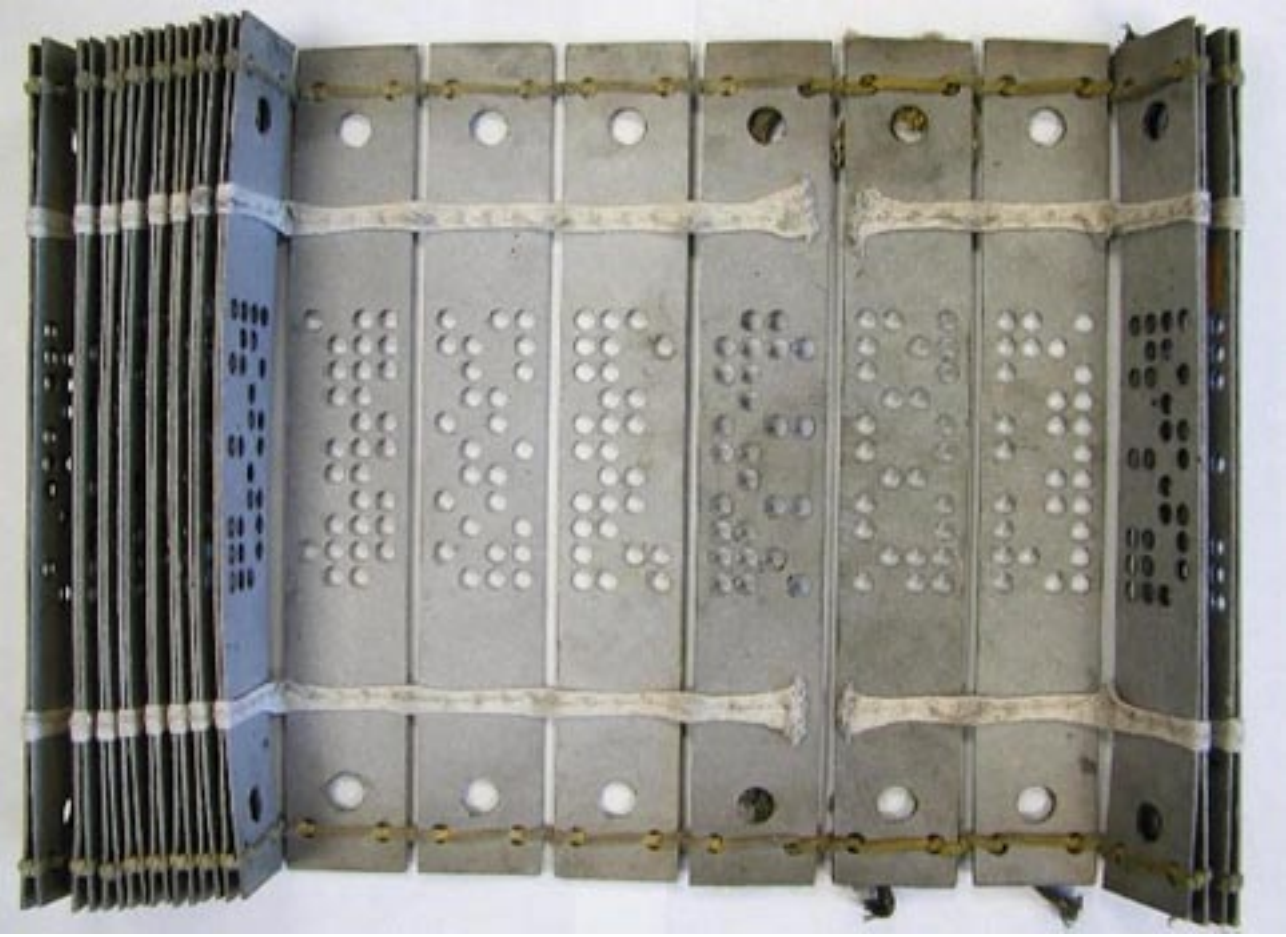
L’alphabet du langage Sémaphore.

http://en.wikipedia.org/wiki/Semaphore_(communication)



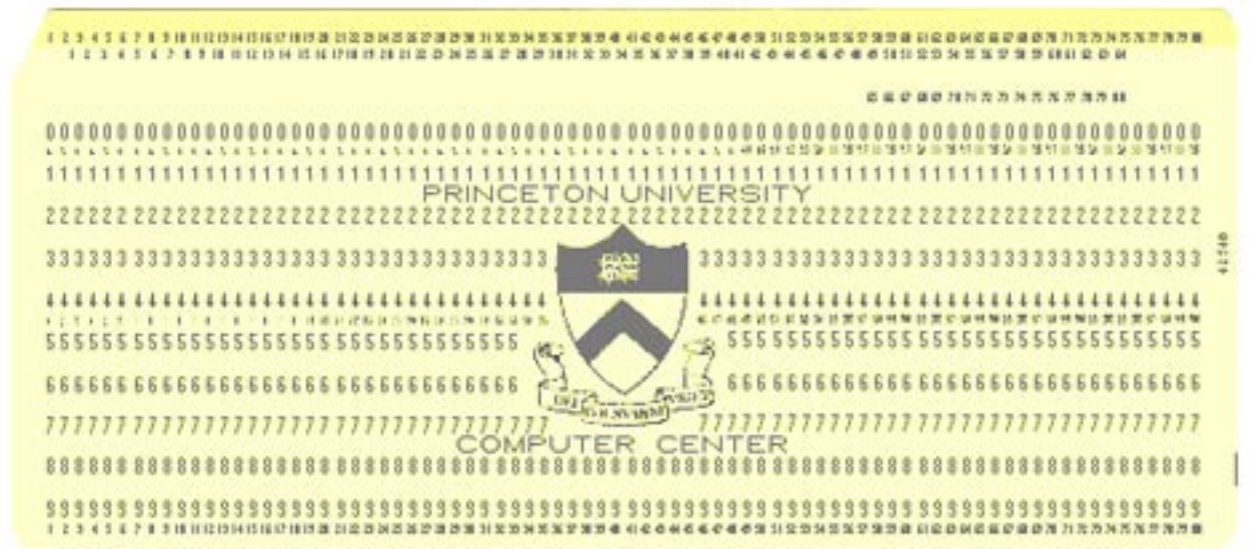
Le code Morse, conçu en 1838 par le peintre Samuel Morse.

http://en.wikipedia.org/wiki/Samuel_Morse



1801 : carte perforée guidant les crochets du métier à tisser Jacquard.

http://fr.wikipedia.org/wiki/M%C3%A9tier_Jacquard

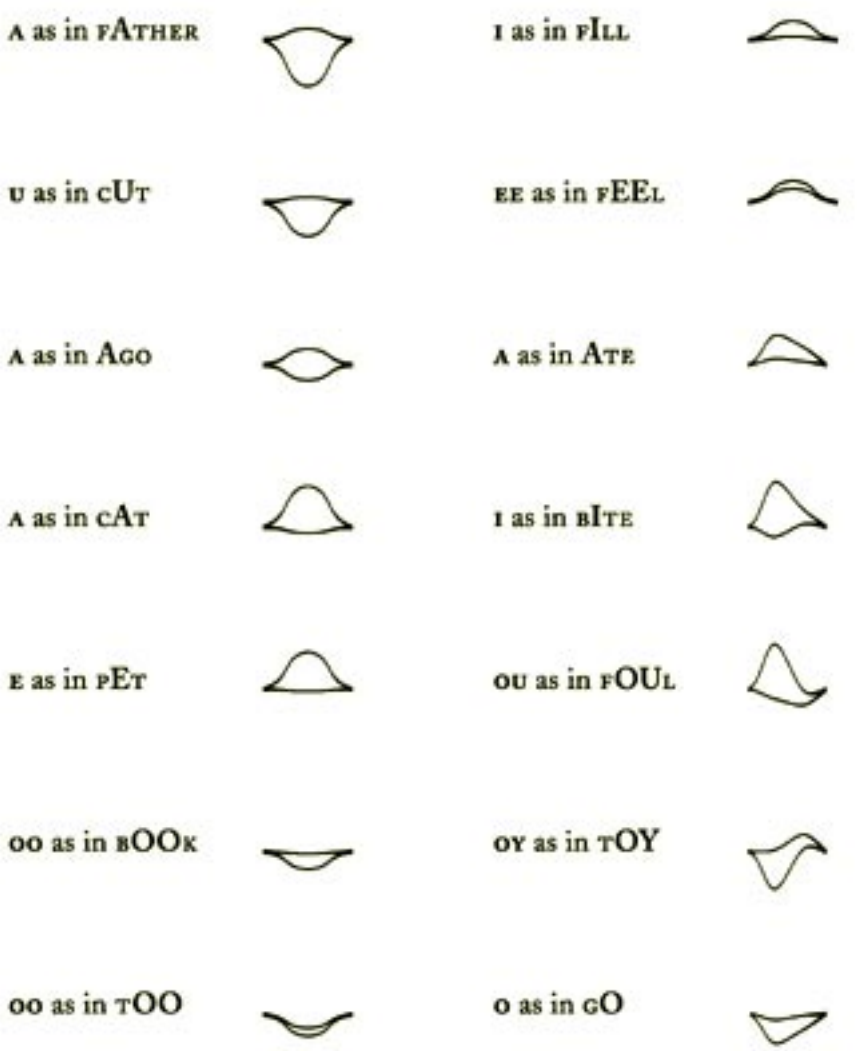


Cartes perforées, premiers supports d'entrée-sortie
et premières mémoires de masse au début de l'informatique.

http://fr.wikipedia.org/wiki/Carte_perforée

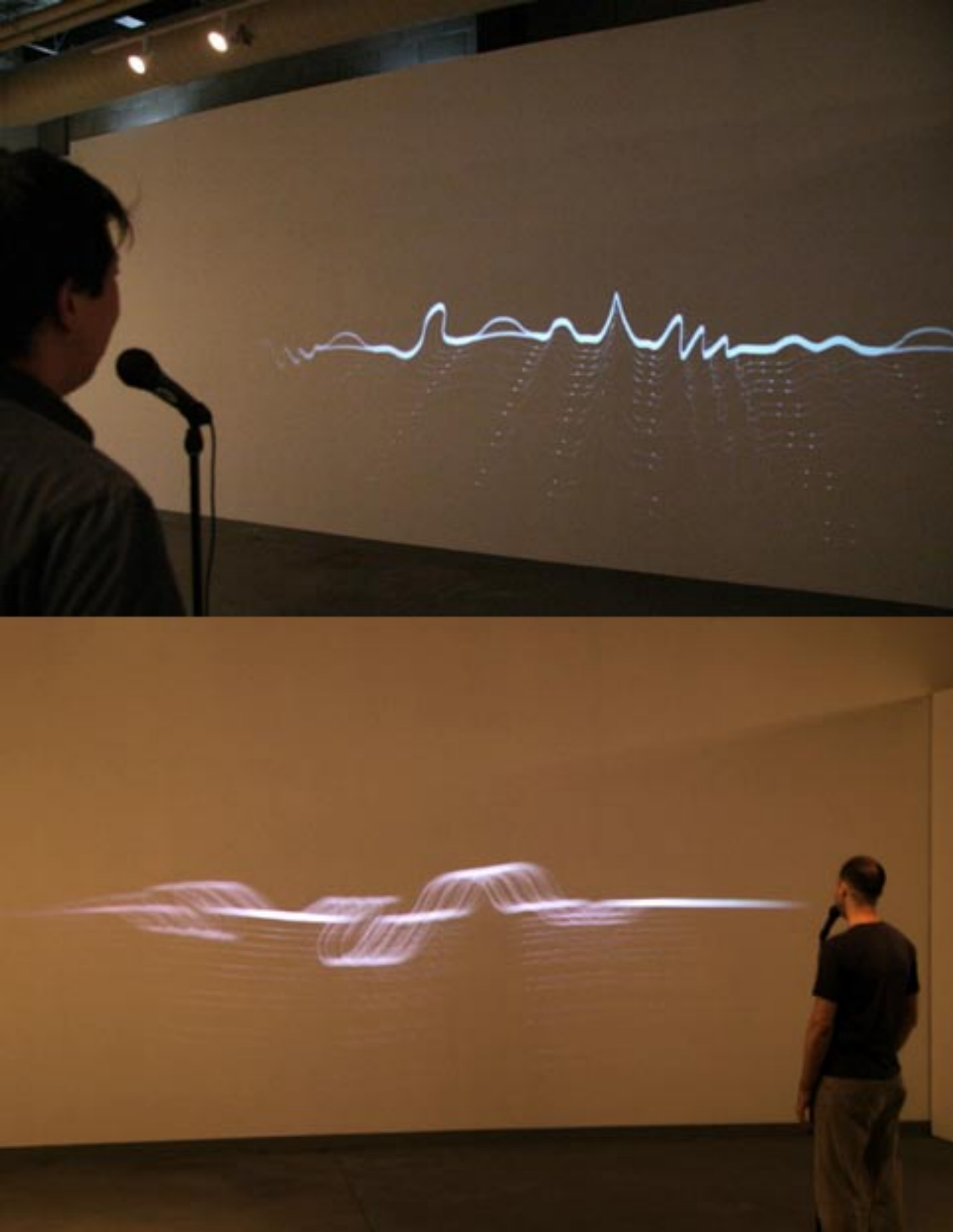
Le système d’écriture tactile Braille.

http://fr.wikipedia.org/wiki/Braille



Peter Cho : Takeluma (2005),
extrait d'un système de représentation de la langue parlée
utilisé dans une installation interactive.

http://www.pcho.net/takeluma



ursonate

einleitung:

Fümms bö wö tää zää Uu, 1
pögiff,
kwii Ee.

Ooooooooooooooooooooooooooooooo, 6

dll rrrrrr beeeee bö, (A) 5
dll rrrrrr beeeee bö fümms bö,
rrrrrr beeeee bö fümms bö wö,
beeeee bö fümms bö wö tää,
bö fümms bö wö tää zää,
fümms bö wö tää zää Uu:

erster teil:

thema 1:

Fümms bö wö tää zää Uu, 1


Rinnzeke
Rinnzeke

Pages précédentes :

Ursonate de Kurt Schwitters.
Voir et écouter :
http://www.ubu.com/historical/schwitters/ursonate.html
http://www.ubu.com/sound/schwitters.html
http://www.peak.org/~dadaist/English/Graphics/ursonate.html

Ursonography de Jaap Blonk & Golan Levin.
Voir :
http://flickr.com/photos/golanlevin/sets/72157594363837537/

Un **langage de programmation**
est un dialecte dans lequel on peut exprimer des programmes.

Ce langage fournit une **syntaxe**,
c’est-à-dire une façon de représenter les ordres donnés à l’ordinateur
qui sera plus facilement manipulable par un être humain.

Ce langage évolué (dit de «haut niveau»)
sera ensuite traduit en **langage machine** (dit de «bas niveau»),
langue maternelle de l'ordinateur composée exclusivement de **0** et de **1**.

Le programme, dans sa forme compréhensible par un humain,
est appelé «**code source**».

Pour en savoir plus :
http://www.commentcamarche.net/langages/langages.php3
http://fr.wikipedia.org/wiki/Langage_de_programmation

Le code binaire est un code à 2 moments.

Il suffit de deux symboles
pour exprimer une information binaire :
0 et 1 (soit, en logique, «faux» et «vrai»).

10011010
(un octet, soit 8 bits)
représente une information binaire utilisant 8 positions.

Chaque position porte le nom de «bit».

En code binaire, le terme **bit** (**bi**nary digi**t**)
est l'équivalent des termes «chiffres» ou «lettres».

13

s'exprime avec deux symboles, chacun étant choisi parmi les chiffres 0 à 9. Nous avons utilisé ici deux **positions** d'un code à dix **moments**, ces dix moments étant les dix chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Treize

s'exprime avec six symboles (lettres), chacun étant choisi parmi les vingt-six lettres de l'alphabet. Nous avons utilisé six **positions** d'un code à vingt-six **moments**.

XIII

s'exprime avec 4 **positions** d'un code à sept **moments**:
les chiffres romains (I, V, X, L, C, M, D).

.interprétation du langage

Avec l'aide d'un **interprète**, nous communiquons avec une chinoise sans connaître sa langue :

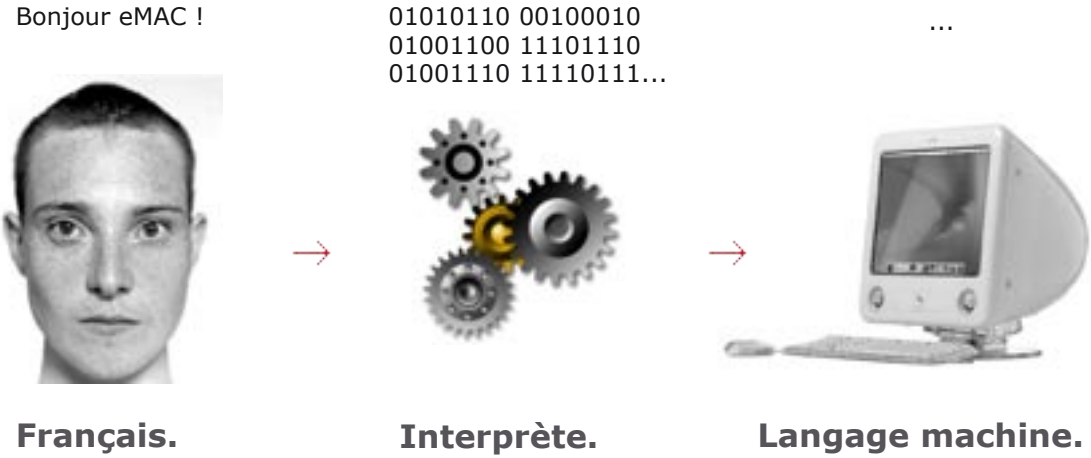
Bonjour Mademoiselle Li !

早，李太太，您早！

...

Français. **Interprète.** **Chinois.**

De la même manière, nous pourrons communiquer avec l'ordinateur sans connaître son langage. Nous allons employer l'ordinateur lui-même (ou plus exactement un programme appelé «**interprète**» ou «**compilateur**» selon le cas) pour traduire une instruction en langage machine, le langage compris par l'ordinateur :



Remarque :
«Communiquer avec l'ordinateur» est une notion toute relative : l'eMAC ne comprend rien à ce que je lui raconte. Il se borne à exécuter. Lui dire bonjour est donc inutile... Le test habituel est «Hello world !», formule consacrée qui démontre que nous allons communiquer *à partir de notre application* avec le monde, mais aussi que nous traitons notre programme comme une *entité autonome qui parlerait en son nom propre.*

Ou plus exactement :

Français : Bonjour eMAC !

Java :

```
public class Hello {
      public static void main(String[] args)
      {
          System.out.println(«Bonjour eMac !»);
      }
}
```

Langage-machine :

```
01010110 00100010
01001100 11101110
01001110 11110111...
```

Affichage :

Et plus exactement encore :

Français : Bonjour eMAC !

Processing :

```
size(200, 200);
background(255);
PFont fontA = loadFont(«HelveticaNeue-
48.vlw»);
textFont(fontA, 14);
int x = 50;
fill(0);
text(«Bonjour eMac !», x, 100);
```

Java :

```
public class Hello {
      public static void main(String[] args)
      {
          System.out.println(«Bonjour eMac !»);
      }
}
```

Assembleur :

```
       .global _start
BONJ:   .ascii  «Bonjour\n»
_start: mov     $4      , %eax
        mov     $1      , %ebx
        mov     $BONJ   , %ecx
        mov     $8      , %edx etc...
```

Langage-machine :

```
01010110 00100010
01001100 11101110
01001110 11110111...
```

Affichage :

Processing est un environnement
qui simplifie la création de programmes
dans un langage plus extensible,
plus puissant, mais plus complexe : Java.

Processing fonctionne grâce à Java et ses outils.

Java est un langage **compilé**.
Il est traduit en langage machine à l'aide d'un compilateur.

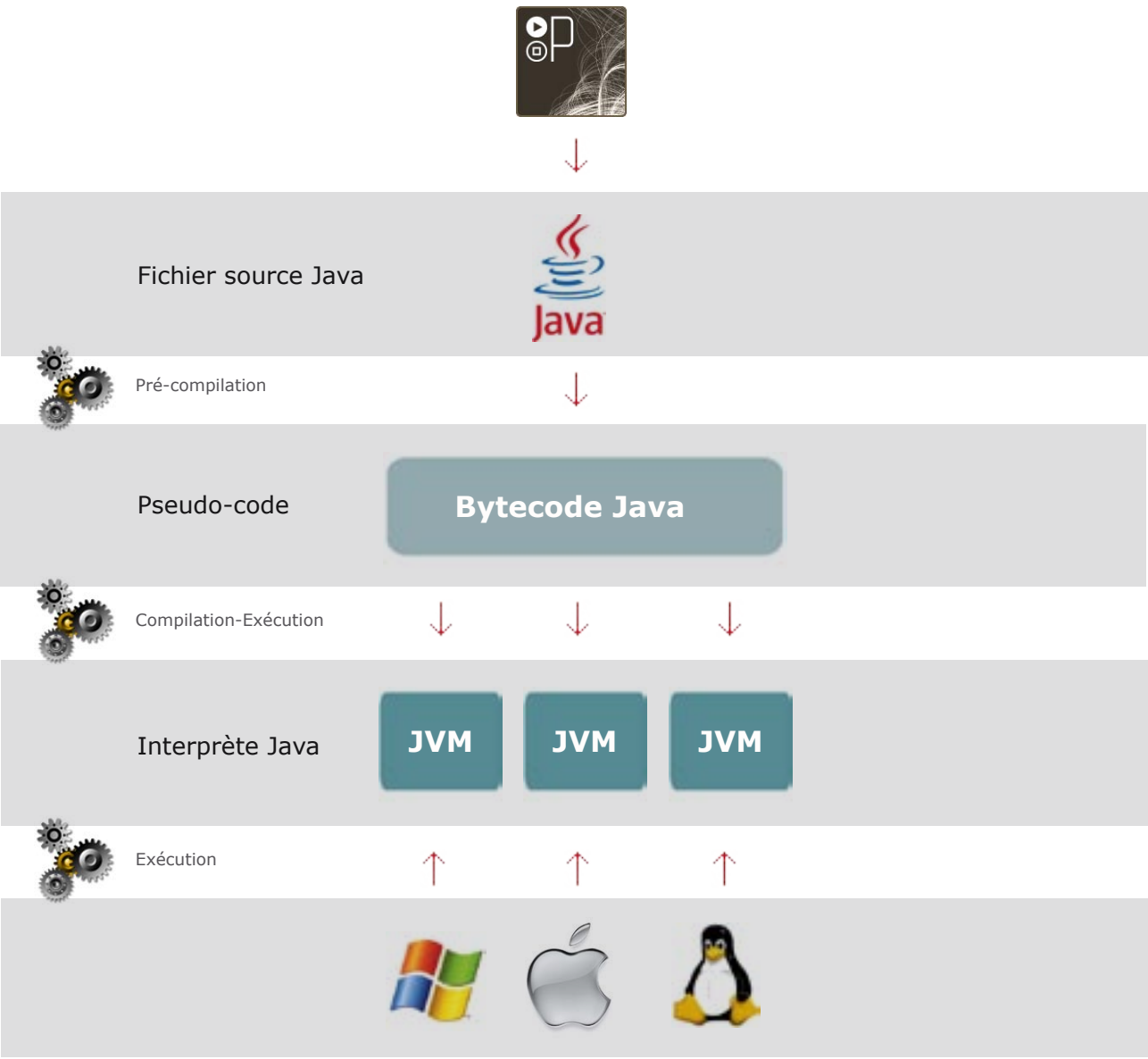
Contrairement aux langages compilés traditionnels, pour lesquels le compilateur crée un fichier binaire directement exécutable par un processeur donné, le code source Java est compilé en un langage intermédiaire : le **pseudo-code** ou **bytecode**.

Grâce à cette caractéristique, un programme écrit en Java est **portable**,
(il ne dépend pas d'une plate-forme donnée).

En réalité, le **bytecode** est interprété
par une machine virtuelle (Java Virtual Machine),
un interpréte Java.

Chaque plate-forme (Mac OSX, Windows, Linux)
possède sa propre machine virtuelle.

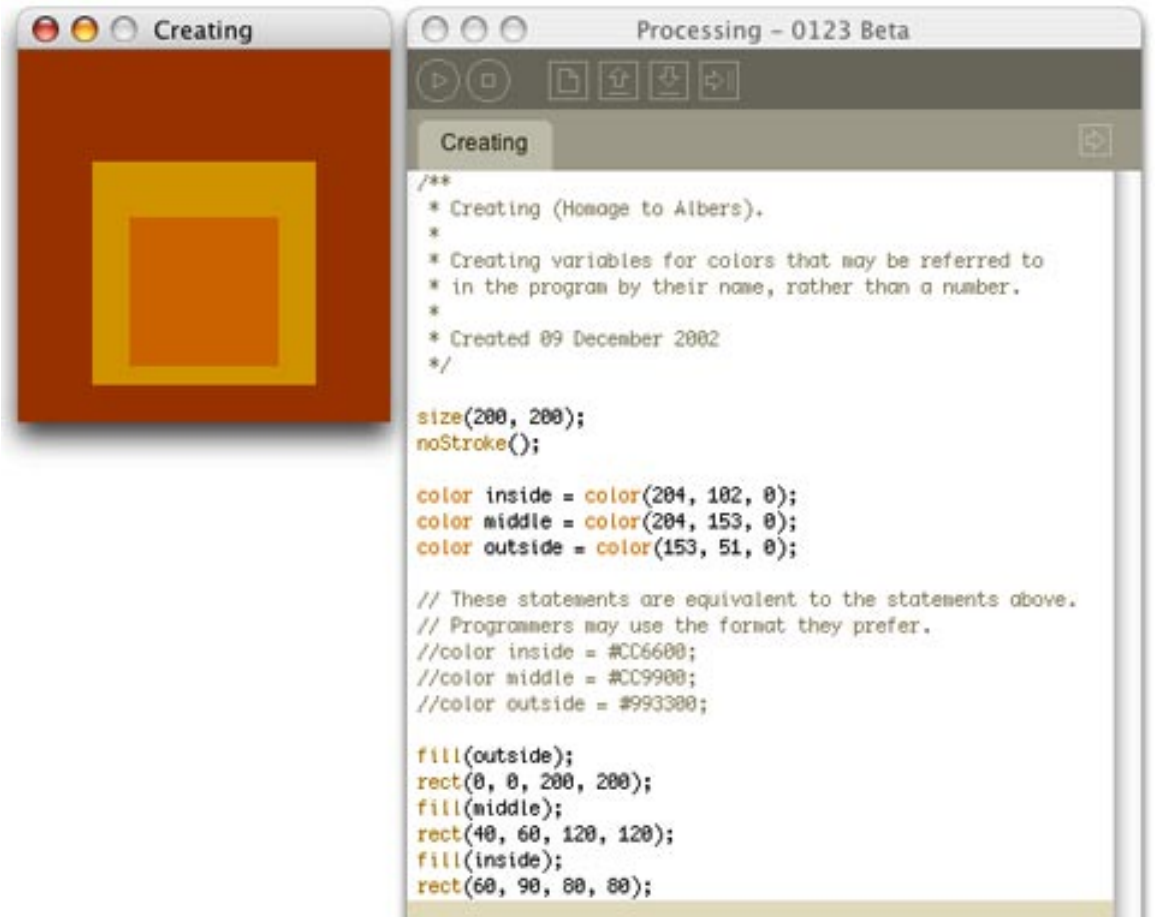
Pour être précis et complet, voici donc ce qui se passe en coulisses :



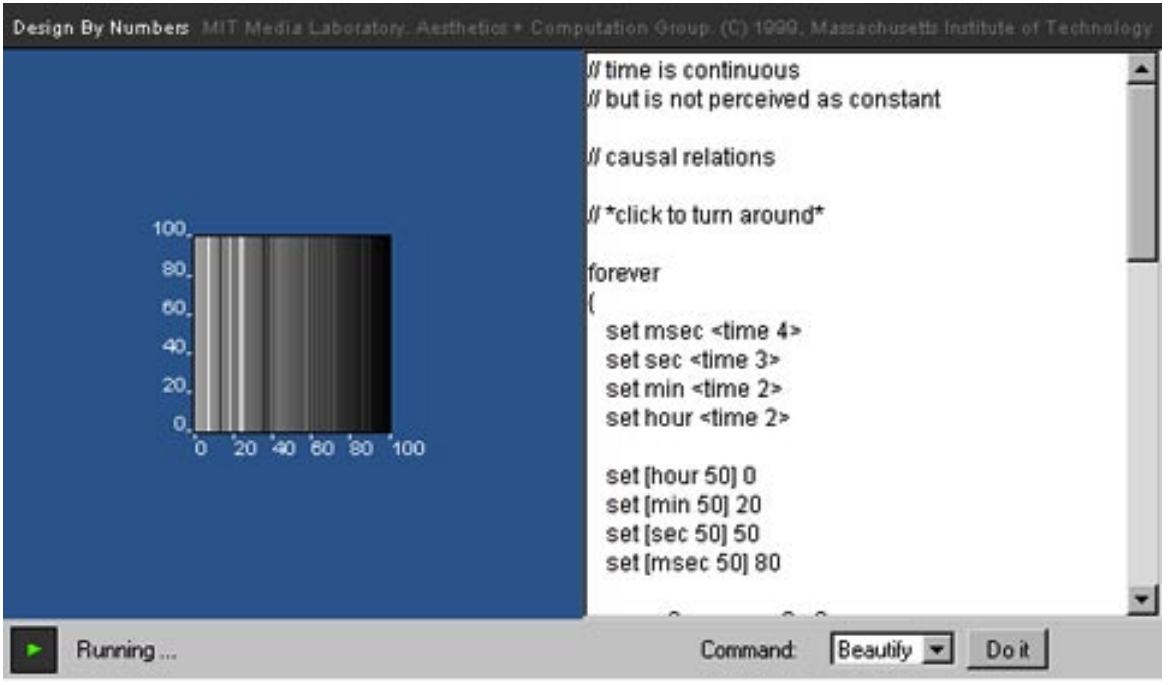
.Processing

PROCESSING est un logiciel libre (open-source),
gratuit et multi-plateformes (Windows, Mac OS X et Linux).
Il a été conçu par Casey Reas et Benjamin Fry comme outil de création et d'apprentissage fondamental à la programmation.

Il est téléchargeable ici :
http://processing.org/download/



Ben Fry et Casey Reas sont d'ex-étudiants de John Maeda à l'Aesthetics & Computation Group du MIT. John Maeda étant lui-même auteur de **Design By Numbers**, environnement précurseur d'apprentissage fondamental, on peut dire que PROCESSING en est le descendant direct.

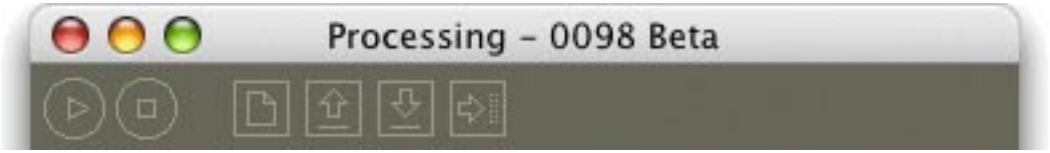


PROCESSING utilise un langage simple et complet,
orienté vers le développement de «sketches»,
programmes dédiés essentiellement à des projets artistiques.

Le traitement du son et de captures vidéo,
l'accès en ligne (et bien d'autres choses encore)
sont rendus possibles grâce à des bibliothèques.

http://processing.org/reference/libraries/index.html

PROCESSING est sain et réduit à l'essentiel :



Run :
exécute le code
(compile le code, ouvre la fenêtre de visualisation
et exécute le programme à l'intérieur)

Stop :
arrête l'exécution d'un programme,
mais ne ferme pas la fenêtre de visualisation.

New :
crée une nouveau «sketches».

Open :
sélectionne et charge un sketche existant.

Save :
enregistre le sketche courant dans le «sketchbook».
Si vous voulez donner un autre nom au croquis,
vous pouvez choisir «Save As» dans le menu «File».

Export :
exporte le sketche courant dans le «sketchbook» comme Applet
JAVA intégré dans une page HTML.

Nous allons ré-examiner les quatre notions de base vue précédemment sous forme d'algorithme appliquées cette fois à l'environnement Processing :

variable.

bifurcation.

boucle.

fonction.

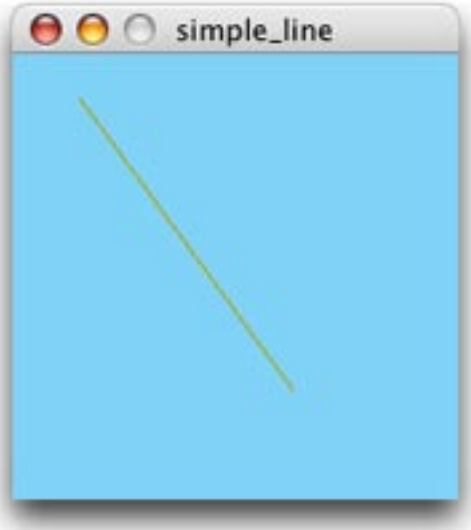
.une variable dans Processing

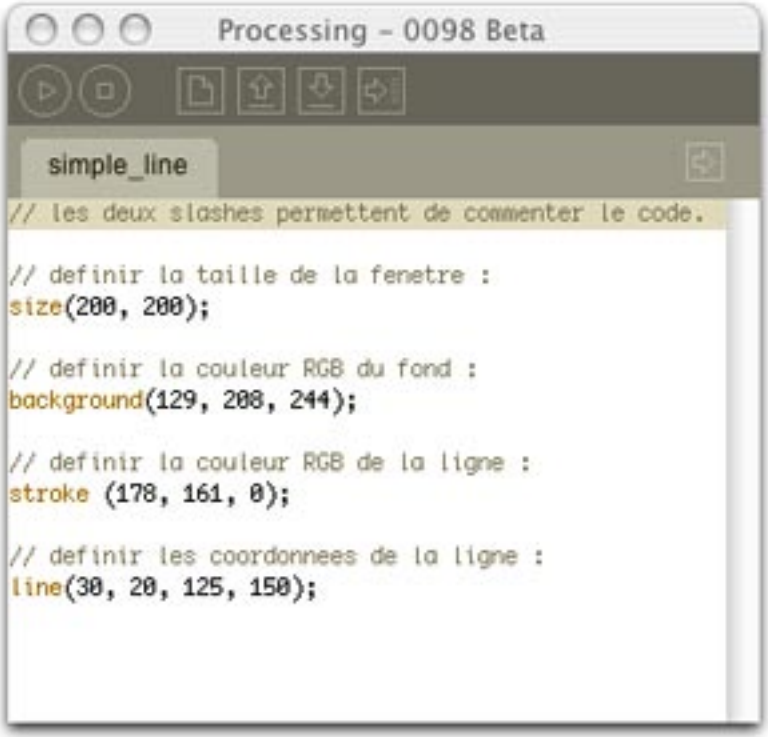
Rappel:
une variable est définie par deux éléments :
un nom et un type.

Certaines **variables** sont prédéfinies par Processing,
mais nous pouvons en créer de manière illimitée,
en respectant certaines règles : respect de la casse, pas d'espace, certains caractères spéciaux sont exclus, etc.

Il existe plusieurs **types** de données numériques.
En voici quelques-uns:

- **int** : nombre entier (integer).
- **float** : nombre à virgule (nombre réel), pour plus de précision.
- **boolean** : valeur logique possèdant 2 états : «true» et «false».
- **char** : représente des caractères isolés.
- **string** : décrit une séquence de caractères.

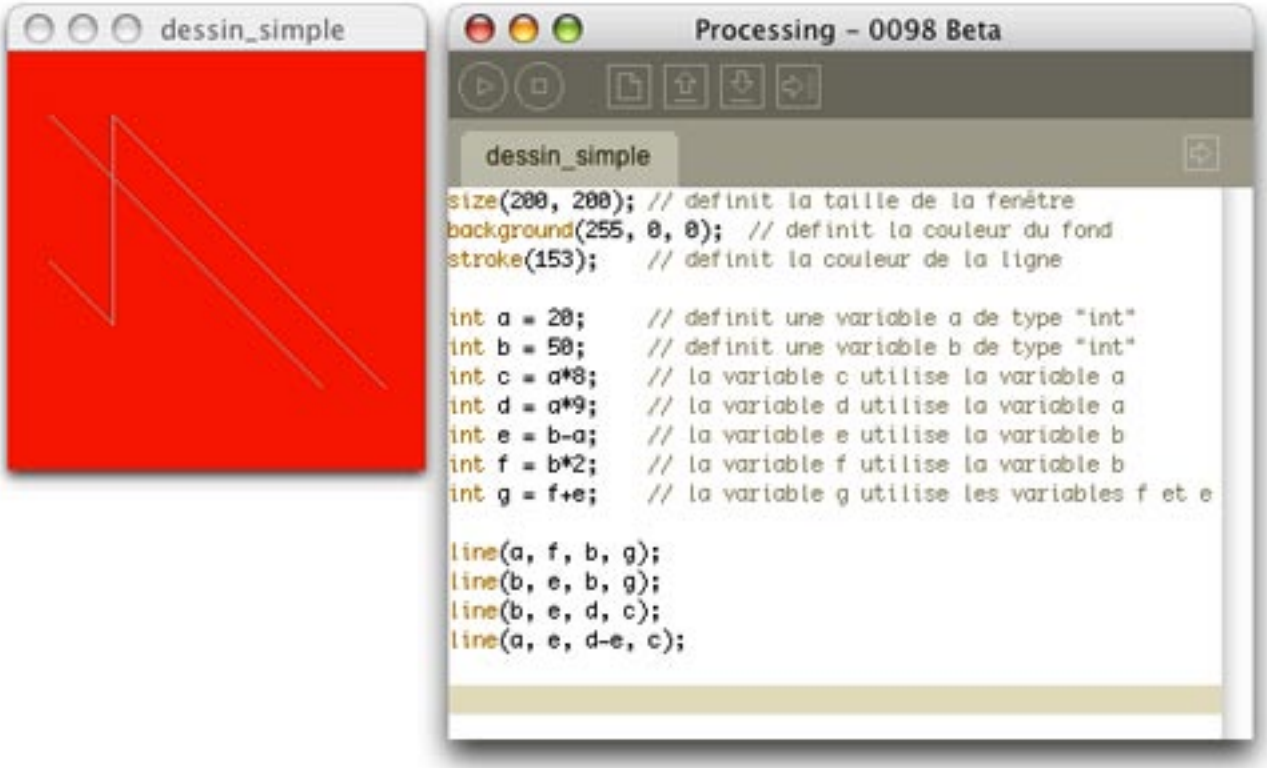




Un exemple simple.

Plusieurs variables prédéfinies par Processing,
suivies de leurs paramètres (de leurs valeurs).

Voir ce programme en ligne.
Télécharger ce programme.



Un exemple plus complexe.

Plusieurs variables globales définies, à la fois par des valeurs de nombre et par des opérations effectuées entre elles.
Une fois définies en début de programme, ces variables pourront être réutilisées autant de fois qu'on le souhaite.

Voir ce programme en ligne.
Télécharger ce programme.

```
size(200, 200); // définit la taille de la fenêtre
background(255, 0, 0);  // définit la couleur du fond
stroke(153);    // définit la couleur de la ligne

int a = 20;     // définit une variable a de type «int»
int b = 50;     // définit une variable b de type «int»
int c = a*8;    // la variable c utilise la variable a
int d = a*9;    // la variable d utilise la variable a
int e = b-a;    // la variable e utilise la variable b
int f = b*2;    // la variable f utilise la variable b
int g = f+e;    // la variable g utilise les variables f et e

line(a, f, b, g);  // dessin de ligne utilisant les variables
line(b, e, b, g);
line(b, e, d, c);
line(a, e, d-e, c);
```

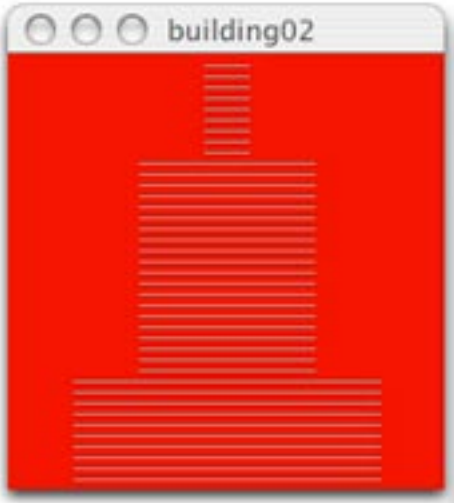
Raccourcis MacOSX pour les caractères spéciaux :

[= maj+alt+(
] = maj+alt+)
{ = alt+(
} = alt+)

.une bifurcation dans Processing

Rappel : l’ordinateur va se poser une **question** pendant le déroulement du programme.

En fonction de la **réponse** à cette question, il va bifurquer dans son déroulement et choisir de faire autre chose.



Processing - 0123 Beta

building02

```
size(200, 200); // definit la taille de la fenêtre
background(255, 0, 0);  // definit la couleur du fond
stroke(153);    // definit la couleur de la ligne

int l = 30; // definit une variable l
int r = width-l;  // definit une variable r

// pour (initialisation; condition; incrémentation)
// pour h = 5, tant que h vaudra moins de la largeur
// h sera augmenté de 5
for(int h = 5; h < width; h += 5) {

  // si h vaut moins de 50 pixels
  // tant que h n'a pas atteint la valeur de 50 pixels
  if(h < 50) {

    // le programme dessine des lignes (x,y,x,y)
    line( 3*l, h, r-2*l, h );
  }

  // autrement si h vaut moins de 150 pixels
  // tant que h n'a pas atteint la valeur de 150 pixels
  else if(h < 150) {

    // le programme dessine des lignes (x,y,x,y)
    line( 2*l, h, r-l, h );
  }

// sinon
  else {

    // le programme dessine des lignes (x,y,x,y)
    line( l, h, r, h );
  }
}
```

Exemple :

Le programme va dessiner un building à 3 étages
en choisissant entre différentes conditions.

Voir ce programme en ligne.
Télécharger ce programme.

On définit deux variables puis une structure de condition pour espacer les lignes en hauteur, selon 3 étages de largeurs différentes.

L'expression for() prend toujours la forme :
for(initialisation**;** condition de la boucle**;** incrémentation**)**

L'incrémentation peut prendre différentes formes :
i++ (augmente de 1) **i+=5** (i = i+5)
i-- (diminue de 1) **i-=5** (i = i-5)

{entre accolades : les instructions de la boucle, ce qu'elle doit faire**}**

```
size(200, 200); // définit la taille de la fenêtre
background(255, 0, 0);  // définit la couleur du fond
stroke(153);    // définit la couleur de la ligne

int l = 30;  // définit une variable l
int r = width-l;  // définit une variable r

// pour (initialisation; condition; incrémentation)
// pour h = 5, tant que h vaudra moins de la largeur
// h sera augmenté de 5
for(int h = 5; h < width; h += 5) {

  // si h vaut moins de 50 pixels
  // tant que h n'a pas atteint la valeur de 50 pixels
  if(h < 50) {

    // le programme dessine des lignes (x,y,x,y)
    line( 3*l, h, r-2*l, h );
  }

  // autrement, si h vaut moins de 150 pixels
  else if(h < 150) {

    // le programme dessine des lignes (x,y,x,y)
    line( 2*l, h, r-l, h );
  }

// sinon
  else {

    // le programme dessine des lignes (x,y,x,y)
    line( l, h, r, h );
  }
}
```

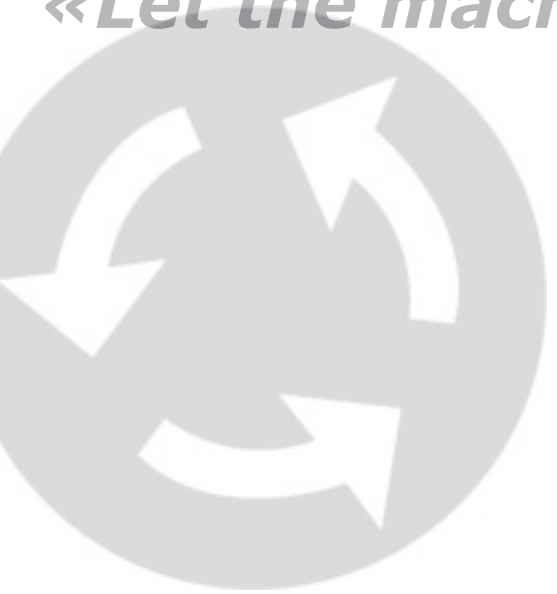
Le même sans commentaire.
Le code est indenté : les blocs d'instructions sont décalés pour améliorer la lisibilité des fonctions imbriquées.

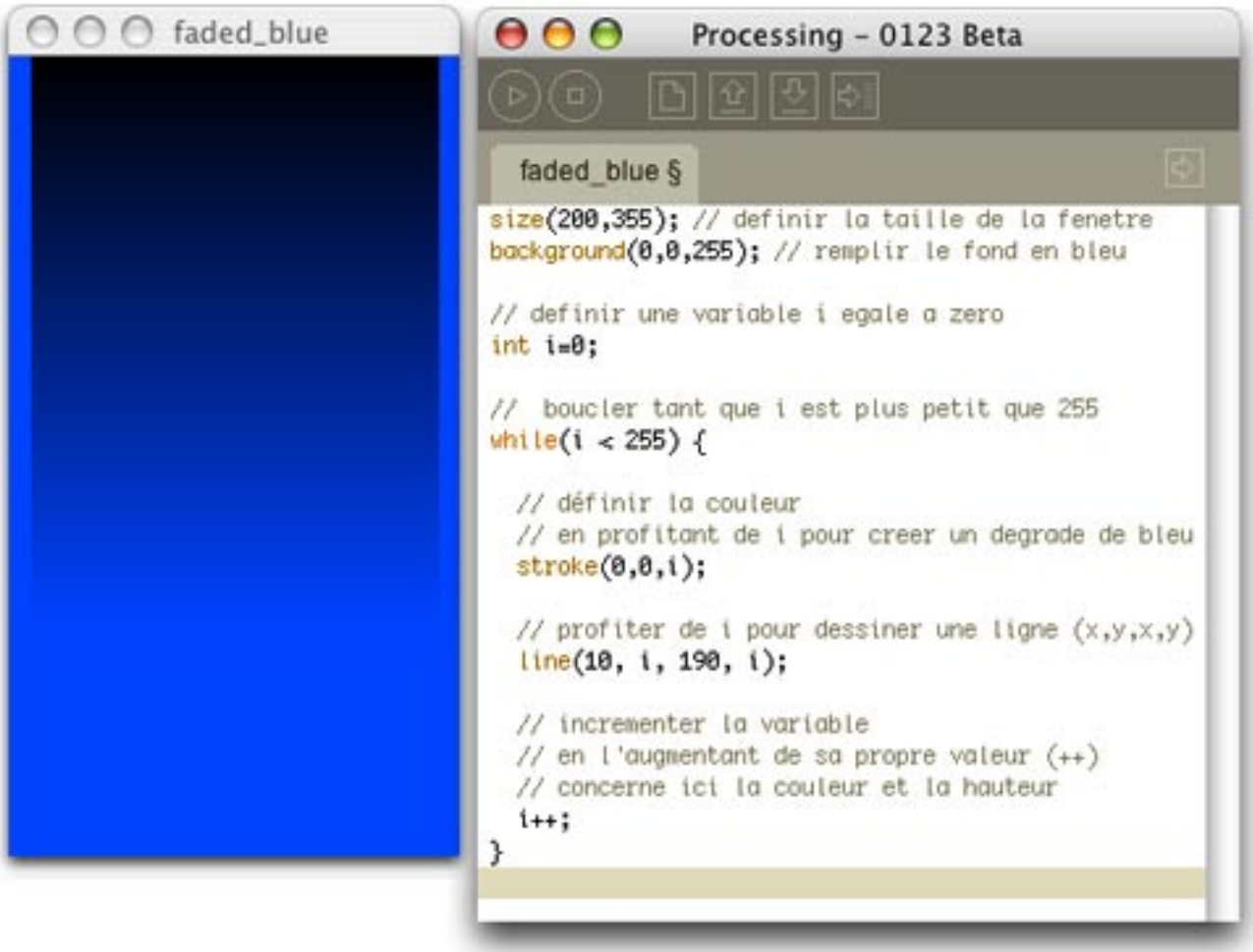
```
size(200, 200);
background(255, 0, 0);
stroke(153);

int l = 30;
int r = width-l;

for(int h = 5; h < width; h += 5) {

  if(h < 50) {
    line( 3*l, h, r-2*l, h );
  }

  else if(h < 150) {
    line( 2*l, h, r-l, h );
  }

  else {
    line( l, h, r, h );
  }
}
```

.une boucle dans Processing

Rappel : le programme va répéter une action
jusqu'à ce qu'une condition remplie
lui demande de passer à autre chose.

«Let the machine do the job !»





Premier exemple : while ()

La boucle while() tourne en rond
tant que ce qui est entre parenthèses reste VRAI.
Ici, dès que la variable i n'est plus inférieure à 256,
l'ordinateur sort de la boucle.

Voir ce programme en ligne.
Télécharger ce programme.

Sur un fond bleu RVB, dessiner un dégradé de bleu foncé vers bleu RVB.
Une variable est définie avec une valeur zéro, puis une boucle est créée pour incrémenter cette valeur jusque 255.
On profite de cette boucle pour définir les coordonnées y d'une ligne (qui du coup va se dessiner en boucle jusqu'à atteindre la valeur 255),
ainsi que la valeur de bleu de la couleur (Rouge,Vert,Bleu),
qui va passer du noir au bleu RVB.

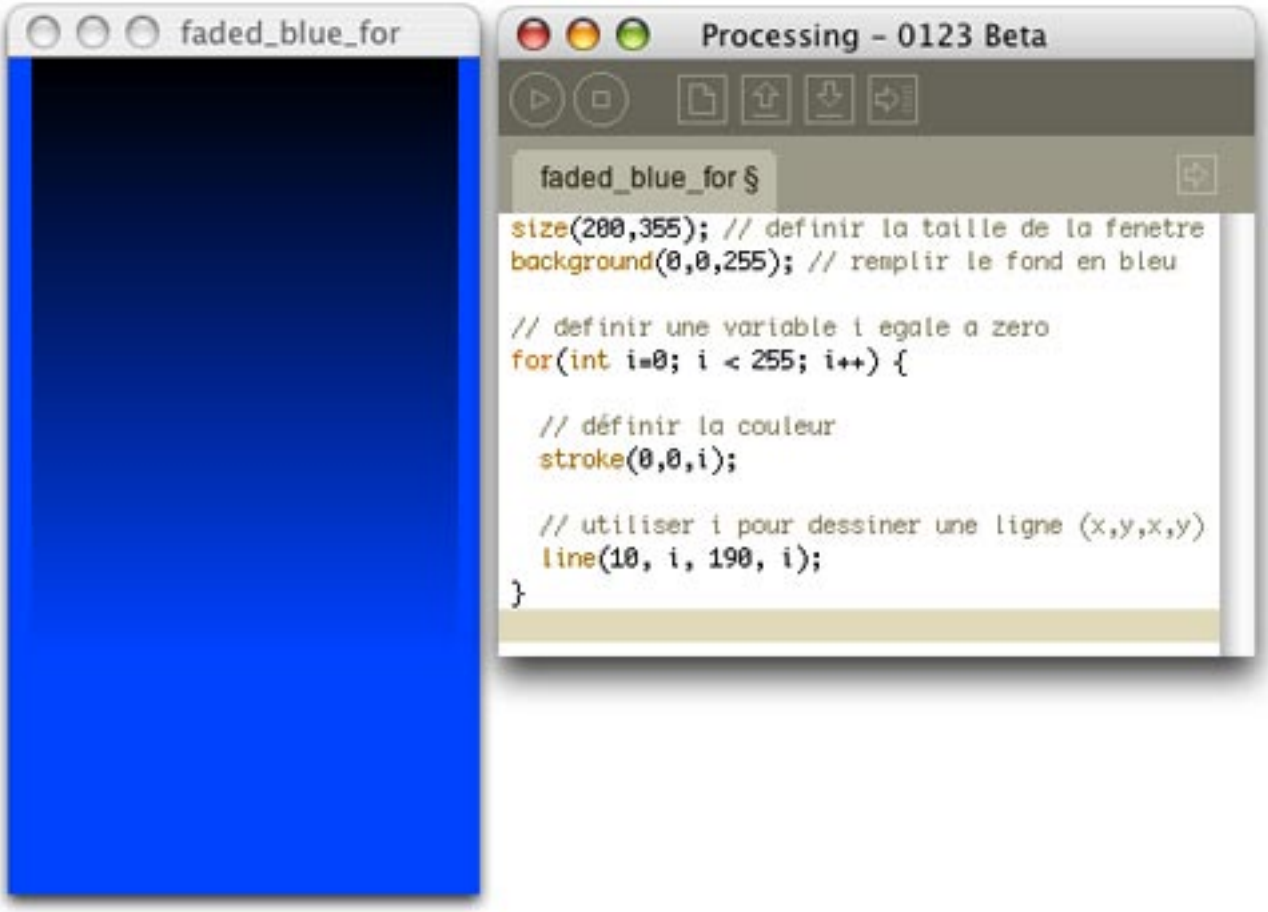
```
size(200,355); // définir la taille de la fenêtre
background(0,0,255); // remplir le fond en bleu

int i=0;  // définir une variable i egale a zero

//  boucler tant que i est plus petit que 255
while( i < 255) {

  // définir la couleur
  // en profitant de i pour créer un dégradé de bleu
  stroke(0,0,i);

  // profiter de i pour dessiner une ligne (x,y,x,y)
  line(10, i, 190, i);

  // incrémenter la variable
  // en l'augmentant de sa propre valeur (++)
  // concerne ici la couleur et la hauteur
  i++;
}
```

Raccourcis MacOSX pour les caractères spéciaux :

[= maj+alt+(
] = maj+alt+)
{ = alt+(
} = alt+)

Le même code sans commentaire.

```
size(200,355);
background(0,0,255);

int i=0;

while( i < 255) {
  stroke(0,0,i);
  line(10, i, 190, i);
  i++;
}
```



Second exemple : **for()**

L'expression for() prend toujours la forme :
for(initialisation**;** condition de la boucle**;** incrémentation**)**
{
instructions
}

La boucle **for()** exécute les actions entre les accolades de façon répétitive.

On définit deux variables puis une structure de condition pour espacer les lignes en hauteur, selon 3 étages de largeurs différentes.

L'expression for() prend toujours la forme :
for(initialisation**;** condition de la boucle**;** incrémentation**)**

L'incrémentation peut prendre différentes formes :
i++ (augmente de 1) **i+=5** (i = i+5)
i-- (diminue de 1) **i-=5** (i = i-5)

{entre accolades : les instructions de la boucle, ce qu'elle doit faire**}**

```
size(200,355); // définir la taille de la fenêtre
background(0,0,255); // remplir le fond en bleu

// définir une variable i égale a zéro
for(int i=0; i < 255; i++) {

  // définir la couleur
  stroke(0,0,i);

  // utiliser i pour dessiner une ligne (x,y,x,y)
  line(10, i, 190, i);
}
```

Raccourcis MacOSX pour les caractères spéciaux :

[= maj+alt+(
] = maj+alt+)
{ = alt+(
} = alt+)

Le même code sans commentaire.

```
size(200,355);
background(0,0,255);

for(int i=0; i < 255; i++) {

  stroke(0,0,i);
  line(10, i, 190, i);
}
```

.une fonction dans Processing

Rappel :
en donnant un nom à un bloc d'instructions, nous pouvons définir un sous-programme, appelé **fonction** ou **méthode**.

Processing - 0098 Beta

interactif_mouse_pressed_com §

```
// void précise que setup() ne retourne aucune valeur
// setup() definit les proprietes initiales du programme
void setup() {

  size(200, 500);
  // definit la taille de la fenetre

  noLoop();
  // empeche l'instruction draw() de s'executer en boucle

  background(51);
  //definit la couleur de fond

  stroke (random(255),random(255),random(255));
  // definit la couleur du trait au hasard
}

float x;
// declare une variable x (nombre à virgule)

// definit une fonction de dessin qui ne retourne aucune valeur
void draw() {

  // += combine une addition à un assignement de valeur
  // donc : une valeur de 0,05 sera additionnée à la variable x
  x += 0.05;

  // en dessinant une ligne de haut en bas
  line(x, 0, x, height);

  // mais si x depasse la largeur de la fenetre
  if (x > width) {
  // x reprend la valeur de 0 (donc revient au debut)
    x = 0;
    stroke (random(255),random(255),random(255));
  }

}
// souris pressee = declenche la boucle de dessin
void mousePressed() {
  stroke (random(255),random(255),random(255));
  loop();
}
// souris lachee = stoppe la boucle de dessin
void mouseReleased() {
  stroke (random(255),random(255),random(255));
  noLoop();
}
```

Voir ce programme en ligne.

Télécharger ce programme.

Le programme va tracer des lignes verticales
lorsqu'on presse la souris
et s'arrêter lorsqu'on lâche la souris.
Si l'accumulation de lignes verticales dépasse la largeur,
il recommence au début.

```
void setup() {
// void précise que setup() ne retourne aucune valeur
// setup() définit les propriétés initiales du programme

  size(200, 500);   // définit la taille de la fenêtre
  noLoop();   // empêche l'instruction draw() de s'exécuter en boucle
  background(51);   // définit la couleur de fond
  stroke (random(255),random(255),random(255));   // couleur du trait au hasard
}

float x;  // déclare une variable x (nombre à virgule)

void draw() {  // définit une fonction de dessin

  // += combine une addition à un assignement de valeur
  // donc : une valeur de 0,05 sera additionnée à la variable x
  x += 0.05;

  // en dessinant une ligne de haut en bas
  line(x, 0, x, height);

  // mais si x dépasse la largeur de la fenêtre
  if (x > width) {
  // x reprend la valeur de 0 (donc revient au début)
    x = 0;
    stroke (random(255),random(255),random(255));
  }

}
souris pressée = déclenche la boucle de dessin :
void mousePressed() {
  stroke (random(255),random(255),random(255));
  loop();
}
souris lachée = stoppe la boucle de dessin
void mouseReleased() {
  stroke (random(255),random(255),random(255));
  noLoop();
}
```

Le même code sans commentaire.
Le code est indenté : les blocs d'instructions sont décalés
pour améliorer la lisibilité des fonctions imbriquées.

```
void setup() {

  size(200, 500);
  noLoop();
  background(51);
  stroke (random(255),random(255),random(255));
}

float x;

void draw() {
  x += 0.05;
  line(x, 0, x, height);

  if (x > width) {
    x = 0;
    stroke (random(255),random(255),random(255));
  }

}

void mousePressed() {
  stroke (random(255),random(255),random(255));
  loop();
}

void mouseReleased() {
  stroke (random(255),random(255),random(255));
  noLoop();
}
```

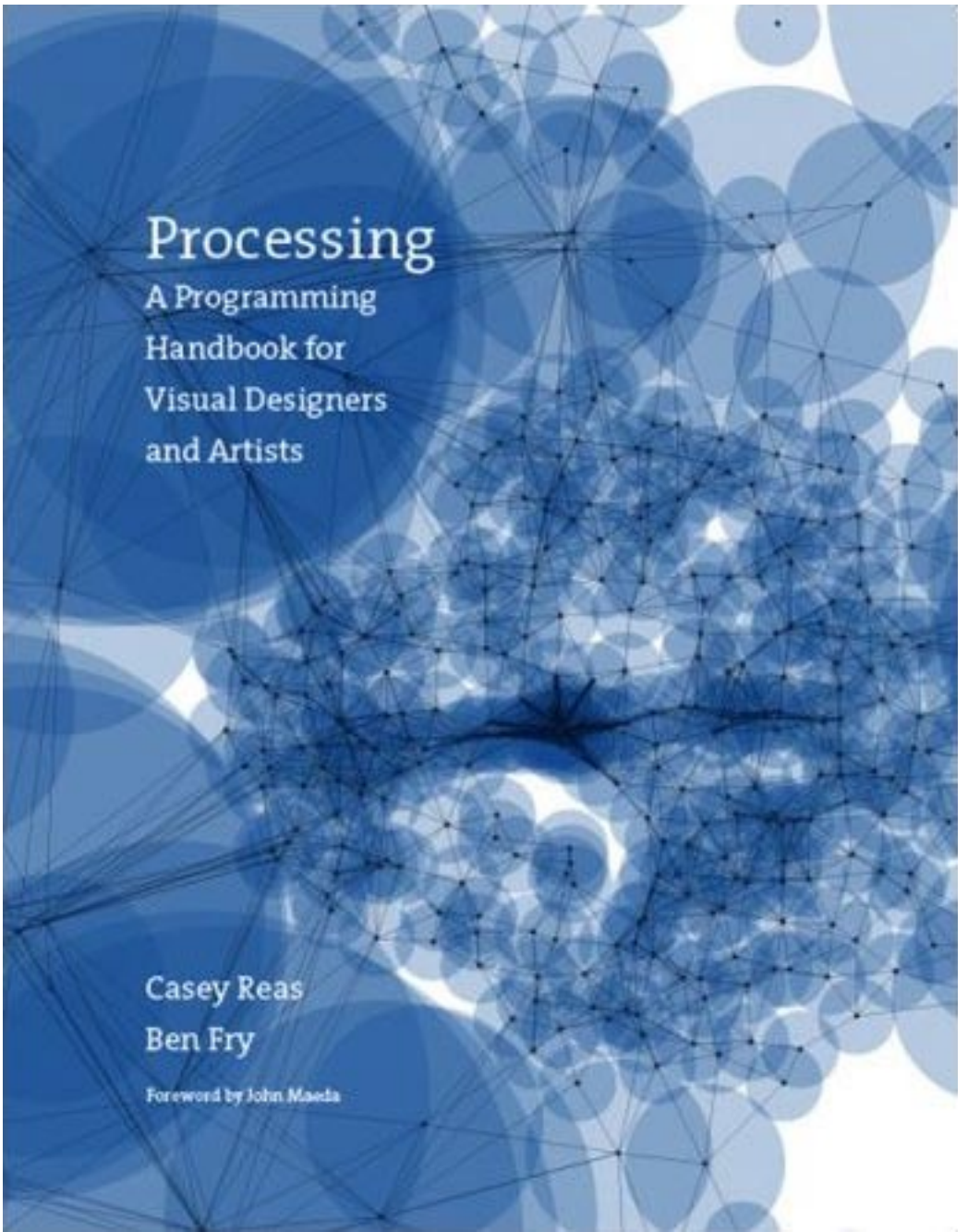
Liens :

Accès en ligne à la page d’exemples.

.annexes 01 : pas à pas vers une animation.

Exemples extraits de «*Processing : a programming handbook for visual designers and artists*» disponible à la bibliothèque.

Un échantillon utile de ce livre, ainsi que ses exemples de code sont téléchargeables ici : http://processing.org/learning/books/



Un programme PROCESSING peut contenir une seule ligne de code ou des milliers. Cette évolutivité est l'un des aspects les plus importants du langage.

L'exemple qui suit présente étape par étape le modeste projet d'animer une séquence de lignes diagonales, de manière à explorer certains des composants de base du langage PROCESSING.

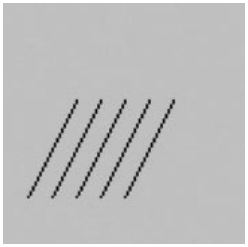
Les termes et symboles utilisés ici ne seront probablement pas familiers aux novices.

Ce pas-à-pas est un condensé de notions développées en détail
dans le livre «*Processing : a programming handbook for visual designers and artists*».

Nous allons expérimenter ces petits programmes dans l'environnement de programmation PROCESSING afin de comprendre comment le code réagit.

PROCESSING a été conçu pour faciliter le dessin d'éléments graphiques tels que des lignes, des ellipses, des courbes dans la fenêtre d'affichage. Ces formes sont positionnées grâce à des chiffres qui définit leurs coordonnées.

La position d'une ligne est définie par quatre chiffres, deux pour chaque bout. Les paramètres utilisés à l'intérieur de la fonction **line()** détermine l'endroit où la ligne sera tracée. Le point d'origine d'un système de coordonnées part du coin supérieur gauche, et les valeurs augmentent vers la droite et vers le bas.



```
line(10, 80, 30, 40);  // ligne de gauche
line(20, 80, 40, 40);
line(30, 80, 50, 40);  // ligne du milieu
line(40, 80, 60, 40);
line(50, 80, 70, 40);  // ligne de droite
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

Les caractéristiques visuelles des formes sont contrôlées
grâce à d'autres éléments codés qui fixent les valeurs de couleur ou de gris,
la largeur des lignes et la qualité du rendu.

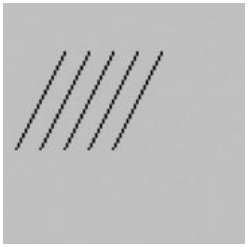
```
background(0);          // Définir la couleur de fond : noir
stroke(255);            // La couleur de la ligne : blanc
strokeWeight(5);        // Largeur de la ligne : 5 pixels
smooth();               // Lisse le bord des lignes

line(10, 80, 30, 40);   // ligne de gauche
line(20, 80, 40, 40);
line(30, 80, 50, 40);   // ligne du milieu
line(40, 80, 60, 40);
line(50, 80, 70, 40);   // ligne de droite
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

Une variable telle que **x**, représente une valeur.
Cette valeur remplace le symbole **x** lorsque le code est exécuté.

Une variable peut alors contrôler de nombreuses fonctions du programme.



```
int x = 5;   // Définir une valeur de position horizontale
int y = 60;  // Définir une valeur de position verticale

line(x, y, x+20, y-40);    // Ligne de [5,60] à [25,20]
line(x+10, y, x+30, y-40); // Ligne de [15,60] à [35,20]
line(x+20, y, x+40, y-40); // Ligne de [25,60] à [45,20]
line(x+30, y, x+50, y-40); // Ligne de [35,60] à [55,20]
line(x+40, y, x+60, y-40); // Ligne de [45,60] à [65,20]
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

Ajouter davantage de structure de contrôle à un programme ouvre de nouvelles possibilités. Les fonctions **setup()** et **draw()** permettent au programme de tourner continuellement, ce qui est nécessaire pour créer des animations et des programmes interactifs.

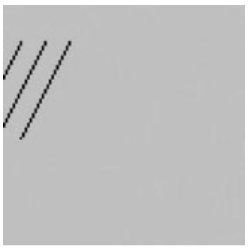
Le code contenu dans **setup()** tourne une seule fois, lorsque le programme démarre, et le code contenu à l'intérieur de **draw()** tourne continuellement.
Un état de l'image («frame») est dessiné dans la fenêtre d'affichage à la fin de chaque boucle traversant la fonction **draw()**.

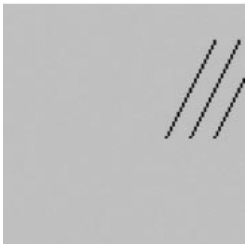
Dans l'exemple suivant, la variable **x** est déclarée comme une variable globale, c'est à dire qu'elle peut être affecté et accessible n'importe où dans le programme.
La valeur de **x** augmente de 1 à chaque «frame», et parce que la position des lignes est contrôlé par **x**, elles sont déplacées à un endroit différent à chaque fois que la valeur change, ce qui déplace les lignes vers la droite.

La ligne 14 dans le code est une structure de contrôle «**if**». Il contient une expression comparant la variable **x** à la valeur 100.

Si l'expression est vraie, le code inclus dans le bloc (entre les accolades { et } associé à la structure de contrôle «**if**») est exécuté.

Si l'expression est fausse, le code inclus dans ce bloc refuse de se lancer.
Lorsque la valeur de x devient supérieur à 100, la ligne de code dans le bloc définit la variable **x** à -40, ce qui provoque le recul des lignes au bord gauche de la fenêtre.





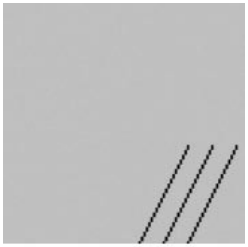
```
int x = 0;   // Définir une valeur de position horizontale
int y = 55;  // Définir une valeur de position verticale

void setup() {
   size(100, 100);  // fenetre a 100 x 100 pixels
}

void draw() {
   background(204);
   line(x, y, x+20, y-40);     // Ligne de gauche
   line(x+10, y, x+30, y-40);  // Ligne du milieu
   line(x+20, y, x+40, y-40);  // Ligne de droite
   x = x + 1;      // Ajouter 1 à x
   if (x > 100) {  // Si x est plus grand que 100
     x = -40;      // assigner -40 à x
   }
}
```

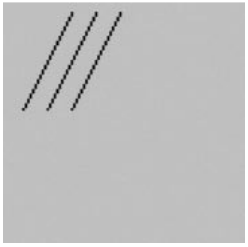
Voir
ce programme
en ligne.

Télécharger
ce programme.

Quand un programme tourne en permanence, PROCESSING stocke des données provenant des périphériques d'entrée tels que la souris et le clavier.

Ces données peuvent être utilisées pour influer sur ce qui se passe dans la fenêtre d'affichage.





```
void setup() {
  size(100, 100);
}

void draw() {
  background(204);
  // Assigner la valeur horizontale du curseur de la souris à x
  float x = mouseX;
  // Assigner la valeur verticale du curseur de la souris à y
  float y = mouseY;
  line(x, y, x+20, y-40);
  line(x+10, y, x+30, y-40);
  line(x+20, y, x+40, y-40);
}
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

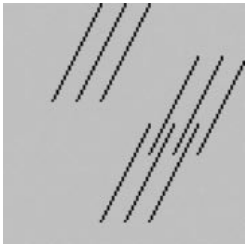
Une fonction est un ensemble de code au sein d'un programme qui effectue une tâche spécifique.

Les fonctions sont de puissants outils de programmation qui rendent les programmes plus facile à lire et à modifier.

La fonction **diagonales()** de l'exemple qui suit a été créée pour tracer une série de trois lignes diagonales à chaque fois qu'elle se lance dans **draw()**.

Deux paramètres, les chiffres entre parenthèses après le nom de la fonction, définissent la position des lignes.

Ces chiffres sont passés dans la définition de la fonction en ligne 12 et sont utilisés comme des valeurs pour les variables **x** et **y** dans les lignes 13 à 15.



```
void setup() {
  size(100, 100);
  noLoop();
}

void draw() {
  diagonals(40, 90);
  diagonals(60, 62);
  diagonals(20, 40);
}

void diagonals(int x, int y) {
  line(x, y, x+20, y-40);
  line(x+10, y, x+30, y-40);
  line(x+20, y, x+40, y-40);
}
```

Voir
ce programme
en ligne.

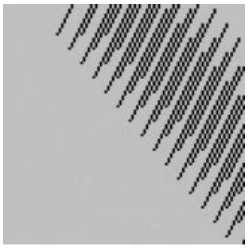
Télécharger
ce programme.

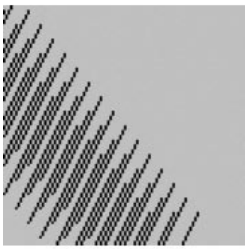
Les variables utilisées dans les programmes précédents stockent chacune une seule donnée.

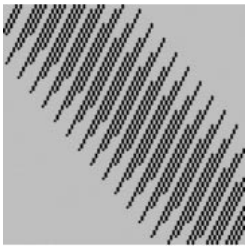
Si nous voulons avoir 20 groupes de lignes à l'écran, il faudra de 40 variables : 20 pour les positions horizontales et 20 pour les positions verticales.

Ceci rend la programmation fastidieuse et peut rendre la lecture des programmes difficile. Au lieu d'utiliser de multiples noms de variables, on peut utiliser des tableaux (**array**).

Un tableau permet de stocker une liste d'éléments de données sous un seul nom. Une structure de contrôle «**for**» peut être utilisée pour parcourir dans l'ordre chaque élément du tableau.







```
int num = 20;
int[] dx = new int[num];  // Declare et cree un tableau
int[] dy = new int[num];  // Declare et cree un tableau

void setup() {
  size(100, 100);
  for (int i = 0; i < num; i++) {
    dx[i] = i * 5;
    dy[i] = 12 + (i * 6);
  }
}

void draw() {
  background(204);
  for (int i = 0; i < num; i++) {
    dx[i] = dx[i] + 1;
    if (dx[i] > 100) {
      dx[i] = -100;
    }
    diagonals(dx[i], dy[i]);
  }
}
void diagonals(int x, int y) {
  line(x, y, x+20, y-40);
  line(x+10, y, x+30, y-40);
  line(x+20, y, x+40, y-40);
}
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

La **programmation orientée objet** est une manière de structurer le code en *objets*, c'est-à-dire en blocs de code distincts qui contiennent à la fois des données et des fonctions.
Cette approche de la programmation permet une liaison étroite entre les groupes de données et les fonctions qui agissent sur ces données.
La fonction **diagonales()** peut être étendue en l'associant à la définition d'une classe.

Les objets sont construits en utilisant la classe comme un moule ou un gabarit.
Les variables servant à positionner les lignes et définissant les attributs du dessin sont appliquées à l'intérieur de la classe définie. Voir le site Processing à ce sujet.

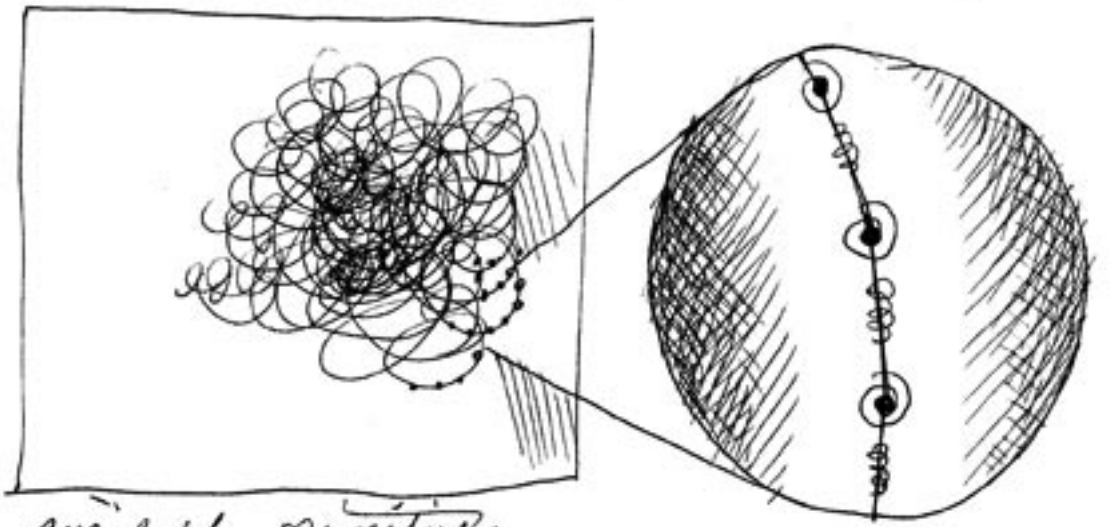
```
Diagonals da, db;

void setup() {
  size(100, 100);
  smooth();
  // Entree : x, y, speed, thick, gray
  da = new Diagonals(0, 80, 1, 2, 0); // L'operateur «new»
  db = new Diagonals(0, 55, 2, 6, 255);
}
void draw() {
  background(204);
  da.update();
  db.update();
}

class Diagonals {
  int x, y, speed, thick, gray; // Attributs de l'objet

// Le constructeur permet de créer un objet de la classe
// Il initialise l'objet, il est execute par l'operateur «new»
  Diagonals(int xpos, int ypos, int s, int t, int g) {
    x = xpos;
    y = ypos;
    speed = s;
    thick = t;
    gray = g;
  }

  void update() {
    // Comportement de l'objet defini par une methode
    strokeWeight(thick);
    stroke(gray);
    line(x, y, x+20, y-40);
    line(x+10, y, x+30, y-40);
    line(x+20, y, x+40, y-40);
    x = x + speed;
    if (x > 100) {
      x = -100;
    }
  }
}
```

Voir
ce programme
en ligne.

Télécharger
ce programme.

.annexes 02 : schémas divers.

floccus / aurora as musical instrument 12/27/99

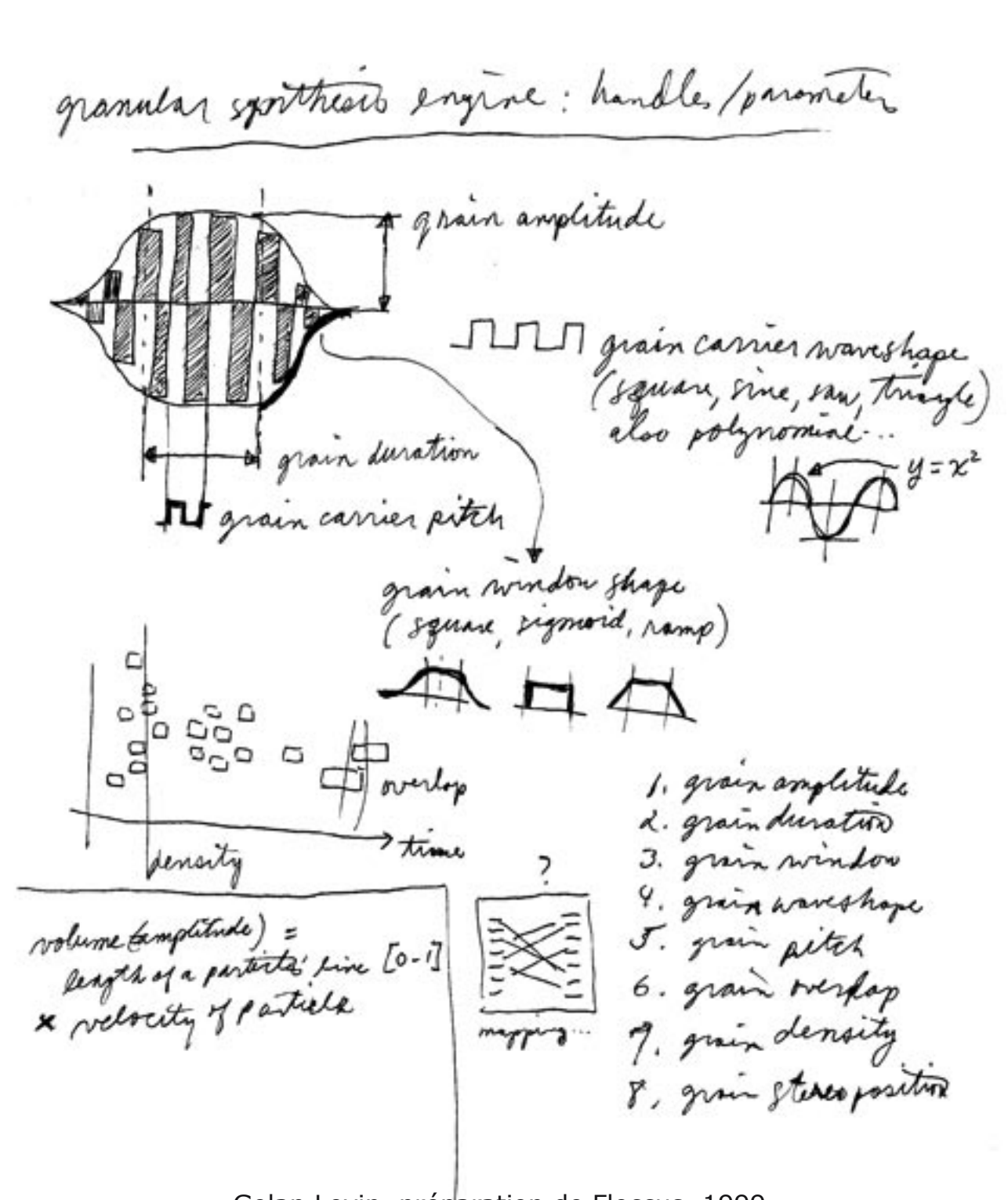


available primitives:

1. absolute left/right position of particle
2. linear - spring tension on particle (stretch/squash) } forces
3. angular - spring tension on particle (pinch/splay) }
4. velocity (magnitude) of particle
5. velocity (orientation) of particle (less interesting) - heading
6. "color" of particle - red/green/blue (1 of 3)
7. length (original) of a particle's line
8. ordinality/position of a particle along its line
9. distance from particle to cursor
10. screen brightness (if any) at particle's location - roughly equal to ~~distance from~~ closeness to other particles.

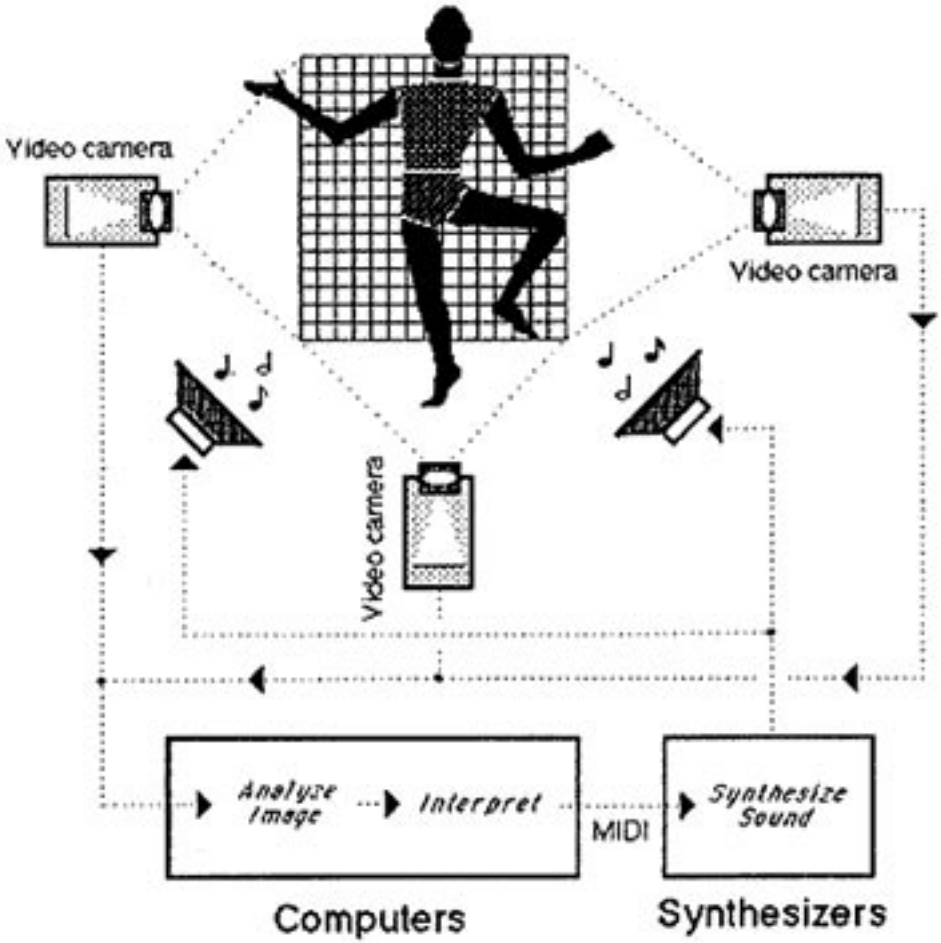
Golan Levin, préparation de Floccus, 1999.

http://flickr.com/photos/golanlevin/



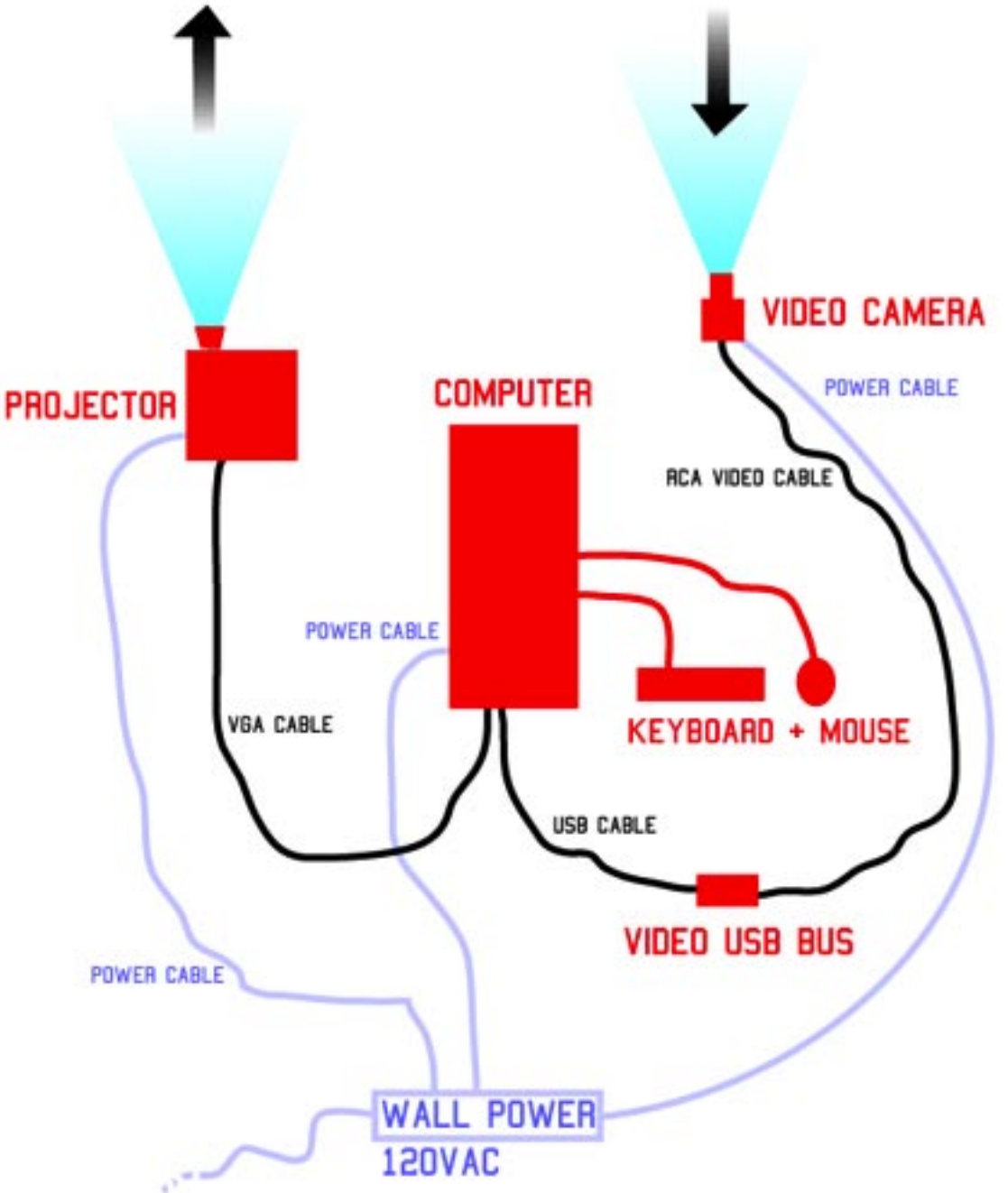
Golan Levin, préparation de Floccus, 1999.

http://flickr.com/photos/golanlevin/



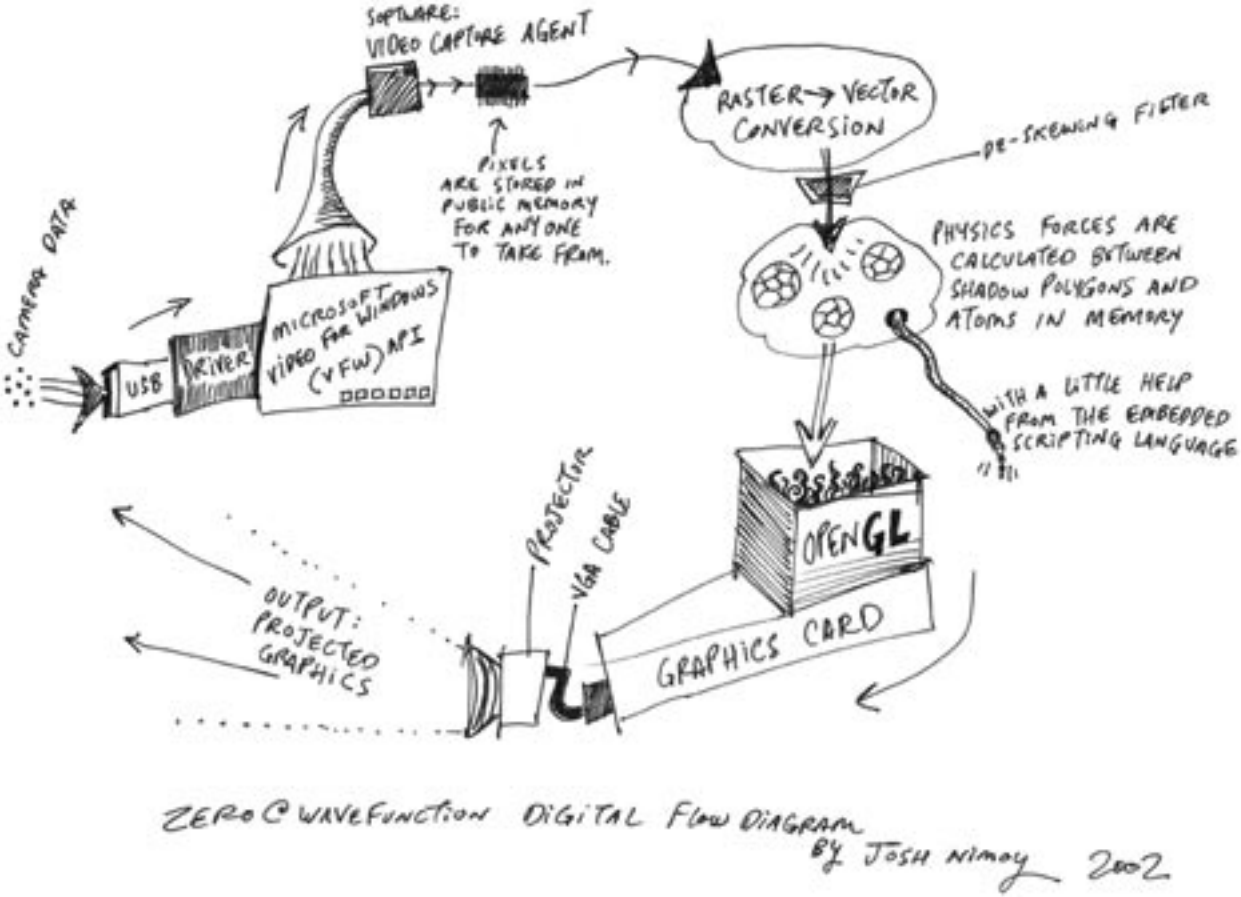
David Rokeby : softVNS (1986-1990),
logiciel et environnement interactif.

http://homepage.mac.com/davidrokeby/vns.html



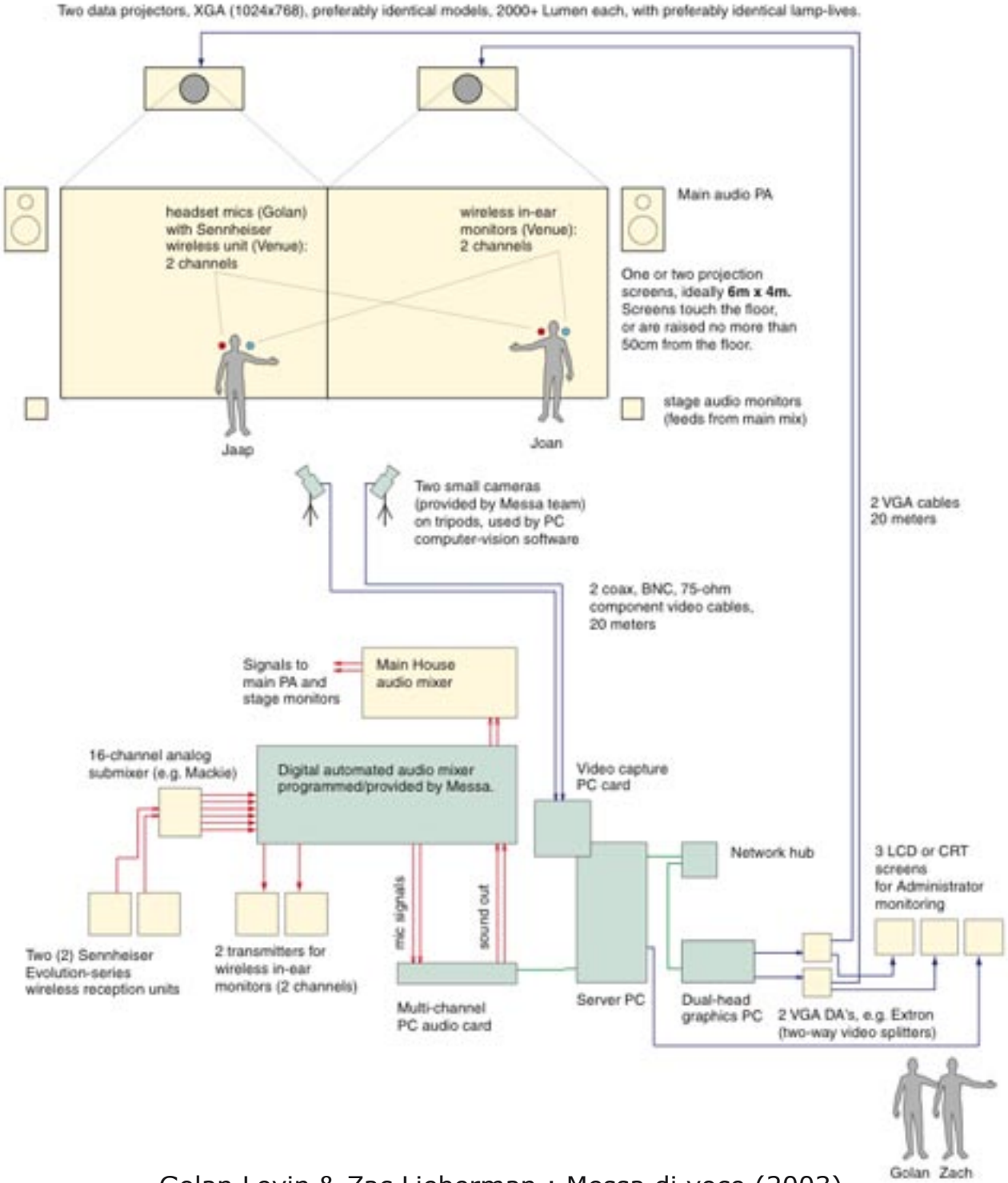
Josh Nimoy : Zero@wavefunction (1998),
logiciel et environnement interactif.

http://www.jtnimoy.com/zerowave/



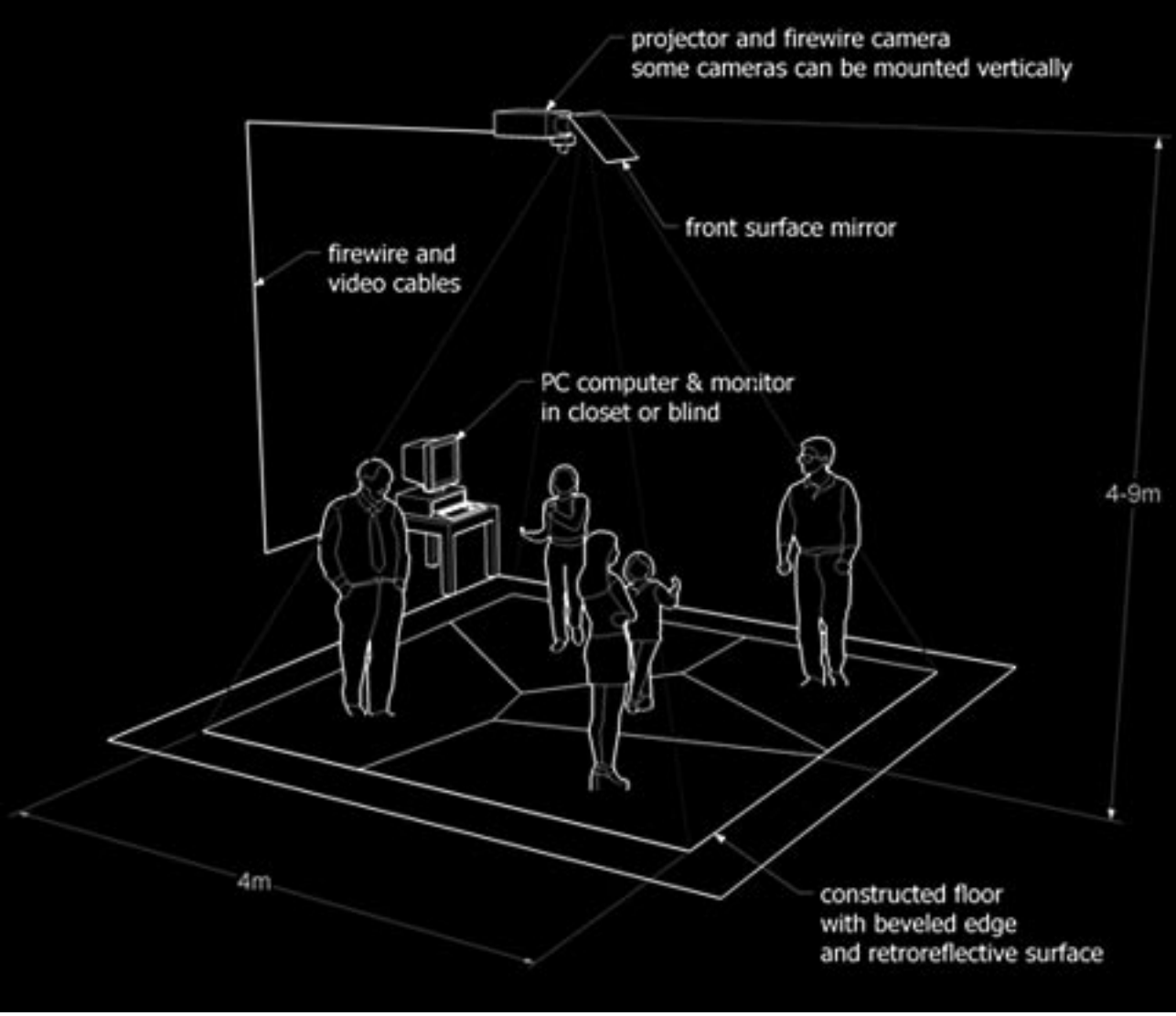
Josh Nimoy : Zero@wavefunction (1998),
logiciel et environnement interactif.

http://www.jtnimoy.com/zerowave/



Golan Levin & Zac Lieberman : Messa di voce (2003),
logiciel, installation et performance interactive.

http://www.tmema.org/messa/



Scott Snibbe : Boundary functions (1998),
logiciel et installation interactive.

http://www.snibbe.com/

.glossaire

adresse : valeur numérique désignant un emplacement en mémoire.

adressage : le mécanisme de calcul des adresses.

affecter : donner une nouvelle valeur à la mémoire se trouvant à une certaine adresse (pouvant être représentée par une variable). Le nom associé est l'affectation. En général : attribuer des ressources.

assembleur : le langage assembleur ou langage d'assemblage, est le langage de programmation lisible pour un humain le plus proche du langage machine utilisé par le microprocesseur de la machine.

boucle : portion de code accompagnée de sa structure de contrôle destinée à être exécutée plusieurs fois (la structure de contrôle relançant l'exécution du code depuis le début tant qu'une condition n'est pas remplie). Les mots-clés associés les plus classiques sont **for** et **while**.

constante : variable (!) dont la valeur ne change pas au cours de l'exécution du programme, c'est simplement un nombre, un caractère, ou une suite de caractères.

donnée(s) : information(s) de nature numérique, représentée(s) sous forme codée en vue d'y être enregistrée(s), traitée(s), conservée(s) et communiquée(s) et compréhensible(s) par la seule machine. **Data** est bien un pluriel et garde toujours ce sens.

instruction : commande élémentaire interprétée et exécutée par le processeur.

itération : séquence d'instructions (ou exécution de la séquence) destinée à être exécutée plusieurs fois (autant de fois qu'on peut en avoir besoin).

itératif : caractérise ce qui se produit plusieurs fois.

génératif : se dit d'un système capable de produire, de manière autonome, des valeurs aléatoires en fonction d'une série de règles prédéfinies.

mémoire : ce terme désigne d'une façon générale tout ce qui peut stocker de l'information. Utilisé seul («la mémoire»), ce terme représente la mémoire centrale d'un ordinateur. Les mémoires de masses peuvent stocker de grandes quantités de données, en général à moyen ou long terme. Les mémoires vives s'effacent quand elles ne sont plus sous tension, tandis que les mémoires mortes gardent les informations mais ne peuvent être que lues (on ne peut pas écrire dessus, sauf dans certains cas particuliers).

opérande : elément sur lequel on effectue une opération. Un opérande peut être une constante ou une variable. Par exemple, dans « 2 + 2 », les opérandes sont les deux deux.

opérateur booléen : opérateurs (et, ou, non, etc) permettant d'effectuer des opérations sur des valeurs binaires. Ces opérateurs sont très utilisés en informatique, par exemple pour faire des tests (si (a ou b) est vrai, alors...).

primitive : en programmation : instruction ou commande de base dans un langage de programmation. En graphisme : élément graphique de base, comme une droite, un plan, une sphère, à partir desquels on peut construire des objets plus complexes.

variable : emplacement en mémoire stockant une donnée pouvant varier au cours de l'exécution d'un programme (par opposition aux constantes). Une variable peut avoir un type, définissant a priori la nature de son contenu.