Algorithmique et programmation

Samuel Toubon
Département d'enseignement informatique
Support de cours conçu en collaboration avec Laurence Duval (Université de Bretagne Occidentale)

septembre 2016

Avertissement

Ce support de cours est commun à tous les étudiants de première année.

- Pour les étudiants issus de prépa maths et de DUT STID, déjà familiers avec la programmation, il constitue une mise à niveau en langage Java. Ce langage sera utilisé durant la suite de votre scolarité à l'Ensai. Ce support de cours n'est qu'un préambule au cours de *complexité et calculabilité*.
- Pour les étudiants internes et issus du concours éco, il constitue la base d'un enseignement plus long, destiné à acquérir de solides connaissances en algorithmique et programmation dans le cadre du cours du même nom.

Présentation de l'enseignement

- Contenu
- Volume
- Évaluation

Agenda

Rôle du support de cours

- C'est un support pour le cours en amphi...
- ...et non un document à lire soi-même.
- En amphi, sont donnés des exemples, des explications, des résultats et des démonstrations indispensables.

Plan du cours

Algorithmique ? Programmation ?

- **Algorithme** : description, en un nombre **fini** d'étapes, d'un enchaînement d'opérations élémentaires, permettant d'effectuer un calcul.
- **Programme** : algorithme codé dans un langage donné exécutable par un ordinateur. Un programme **implémente** un algorithme.
- **Implémenter** : « passer au niveau en dessous ».
  Maths $\rightarrow$ Algorithme $\rightarrow$ Programme

Objectifs

- Être capable de mettre en œuvre des algorithmes simples.
- Maîtriser les principales structures algorithmiques.
- Savoir adopter une démarche d'identification de l'origine d'un problème dans un algorithme.

Le langage de l'ordinateur

1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1
1 1 1 1 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 1 1
0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 1 1 1 0
1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1
0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1
1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 1
1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 1 0 1 0 1 1 0 0
0 0 1 1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 1
0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 1 0
0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0
0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 1 1
0 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 0
1 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0

Quantités d'information

- Bit : 1 ou 0.
- Octet : 8 bits 01100010. 1 Octet $\approx$ 1 Byte.
- K
  - 1 kibioctet (Kio) = 1024 (2^{10}) octets
  - 1 kilooctet (Ko) = 1000 (10^3) octets
- M
  - 1 mébioctet (Mio) = 1024^2 (2^{20}) octets
  - 1 mégaoctet (Mo) = 1000^2 (10^6) octets
- G
  - 1 gibioctet (Gio) = 1024^3 (2^{30}) octets
  - 1 gigaoctet (Go) = 1000^3 (10^9) octets
- Téra $\leftrightarrow$ Tébi
- Péta $\leftrightarrow$ Pébi

Florence Porcel @FlorencePorcel · 41 min

Si on utilisait plus souvent les unités des préfixes binaires, la vie serait plus drôle. (JE TRAVAILLE.)

Unités de bits v · d · m

Ordre de grandeur	Système international (SI)			Préfixes binaires		
	Unité	Notation	Valeur	Unité	Notation	Valeur
1	**bit**	**bit**	1 bit	**bit**	**bit**	1 bit
10^3	kilobit	kbit	10^3 bits	kibibit	Kibit	2^{10} bits
10^6	mégabit	Mbit	10^6 bits	mébibit	Mibit	2^{20} bits
10^9	gigabit	Gbit	10^9 bits	gibibit	Gibit	2^{30} bits
10^{12}	térabit	Tbit	10^{12} bits	tébibit	Tibit	2^{40} bits
10^{15}	pétabit	Pbit	10^{15} bits	pébibit	Pibit	2^{50} bits
10^{18}	exabit	Ebit	10^{18} bits	exbibit	Eibit	2^{60} bits
10^{21}	zettabit	Zbit	10^{21} bits	zébibit	Zibit	2^{70} bits
10^{24}	yottabit	Ybit	10^{24} bits	yobibit	Yibit	2^{80} bits

18 21

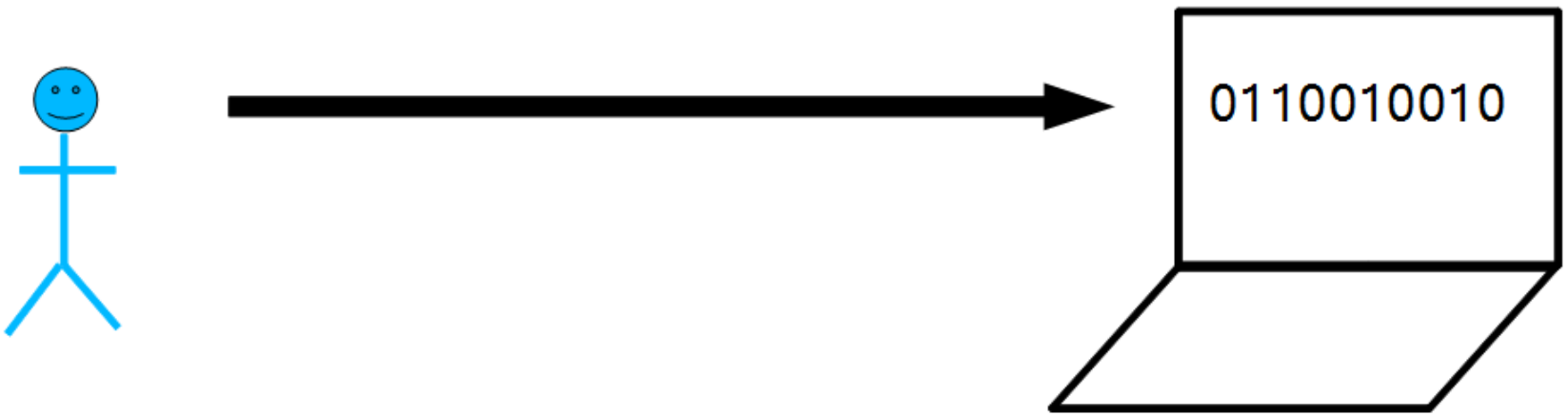
Codage des caractères

- Exemple : ASCII.
  - A : 10000001
  - B : 10000010
  - C : 10000011

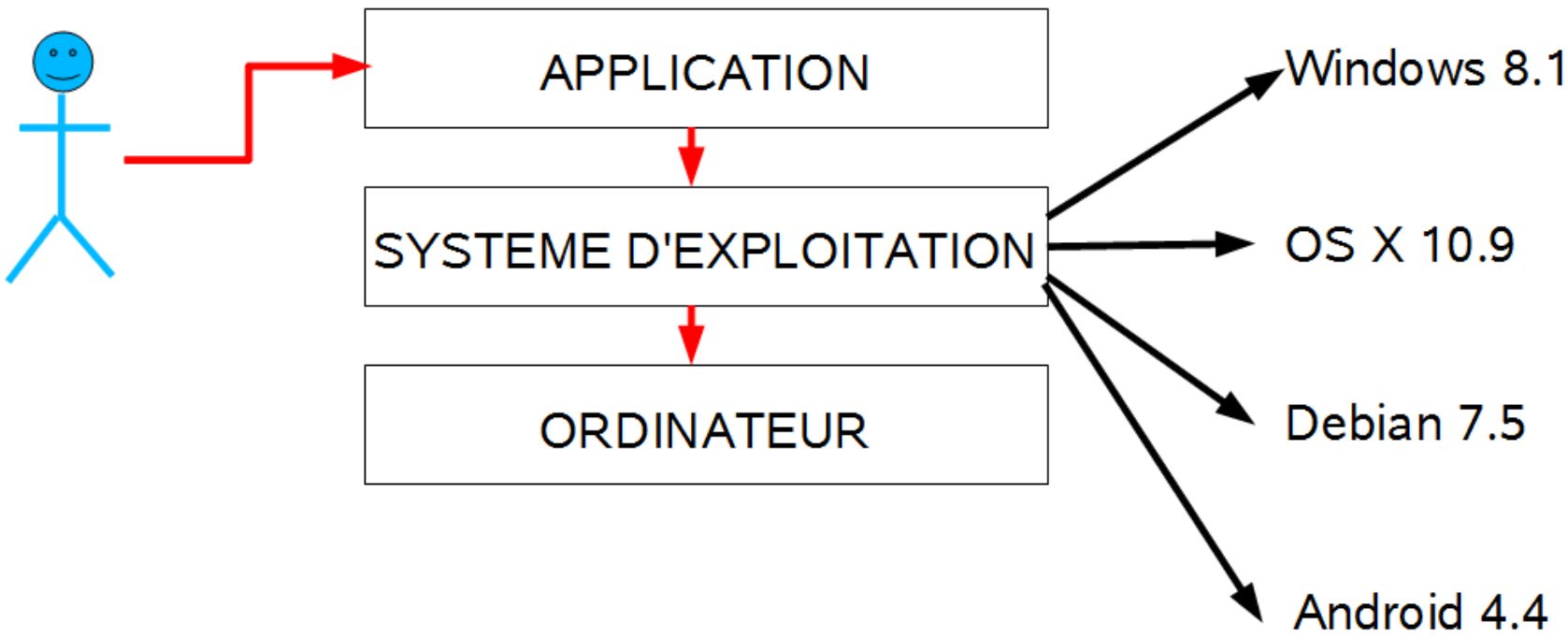
Codage des opérations

- Opération codée sur 3 bits
  - 001 : addition
  - 010 : soustraction
  - ...
- Nombre codé sur un octet (8 bits)
- 001 00100010 00010111 $\rightarrow$ 00111001
- 010 00001101 00001001 $\rightarrow$ 00000100

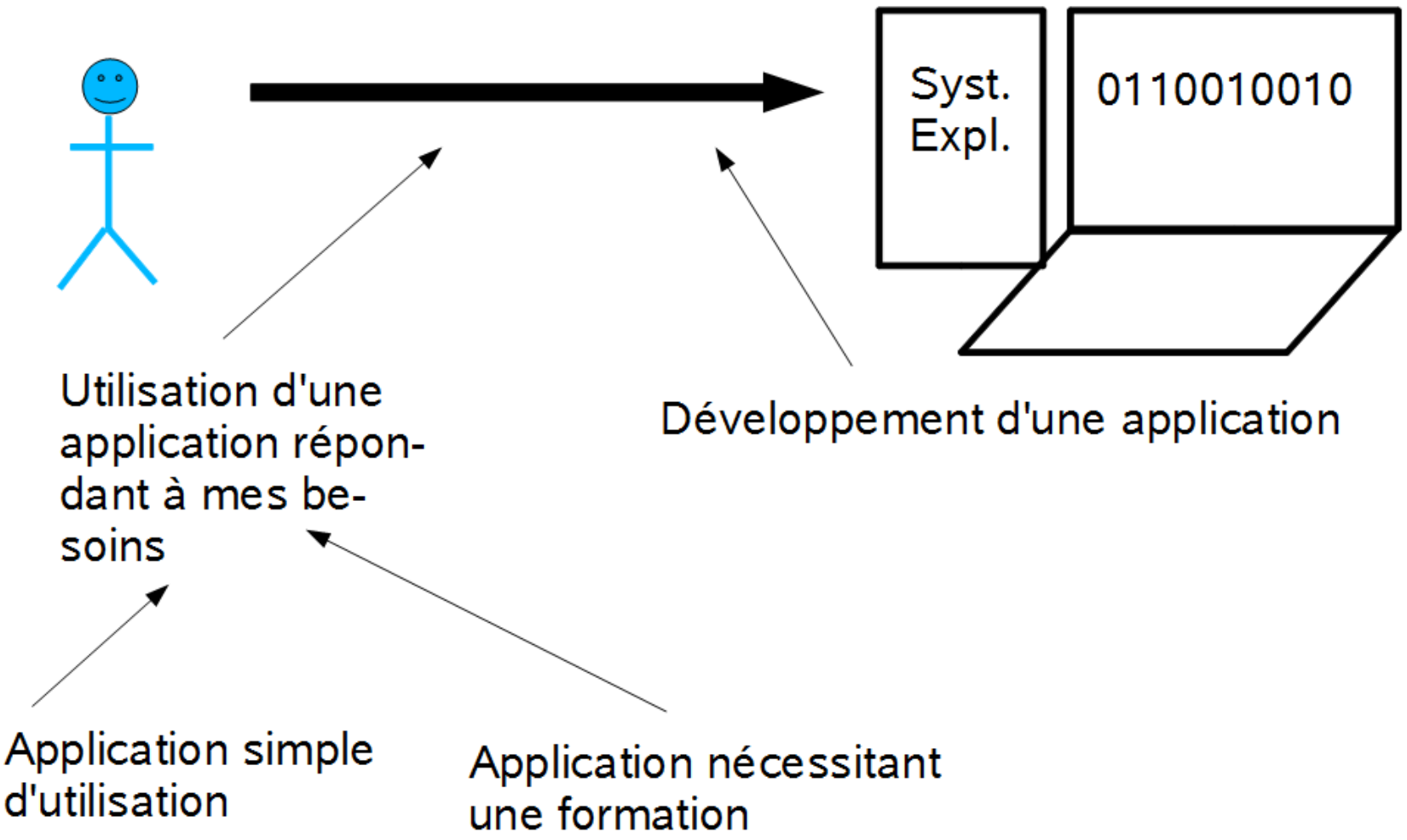
Comment communiquer avec un ordinateur ?



Système d'exploitation



Quelle application utiliser ?



Quelle application utiliser ?

- Application existante répondant au besoin
  - Traitement de texte : LibreOffice, Word...
  - Navigation sur Internet : Firefox, Chrome...
  - Statistique : SAS, R, SPSS, SPAD...
- Développement d'une application
  - Externaliser
    - Réaliser un cahier des charges
    - ...
  - Réaliser soi-même
    - Choisir le langage de programmation
    - Choisir le mode de stockage des données
    - ...

Langage de programmation ?

- Programmation objet
  - Java
  - C++
  - Python
  - PHP
- **Programmation impérative**
  - Python
  - C
  - PHP
- Programmation fonctionnelle
  - Lisp
  - Scheme

Pourquoi autant de langages ?

https ://www.college-de-france.fr/site/gerard-berry/course-2015-11-04-16h00.htm

▸ Lien

18’58

Programmation impérative

Programme = suite d'instructions.

- Ordre d'exécution des instructions défini par le programmeur.

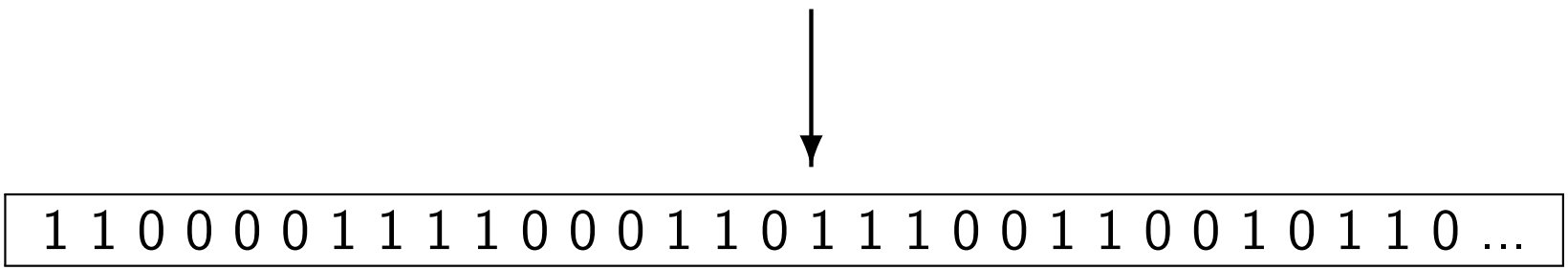
```
1  fCM = 220 - age;
2  fCE = 0.8 * fCM;
```

- Exemple : fonction calculant la somme des n premiers entiers.

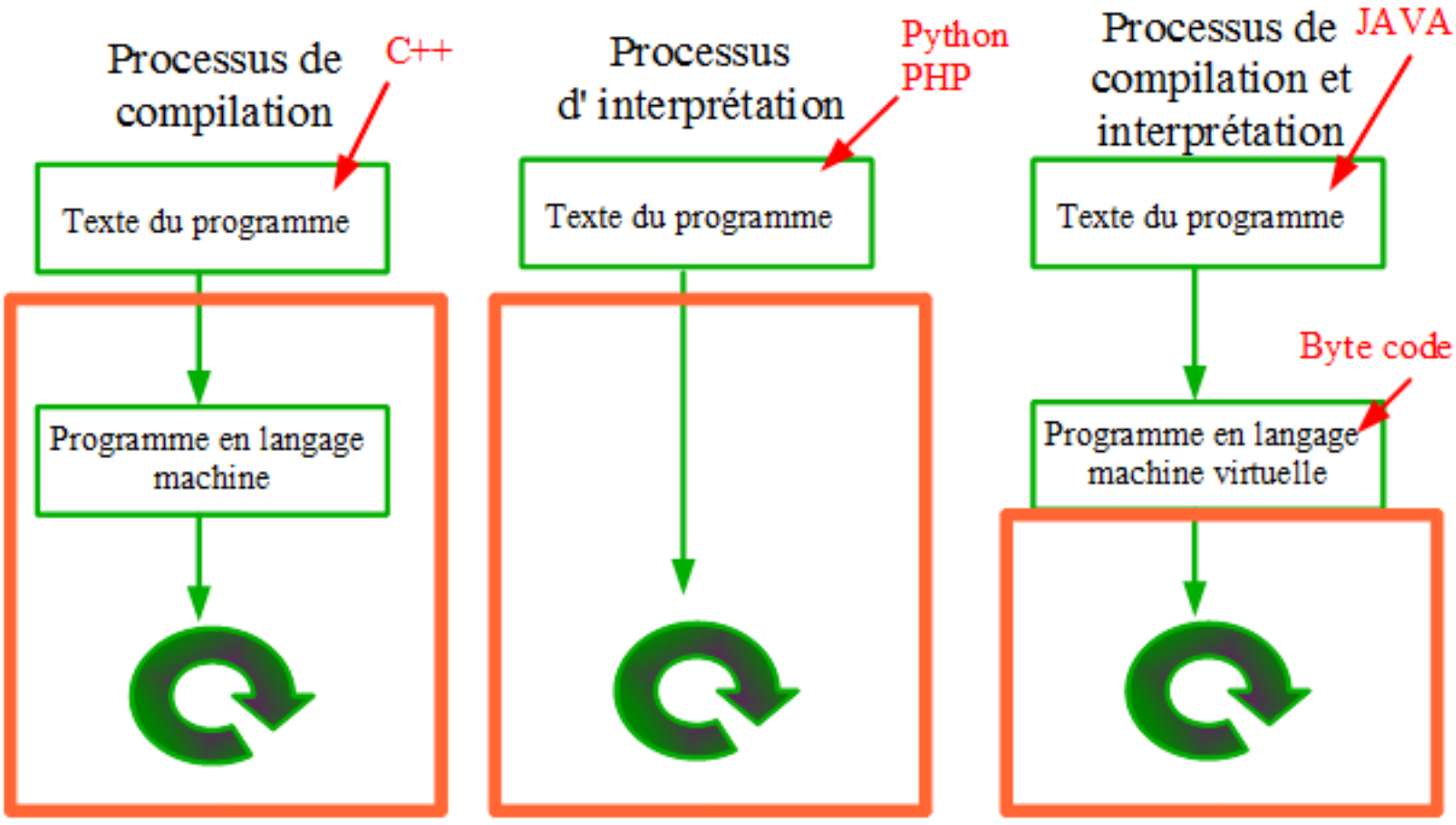
```
1  int sommePremEntiers(int n) {
2      int resultat;
3      resultat = (n * (n+1))/2;
4      return resultat;
5  }
```

Passage du programme au langage machine

```
1  int sommePremEntiers(int n) {
2      int resultat;
3      resultat = (n * (n+1))/2;
4      return resultat;
5  }
```



Compilation et interprétation



Stockage et récupération de données

- Dans des fichiers.
- En bases de données.
  - bases de données relationelles
    - langage SQL
    - structure dirigée par la sémantique des données
    - supprimer la redondance d'information
    - aucune perte d'information
  - bases de données multidimensionnelles
  - ...
- ...

Composantes d'un programme

<table>
<tr><td rowspan="3">Données</td><td colspan="3">Traitements</td></tr>
<tr><td rowspan="2">Opérations sur les données</td><td colspan="2">Enchaînements des opérations</td></tr>
<tr><td>Sélection des opérations</td><td>Répétition des opérations</td></tr>
</table>

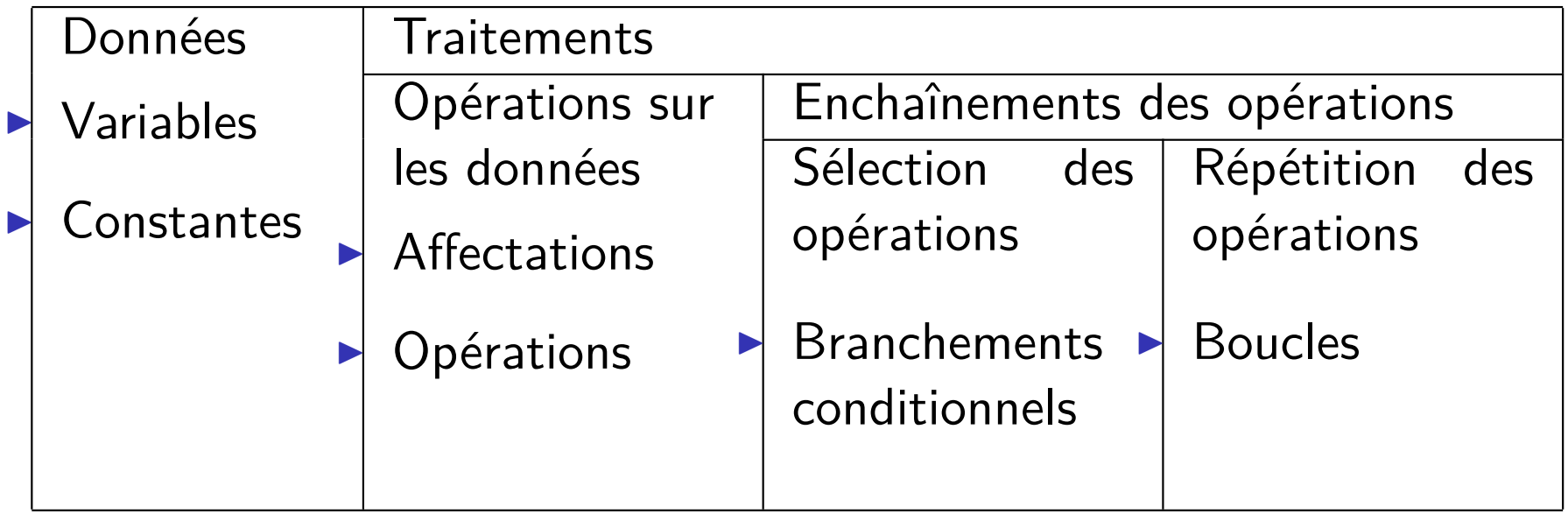
Exemple : recette de cuisine

<table>
<tr><td rowspan="3">Ingrédients (lait, farine, . . .)</td><td colspan="3">Préparation</td></tr>
<tr><td>Faire bouillir le lait.</td><td colspan="2">Incorporer la farine puis incorporer les fruits. . .</td></tr>
<tr><td>Faire fondre le beure. . .</td><td>Si saison = printemps
alors
ajouter cerises
sinon
ajouter pommes</td><td>Tant que la pâte est trop liquide
ajouter farine</td></tr>
</table>

Exemple : montage de meuble en kit

<table>
<tr><td rowspan="3">Pièces de montage (planches, pointes, vis, . . .)</td><td colspan="3">Plan de montage</td></tr>
<tr><td>Mettre une vis 320
Poser la planche 2. . .</td><td colspan="2">Mettre la charnière puis placer la porte gauche. . .</td></tr>
<tr><td>Si ensemble non solide alors ajouter cornières</td><td>Tant que non lisse poncer</td></tr>
</table>

Deux composantes d'un programme



Les données

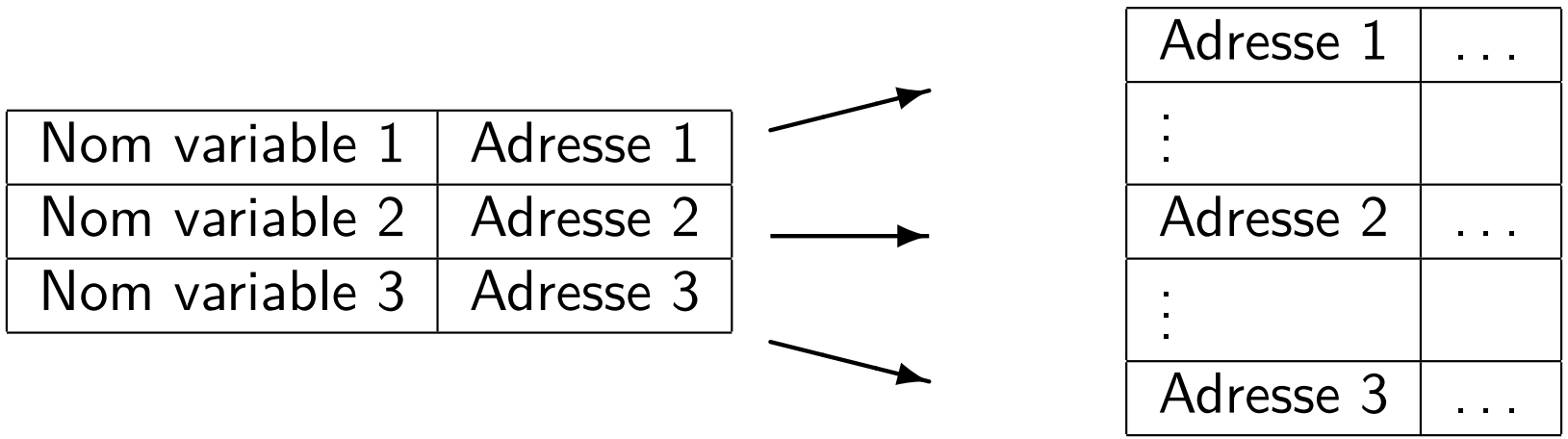
- Zone de mémoire permettant le stockage d'informations
- Identifiée par un nom
- Possède un type (entier, réel, chaîne de caractères. . .)
- **Variable** : modifiable
  - age : entier
  - nom : chaîne de caractères
- **Constante** : non modifiable
  - 4
  - 8
  - PI
  - MAX_FLOAT

Les types de données

- Les données sont typées, exemples de types :
  - entier
  - réel
  - caractère
  - chaîne de caractères
  - booléen (vrai, faux)
  - types de données complexes (classes)
- Exemples de valeurs
  - entiers : 4 25 7
  - réels : 4.7 4.0 7.0
  - chaînes de caractères : "mon texte" "7" " :)" mais aussi "" " " "a"
  - caractères : 'T' '7'
  - booléens : true false

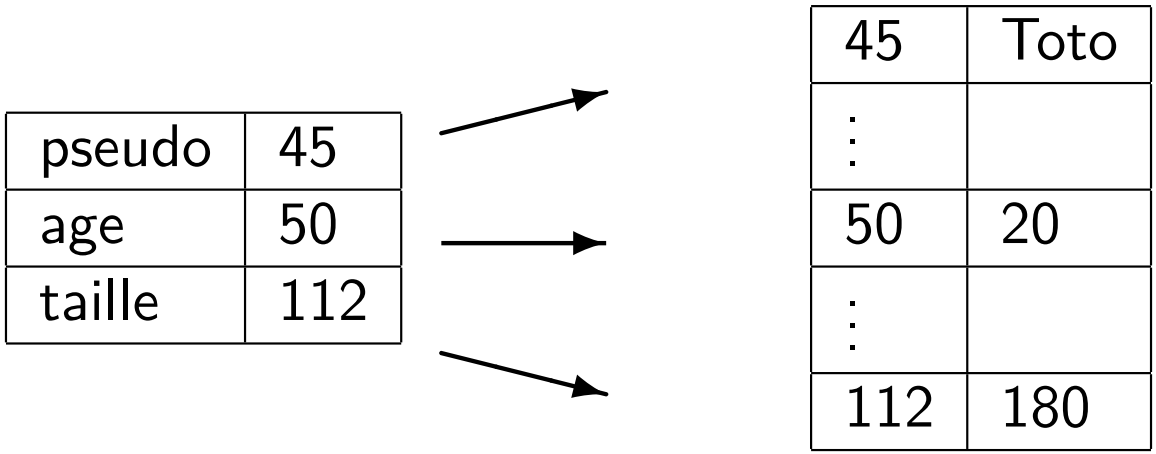
Modèle simplifié du stockage des variables

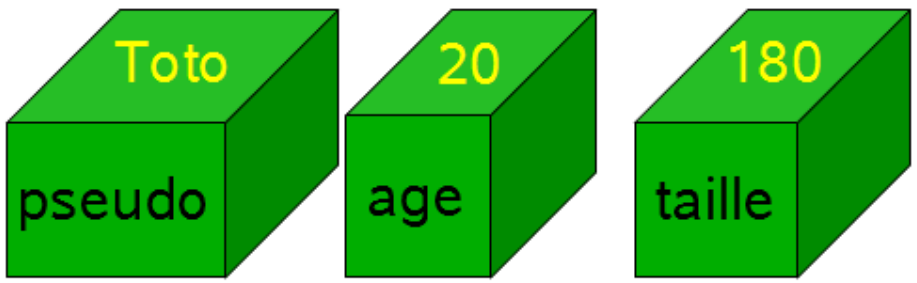
Tableau des variables



Exemple de stockage de variables

Tableau des variables





Structure d'un programme

Un programme (par exemple une fonction ou une procédure) peut être considéré comme une boîte noire.

- On lui fournit des données en entrée (= des paramètres).
- Elle répond en renvoyant des données en sortie (= le résultat).

Les traitements

Comment les données sont-elles manipulées ?

- Opérations sur les données.
- Ordre d'exécution des opéations.
- Conditions d'exécution des opérations.

Écriture d'un programme

- Recherche de l'algorithme. Quelle méthodologie adopter pour répondre à mon problème ?
- Écriture dans un langage de programmation. Comment traduire mon algorithme ?

Qualités d'un programme

- Valide
- Fiable
- Extensible
- Réutilisable
- Compatible

Exemple : calcul de la fréquence cardiaque maximale

Formule de calcul : fCM = 220 - age.

- Création de la variable fCM.

```
1  float fCM;
```

- Affectation du résultat de l'opération 220 - age à la variable fCM.

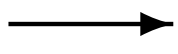
```
1  fCM = 220 − age;
```

  - *= est l'opérateur indiquant que la variable fCM (en partie gauche) prend comme valeur le résultat de la partie droite (220 - age)*
  - *; marque la fin d'une instruction*
  - Calcul de la valeur 220 - age.
  - Modification de la valeur de fCM.

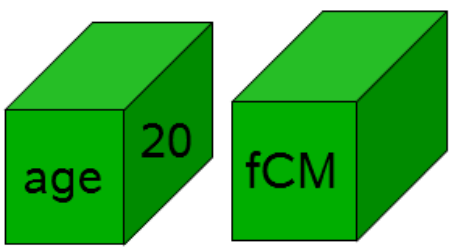
Exemple : calcul de la fréquence cardiaque maximale

- **Création de la variable fCM.**
- Calcul de la valeur (fCM = 220 - age).
- Modification de la valeur de fCM.

pseudo	45
age	50
taille	112
fCM	233



45	Toto
⋮	
50	20
⋮	
112	180
⋮	
233	

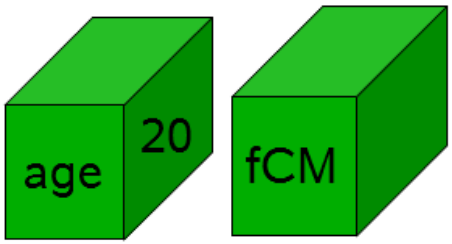


Exemple : calcul de la fréquence cardiaque maximale

- Création de la variable fCM.
- **Calcul de la valeur (fCM = 220 - age).**
- Modification de la valeur de fCM.

pseudo	45
age	50
taille	112
fCM	233

45	Toto
⋮	
50	20
⋮	
112	180
⋮	
233	

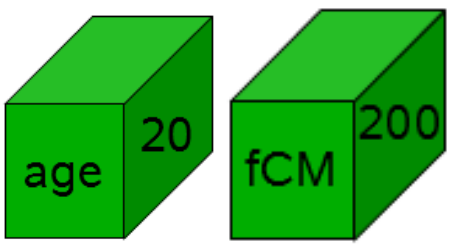


Exemple : calcul de la fréquence cardiaque maximale

- Création de la variable fCM.
- Calcul de la valeur (fCM = 220 - age).
- **Modification de la valeur de fCM.**

pseudo	45
age	50
taille	112
fCM	233

45	Toto
⋮	
50	20
⋮	
112	180
⋮	
233	**200**



Processing et Java

- Processing
  - Logiciel écrit en Java.
  - Permet d'écrire des aplications « simples » en utilisant du code Java.
- Trois composantes d'un programme Processing
  - Déclaration des variables
  - Déclaration des fonctions
  - Initialisation : la procédure setup() est exécutée une fois au début.
  - Traitements : la procédure draw() est exécutée « en boucle » jusqu'à l'arrêt du programme.

Mon premier programme : affichage dans la console

```
1  void setup(){
2      print("Bonjour");
3      println("Bonjour");
4  }
```

Fin de ligne de commande

- Point virgule : **;**
- À la fin de chaque ligne de commande, mettre un **;** pour signaler que la ligne de commande est terminée.

```
1  void setup(){
2      int a;
3      a = 6;
4  }
```

- Une ligne de commande peut s'écrire sur plusieurs lignes physiques.

```
1  void setup(){
2      int a;
3      a =
4          6;
5  }
```

Appel de fonctions et procédures

Utilisation de fonctions ou procédures prédéfinies.
Exemples :

- sqrt(n) : racine carrée
- sq(n) : carré
- pow(n,e) : calcule n à la puissance e
- print(chaineCar) : affiche dans la console la chaîne contenue dans la variable chaineCar

Appel de fonctions et procédures

Attention :

- Si un programme retourne une valeur, c'est une fonction, sinon c'est une procédure.
  - fonctions : sqrt, sq, pow
  - procédure : print, println
- Si un programme a plusieurs paramètres, ils sont séparés par des virgules : pow(n,e).
- Nous pouvons créer nos propres programmes.

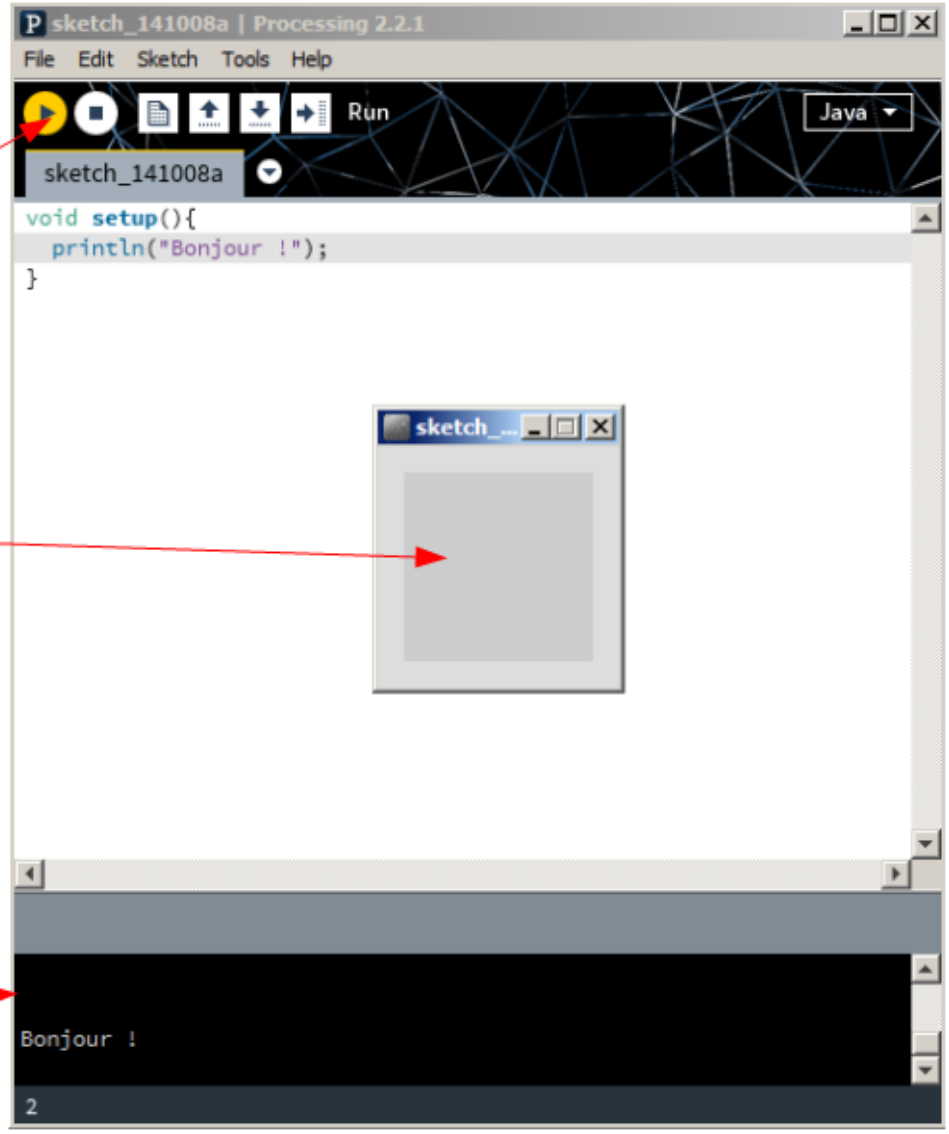
Environnement de développement avec Processing

Environnement de développement avec Processing

Programme en cours d'exécution

Fenêtre graphique

Console où s'affichent les résultats des commandes print et println



Mon deuxième programme

```
1  void setup(){
2      /* ce programme permet de calculer la frquence cardiaque d
           entrainement */
3      int age;
4      float fCM, fCE;
5      age = 21;
6      fCM = 220 - age;
7      fCE = 0.8 * fCM;
8      println("Si vous avez : " + age + " ans votre frequence
           cardiaque d entrainement est " + fCE);
9  }
```

Mon troisième programme

```
1  float freqCardEnt(int age){
2      float fCM, fCE;
3      fCM = 220 - age;
4      fCE = 0.8 * fCM;
5      return fCE;
6  }
7  void setup(){
8      int valAge = 21;
9      println("Si vous avez : " + valAge + " ans votre "+
10     "frequence cardiaque d entrainement est " +
11     freqCardEnt(valAge));
12     valAge = 31;
13     println("Si vous avez : " + valAge + " ans votre"+
14     "frequence cardiaque d entrainement est "+
15     freqCardEnt(valAge));
16 }
```

Commentaires

- Sur une ligne : //. Permet de commenter une ligne ou une fin de ligne.
- Sur une ou plusieurs lignes : commence par /* et se termine par */.

```
1  int factorielle(int nombre) {
2      int fact=1;
3      int i = 1;
4      /* ceci est
5       un commentaire
6       sur plusieurs lignes */
7      do {
8          fact = /* commentaire dans la ligne */ fact * i;
9          i++ ; //commentaire de fin de ligne
10     } while (i<=nombre);
11     // commentaire sur une ligne
12     return fact;
13 }
```

Différentes catégories de données

Propriétés :

- Variables
- Constantes

Types :

- Types de bases
- Objets

Nom des variables

- Commence par une lettre.
- Est une séquence de lettres, de chiffres, ou du caractère _, aucun autre caractère autorisé.
- Sa casse est significative, c'est-à-dire que l'on distingue les majuscules et les minuscules. maVariable $\neq$ maVariAble
- Ne peut être un mot réservé du langage (par exemple *while*).

Conventions de nommage

- Une variable commence par une lettre minuscule, les majuscules servent à séparer les mots composant le nom : *maPremiereVariable*.
- Une constante a toutes ses lettres écrites en majuscules (avec éventuellement des chiffres), le souligné sert à séparer les mots composant le nom : *MA_PREMIERE_CONSTANTE*.

Quelques types de base (=primitifs)

- Entier :

type	esp. (octets)	valeur min.	valeur max.
byte	1	$-2^7 =$ -128	$2^7 - 1 = 127$
short	2	$-2^{15} =$ -32 768	$2^{15} - 1 = 32\ 767$
int	4	$-2^{31} =$ -2 147 483 648	$2^{31} - 1 =$ 2 147 483 647
long	8	$-2^{63} \approx -9 \cdot 10^{18}$	$2^{63} - 1 \approx 9 \cdot 10^{18}$

- Réel :

type	esp. (octets)	amplitude	précision
float	4	limitée	limitée
double	8	moins limitée	moins limitée

- Booléen : *true* ou *false*
- Caractère : char (2 octets), un caractère s'écrit entre deux simples quotes : ' '.

Précision des flottants : illustration

```
1  void setup(){
2    float a = 0.1;
3    float b = 0.1;
4    float c = 0.01;
5
6    float d = a*b;
7
8    println(d==c);
9  }
```

Précision des flottants : comparaison avec R et SAS

R	SAS

```
> a <- 0.1
> b <- 0.1
> a*b
[1] 0.01
> c <- 0.01
> c==(a*b)
[1] FALSE
```

```
data a;
v=0.1;
vv=0.1;
v2=0.01;
bool=(v*vv=v2);
run;
```

Affectation d'une valeur

- Opérateur d'affectation : =
- Signifie : « prendre la valeur de la partie droite et la mettre dans la partie gauche ».

```
1  void setup(){
2      int a, b, c; //declaration des variables
3      a = 5;
4      b = a + 3;
5      c = a + b;
6
7      char lettre;
8      lettre = 'R';
9  }
```

Quelques types de base (=primitifs)

- Entier :

```
1 int monEntier = 5;
2 int monDeuxiemeEntier = -4245;
```

- Réel :

```
1 float monReel = 5.0/8; //0,625 sera stocke !
2 float monReel = -56.3467; //avec un point !
```

- Booléen :

```
1 boolean monBooleen = true;
2 boolean monBooleen = false;
```

- Caractère :

```
1 char monCaractere = 'a';
2 char monCaractere = 'C';
```

Amusons-nous un peu !

Qu'affichent ces programmes ?

```
1  void setup(){
2    int a = Integer.MAX_VALUE;
3    a = a + 1;
4    println(a);
5  }
```

```
1  void setup(){
2    float prixTTC;
3    prixTTC = (1+0.2)*50;
4    println(prixTTC);
5  }
```

Chaîne de caractères

- String
- type non primitif (pas de base avec le langage)
- le contenu de la chaîne s'écrit entre double quotes

```
1  String chaineCar;
2  chaineCar = "contenu";
```

- Pour accéder au i^{eme} caractère de la chaîne : charAt(i-1).
- Pour obtenir la longueur de la chaîne : length().

```
1  char a = chaineCar.charAt(5); //6e caractere de la chaine
2  int b = chaineCar.length();
```

Chaîne de caractères

- Concaténation des chaînes avec l'opérateur +.

```
1  void setup(){
2      String maChaine1 = "Premiere partie " ;
3      String maChaine2 = " deuxieme partie" ;
4      println(maChaine1 + maChaine2) ;
5      int val = 12 ;
6      println(maChaine1 + val + maChaine2) ;
7  }
```

Chaîne de caractères

- Test d'égalité des chaînes avec la fonction equals(String).

```
1  void setup(){
2      String a = "good";
3      String b = "bye";
4      String c = a + b;
5
6      if (c == "goodbye") {
7          println("The == works");
8      }
9
10     if (c.equals("goodbye")) {
11         println("The .equals works");
12     }
13 }
```

Qu'est-ce qu'un algorithme ? (rappel)

Cf début du cours :

- **Algorithme** : description, en un nombre **fini** d'étapes, d'un enchaînement d'opérations élémentaires, permettant d'effectuer un calcul.

Exemples :

- La recette de cuisine.
- Les instructions d'assemblage d'un meuble.

Quels types de traitements ? (rappel)

Un ensemble d'instructions :

- Affectations
- Opérations

Des conditions d'exécution des instructions :

- Structure conditionnelle (ensemble d'instructions exécuté si certaines conditions sont vérifiées)
- Structure itérative (ensemble d'instructions exécuté un certain nombre de fois)

Opérations

- Addition : $+$
- Soustraction : -
- Produit : *
- Division : /
- Reste de la division entière (modulo) : %
  - $23 \% 7 = 2$

Écritures compactes

- $i = i + 1; \iff i + +;$
- $i = i - 1; \iff i - -;$
- $i = i + 10; \iff i+ = 10;$
- $i = i * 1; \iff i* = 8;$
- ...

Le piège de la division

Qu'affiche le code suivant ?

```
1  void setup(){
2      float a = 2/3;
3      float b = 2/3.;
4      float c = 3;
5      float d = (c*2)/3;
6      float e = c*(2/3);
7      println(a + ", " + b + ", " + c);
8      println(d + ", " + e);
9  }
```

Structures de contrôles

Structures conditionnelles :

- Si. . . alors. . . sinon. . .
- Choix selon

Structures itératives :

- Boucle pour
- Boucle tant. . . que
- Boucle répéter. . . tant que

Expression conditionnelle

- Présente dans toute structure de contrôle
- Est vraie ou fausse (booléen) – *true* ou *false*.
- Exemple :
  - var == 5 (vraie, si et seulement si var est égale à 5)
  - Notez le double symbole = !
- Opérateurs conditionnels :
  - >, <, >=, <=, !=, ==
  - exemples :
    - a>2
    - b==7
    - (x<7) && (z==9)

Composition d'expression conditionnelle

- Et : &&
- Ou : ||
- Non : !

Tableaux de vérité

Exemples

A	B	A && B	A || B
Vrai	Vrai	Vrai	Vrai
Vrai	Faux	Faux	Vrai
Faux	Vrai	Faux	Vrai
Faux	Faux	Faux	Faux

A	! A
Vrai	Faux
Faux	Vrai

x	(x>9)	(x<15)	(x>9) && (x<15)
7	Faux	Vrai	Faux
18	Vrai	Faux	Faux
10	Vrai	Vrai	Vrai
1	Faux	Vrai	Faux

Structure conditionnelle simple

Un bloc d'instructions est exécuté si une condition est vérifiée.

```
1  if(condition){
2      instruction1;
3      instruction2;
4      ...
5      instructionN;
6  }
```

Structure conditionnelle

Un bloc d'instructions est exécuté si une condition est vraie. Si elle est fausse un autre bloc est exécuté.

```
1  if(condition){
2      instruction1;
3      instruction2;
4      ...
5      instructionN;
6  }
7  else{
8      instructionP;
9      ...
10     instructionQ;
11 }
```

Structure conditionnelle : exemple

Conseil à un coureur.

```
1  void conseilCoureur(int age, float fC){
2      float fCE = (220-age)*0.8;
3      if (fC >= fCE) {
4          println("Ralentir");
5      } else {
6          println("Continuer ainsi");
7      }
8  }
9  void setup(){
10     int a = 20;
11     float f = 153.5;
12     conseilCoureur(a,f);
13 }
```

Imbrication de structures conditionnelles

```
1  if(conditionA1) {
2      Bloc d instructions A 1
3  }
4  else{
5      if(conditionA2){
6          Bloc d instructions A 2
7      }
8      else {
9          Bloc d instructions A 3
10     }
11 }
```

Imbrication de structures conditionnelles

```
1  if(conditionB1) {
2      if(conditionB2){
3          Bloc d instructions B 2
4      }
5      else {
6          Bloc d instructions B 1
7      }
8  }
9  else{
10     Bloc d instructions B 3
11 }
```

Imbrication de structures conditionnelles

```
1  if (conditionA1) {
2      Bloc d instructions A 1
3  }
4  else if (condition A2){
5      Bloc d instructions A 2
6  }
7  else{
8      Bloc d instructions A 3
9  }
```

Structure de choix

```
switch (variable) {
    /* variable doit etre un entier ou un caractere */
    case CONSTANTE1 :
        liste d instructions1
        break;
    case CONSTANTE2 :
        liste d instructions2
        break;
    case CONSTANTE3 :
        liste d instructions3
        break;
    default :
        liste d instructions4
}
```

Structure de choix : exemple

```
1  int var;
2
3  ...
4
5  switch (var) {
6          case 1 : procedure1();
7                   break;
8          case 2 : procedure2();
9                   break;
10         case 3 : procedure3();
11                  break;
12         default : println("Votre valeur n'est pas comprise entre 1
                       et 3");
13 }
```

Structure itérative

- Boucle *for* : pour exécuter un bloc d'instructions un nombre de fois qui est fixé avant de débuter l'exécution de la liste d'instructions.
- Boucle *while* : pour exécuter un bloc d'instructions tant qu'une condition donnée est vraie. Si avant de débuter l'exécution du bloc la condition est fausse alors aucune instruction n'est exécutée.
- Boucle *do . . . while* pour exécuter un bloc d'instructions tant qu'une condition donnée est vraie. Quelle que soit la condition, le bloc d'instructions est exécuté au moins une fois.

Boucle for

- Nombre d'itérations connu : le bloc d'instructions est exécuté x fois.
- Une variable, indice de boucle, varie d'un pas constant.
- L'indice de boucle ne doit être utilisé que dans la boucle.

```
1  for (condition debut ; condition fin ; incrementation
       indice) {
2      <bloc d instructions>
3  }
```

```
1  void setup(){
2      int n = 5;
3      for (int i=0 ; i<n ; i++) {
4          println ( i );
5      }
6  }
```

Boucle for : exemple

```
1  int factorielle(int nombre){
2      int fact = 1;
3      for (int i=1 ; i<=nombre ; i++) {
4          fact = fact * i;
5      }
6      return fact;
7  }
8  void setup() {
9      int nb = 5;
10     println( "la factorielle de " + nb + " est " +
11         factorielle(nb));
12 }
```

Boucle for : exemple

```
1  int sommeNPremiers(int n) {
2      int somme=0;
3      for (int i=1 ; i<=n ; i++) {
4          somme = somme + i;
5      }
6      return somme;
7  }
8  void setup(){
9      int n = 5;
10     println( "la somme des " + n +
11         " premiers entiers est " + sommeNPremiers(n) );
12 }
```

Boucle while

- Répétition d'un bloc d'instructions tant qu'une condition est vérifiée.
- Le test se place avant le bloc d'instructions.
- Il ne faut pas sortir de la boucle autrement que sur la non vérification de la condition.

```
1 while (condition) {
2       liste instructions
3 }
```

Boucle while : exemple

```
1  int factorielle(int nombre){
2      int fact=1;
3      int i = 1;
4      while (i<=nombre) {
5          fact = fact * i;
6          i++;
7      }
8      return fact;
9  }
10 void setup() {
11     int n = 5;
12     println( "La factorielle de " + n + " est "
13           + factorielle(n) );
14 }
```

Boucle while : exemple

```
1  int sommeNPremiers(int n) {
2      int somme=0;
3      int i = 1;
4      while (i<=n ) {
5          somme = somme + i;
6          i++;
7      }
8      return somme;
9  }
10 void setup(){
11     int n = 5;
12     println( "la somme des " + n +
13         " premiers entiers est " + sommeNPremiers(n) );
14 }
```

Boucle while : exemple

```
1  void affichage(int n) {
2      int i = 1;
3      while (i<=n ) {
4          if ((i%3) == 0) {
5              println( i );
6          }
7          i++;
8      }
9  }
10 void setup(){
11     int n = 5;
12     affichage(n);
13 }
```

Boucle while : exemple

```
1  int pge(int n) {
2      int val = n;
3      while ((val%3) != 0) {
4          val--;
5      }
6      return val;
7  }
8  void setup(){
9      int n = 5;
10     println( "Le plus grand entier inferieur  " + n
11         + " et multiple de 3 est " + pge(n));
12 }
```

Boucle do . . . while

- Répétition d'un bloc d'instructions tant qu'une condition est vérifiée.
- Le test se place après le bloc d'instructions.
- Il ne faut pas sortir de la boucle autrement que sur la non vérification de la condition.

```
1 do{
2     <bloc d instructions>
3 }
4 while (<condition>);
```

Boucle do . . . while : exemple

```
1  int factorielle(int nombre){
2      int fact=1;
3      int i = 1;
4      do {
5          fact = fact * i;
6          i++;
7      } while (i<=nombre);
8      return fact;
9  }
10 void setup() {
11     int n = 5;
12     println( "La factorielle de " + n + " est "
13           + factorielle(n) );
14 }
```

Comparaison while et do . . . while

Imaginons maintenant que i soit initialisé à 2.
Dans quel cas les fonctions *factorielle* des diapositives 89 et 94 n'affichent-ils pas la même chose ?

Structurer un programme

- Structurer un programme, c'est le décomposer en fonctions et procédures.
- Un programme est une suite de lignes logiques et de lignes blanches.
- Ligne logique :
  - une ou plusieurs lignes physiques
  - terminée par **;**
  - souvent, une ligne logique = une ligne physique
- Ligne blanche :
  - ligne ne contenant aucun caractère
  - ligne ne contenant que des commentaires

Manières de décomposer un programme

- Tout dans le programme principal (setup).
  - trop lourd
  - illisible
  - trop de variables
  - code dupliqué
  - moins évolutif
- Décomposition en sous-programmes.
  - définir des sous-problèmes
  - rédiger des fonctions et des procédures

Fonction vs procédure

- Une fonction renvoie quelque chose (une valeur, avec un type fixé).
- Une procédure ne renvoie rien (void), elle n'agit que par effet de bord (des affichages, ...).

Exemple : fonction vs procédure

```
1  String coucou(){
2    return "Coucou";
3  }
4
5  void setup(){
6    println(coucou());
7  }
```

```
1  void coucou(){
2    println("Coucou");
3  }
4
5  void setup(){
6    coucou();
7  }
```

(Dé)composition d'un programme

- Un algorithme est composé de trois parties :
  - données à traiter (= entrées = paramètres)
  - traitement des données
  - résultat (= sortie)
- Le traitement des données peut se décomposer en sous-problèmes. Chaque sous-problème est composé de :
  - données à traiter
  - traitement des données
  - résultat

Passages de paramètres

- Les paramètres des fonctions et procédures peuvent être modifiés.
- Les types primitifs et les String sont passés aux fonctions et procédure **par recopie** (= par valeur).
- Les tableaux et les objets (mutables) sont passés **par référence**.

Passages de paramètres

pass by reference

```
cup =
```

```
fillCup(      )
```

pass by value

```
cup =
```

```
fillCup(      )
```

www.penjee.com

Portée des variables

- Les variables sont crées où elles sont déclarées.
- Elles sont détruites lorsqu'on quitte le bloc d'instructions où elles sont déclarées.
- Après leur déclaration, elles sont utilisables danc le bloc où elles ont été créées et dans tous les blocs imbriqués suivants.

  Rappel : un bloc d'instructions est un ensemble d'instructions contenues entre 2 instructions indentées au même niveau.

Exemple. Qu'affiche le code suivant ?

```
 1  int maFonction(int n)
 2  {
 3    n++;
 4    return n;
 5  }
 6
 7  void maProcedure(int n)
 8  {
 9    n++;
10    println(n);
11  }
12
13  void setup(){
14    int n=57;
15    println(maFonction(n));
16    maProcedure(n);
17  }
```

Questions à se poser

- La chaîne peut-elle être vide ? Si oui, faut-il traiter ce cas à part ?
- Le premier élément peut-il être obtenu commes les autres ?
- Quel est le dernier élément ? Faut-il le traiter comme les autres ?

Exemple 1

- Compter le nombre de « e » dans une phrase.
- Traitement d'une chaîne dont l'élément final est un point.

- Accès à un élément : maPhrase.charAt(i).
- Accès au premier élément : maPhrase.charAt(0).
- Le dernier élément est le caractère '.'.
- La chaîne ne peut être vide, il y a au moins un point.
- On fait le test maPhrase.charAt(i)=='e'.

Exemple 1 : do . . . while

```
1  int compteE(String maPhrase){
2    int nbE =0;
3    int indLettre = -1;
4    do{
5      indLettre = indLettre + 1;
6      if (maPhrase.charAt(indLettre) == 'e'){
7        nbE++;
8      }
9    }while (maPhrase.charAt(indLettre) != '.');
10   return nbE;
11 }
12 void setup(){
13   println(compteE("exemple."));
14 }
```

Exemple 1 : while

```
1  int compteE(String maPhrase){
2    int nbE =0;
3    int indLettre = 0;
4    while (maPhrase.charAt(indLettre) != '.'){
5      if (maPhrase.charAt(indLettre) == 'e'){
6        nbE++;
7      }
8      indLettre = indLettre + 1;
9    }
10   return nbE;
11 }
12 void setup(){
13   println(compteE("exemple."));
14 }
```

Exemple 2

- Compter le nombre de « le » dans une phrase.
- Traitement d'une chaîne dont l'élément final est un point.

- Accès à un élément : maPhrase.charAt(i).
- Accès au premier élément : maPhrase.charAt(0).
- Le dernier élément est le caractère '.'.
- La chaîne ne peut être vide, il y a au moins un point.
- On fait un des tests suivants :
  - le caractère courant est un e et le précédent était un l (maPhrase.charAt(k)=='e') et (maPhrase.charAt(k-1)=='l')
  - le caractère courant est un l et le suivant est un e (maPhrase.charAt(k)=='l') et (maPhrase.charAt(k+1)=='e')

Exemple 2a : deux caractères

```
 1  int compteLe(String maPhrase) {
 2    int nbLe =0;
 3    char carPrec ='f';
 4    int indLettre =0;
 5    char carCour= maPhrase.charAt(indLettre);
 6    while (carCour != '.') {
 7      if ((carCour == 'e') && (carPrec=='l') ) {
 8        nbLe = nbLe + 1;
 9      }
10      indLettre = indLettre + 1;
11      carPrec = carCour;
12      carCour = maPhrase.charAt(indLettre);
13    }
14    return nbLe;
15  }
```

Exemple 2a : un booléen

```
1  int compteLe(String maPhrase) {
2    int nbLe = 0; boolean precL = false; int indLettre = 0;
3    char carCour = maPhrase.charAt(indLettre);
4    while (carCour != '.') {
5      if ((carCour == 'e') && precL) {
6        nbLe = nbLe + 1;
7      }
8      if (carCour == 'l') {
9        precL = true;
10     } else {
11       precL = false;
12     }
13     indLettre = indLettre + 1;
14     carCour = maPhrase.charAt(indLettre);
15   }
16   return nbLe;
17 }
```

Exemple 2a : un booléen (raccourci)

```
1  int compteLe(String maPhrase) {
2    int nbLe =0;
3    boolean precL =false;
4    int indLettre =0;
5    char carCour = maPhrase.charAt(indLettre);
6    while (carCour != '.') {
7      if ((carCour == 'e') && precL) {
8        nbLe = nbLe + 1;
9      }
10     precL = (carCour == 'l') ;
11     indLettre = indLettre + 1;
12     carCour = maPhrase.charAt(indLettre);
13   }
14   return nbLe;
15 }
```

Exemple 2b : deux caractères

```
 1  int compteLe(String maPhrase) {
 2    int nbLe = 0; int indLettre = 0; char carSuiv = 'f';
 3    char carCour = maPhrase.charAt(indLettre);
 4    if (carCour != '.' ) {
 5      carSuiv = maPhrase.charAt(indLettre + 1);
 6    }
 7    while (carCour != '.') {
 8      if ( (carCour =='l') && (carSuiv=='e')) {
 9        nbLe = nbLe + 1;
10      }
11      indLettre = indLettre + 1;
12      carCour = carSuiv;
13      if (carCour != '.' ) {
14        carSuiv = maPhrase.charAt(indLettre + 1) ;
15      }
16    }
17    return nbLe;
```

Exemple 3

- Compter le nombre de mots.
- Traitement d'une chaîne dont l'élément final est un point.

- Solution 1 : compter les débuts de mots.
- Solution 2 : compter les fins de mots.

Définition

- Une fonction récursive est une fonction qui s'appelle elle-même.
- Par exemple, la factorielle peut être implémentée de manière récursive : $n! = n \cdot (n-1)!$.

Exemple : fonction factorielle

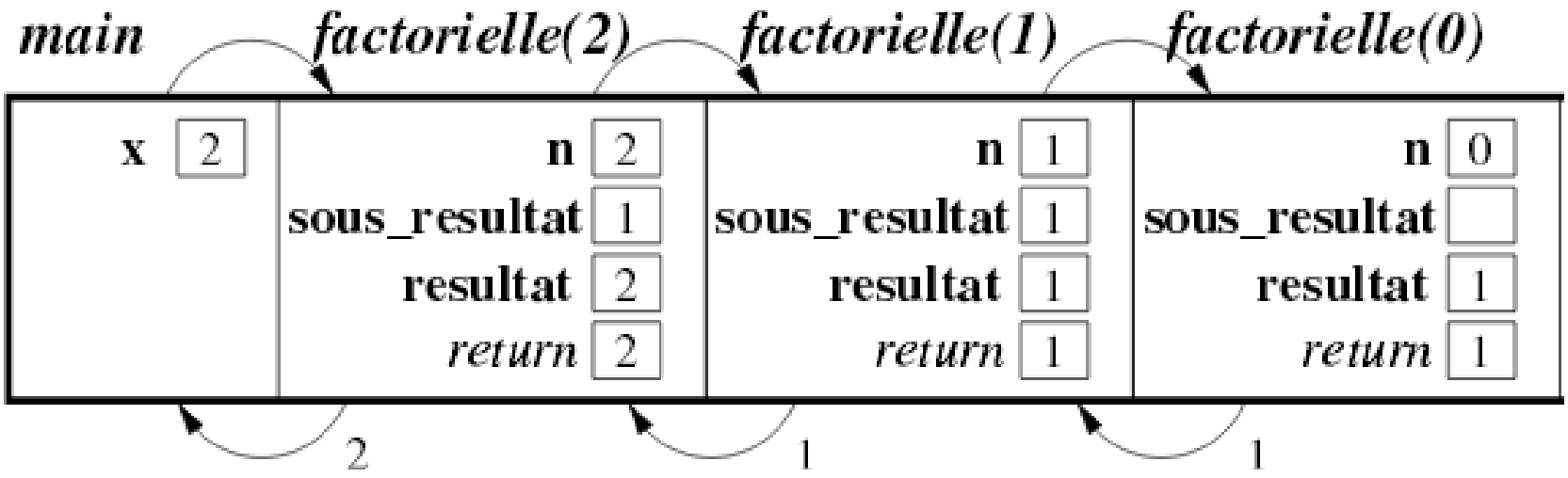
```
 1  int fact(int n) {
 2      int factorielle;
 3      if (n <= 1) {
 4          factorielle = 1;
 5      }
 6      else {
 7          factorielle = n * fact(n-1);
 8      }
 9      return factorielle;
10  }
11  void setup() {
12    println(fact(8));
13  }
```

Récursivité : analyse récursive

- Trouver le critère d'arrêt de la récursivité. Pour la factorielle : $n \leq 1$.
- Lors de l'appel récursif, l'ensemble des paramètres doit tendre vers le critère d'arrêt. Pour la factorielle : c'est évident.

Récursivité : structure de pile

- Mémorisation des valeurs avec une pile.



Source : deptinfo.cnam.fr

Les tableaux : définition

- Contrairement à une variable, un tableau permet de stocker plusieurs données. Les données doivent être du même type.
- En Java, les tableaux (basiques) sont de taille fixe. Une fois créée, la taille du tableau ne peut pas évoluer.
- Les éléments d'un tableau sont :
  - ordonnés
  - modifiables
  - d'un type donné
- Mode d'écriture :
  - nom de tableau (comme les noms de variables)
  - $\{$: en début de tableau
  - $\}$: en fin de tableau
  - , : séparateur des éléments d'un tableau

Les tableaux : création

Deux manières de créer une liste :

- en donnant ses valeurs :

```
1  int[] monTableau = {1, 2, 3, 87, 34, 12, 1};
```

- en donnant seulement sa taille :

```
1  int[] monAutreTableau = new int[9];
```

Les tableaux : exemples

```
1  void setup(){
2      int[] joursParMois = { 28 , 29 , 30 , 31 };
3      String[] joursOuvres = { "lundi", "mardi" , "mercredi" ,
4      "jeudi" , "vendredi"};
5      int [][] matrice ={{1,2,3},{4,5,6},{7,8,9}};
6  }
```

Les tableaux : accès aux données

Le premier élément est à l'indice 0.

```
1  void setup() {
2    int[] maListe = {1, 2, 3};
3    int[] monAutreListe = new int[3];
4    for (int i=0; i<3; i++) {
5      monAutreListe[i] = maListe[i];
6    }
7    println(monAutreListe[2]);
8  }
```

Les tableaux : qu'affiche ce code ?

```
1  void setup() {
2    int[] maListe = {1, 2, 3};
3    int[] monAutreListe = new int[3];
4    for (int i=0; i<3; i++) {
5      monAutreListe[i] = maListe[i];
6    }
7    maListe[2]=5;
8    println(monAutreListe[2]);
9    monAutreListe=maListe;
10   maListe[1]=7;
11   println(monAutreListe[1]);
12 }
```

Cas particulier : les matrices

- En informatique, les mots tableau et liste désignent des structure à une dimension.
- Pour se référer à une structure bidimensionnelle, on préfère le mot *matrice*.
- Une matrice est un *tableau de tableaux*.
- Rien n'impose qu'une matrice soit carrée. . .
- . . . ni même rectangle.
- Une matrice mathématique est représentable de manière immédiate par une matrice en informatique. L'inverse n'est pas vrai.
- Chacun doit définir sa convention pour ce qui est de distinguer les lignes des colonnes.

Affichage d'une matrice

```
1  void afficheMatrice(int[][] maMatrice) {
2    int nbLignes = maMatrice.length;
3    int nbColonnes = maMatrice[0].length;
4    for (int indLigne=0; indLigne < nbLignes; indLigne++) {
5      for (int indColonne=0; indColonne < nbColonnes;
           indColonne++) {
6        print(maMatrice[indLigne][indColonne]);
7        print(" ");
8      }
9      println() ;
10   }
11 }
12 void setup() {
13   int[][] maMatrice = {{1,2},{3,4}};
14   afficheMatrice(maMatrice);
15 }
```

Les tableaux : compléments

Se référer au livre sur la plate-forme d'enseignement, pages 79 à 82.

Rôle des fichiers

- Échanger de données entre plusieurs programmes.
- Stocker des ensembles d'informations (appelées « enregistrement »).
- Conserver des données à la fermeture d'une application.
- Récupérer des données à l'ouverture d'une application.
- ...

Types de fichiers

Il existe différent types de manières de parcourir un fichier en lecture et en écriture.

- Accès séquentiel : on accède à chaque enregistrement en commençant toujours par le premier puis en passant à l'enregistrement suivant.
- Accès direct (aussi appelé aléatoire) : on accède directement à un enregistrement avec son adresse dans le fichier.
- Accès indexé : le fichier est complété de données permettant d'accéder directement à l'information dans le fichier (évite de parcourir tout le fichier comme en mode séquentiel).

Pour la culture, notez le parallèle avec la gestion de la mémoire. Random Access Memory = « mémoire à accès non séquentiel ».

Identification d'un fichier

Nom du fichier dans le système d'exploitation :

- monFichierDonnees.txt
- monRepert1/monSousRepert2/monFichierDonnees.txt

Changement de répertoire :

- Adresse absolue : P :/rep1/rep12/monFichierDonnees.txt
- Adresse relative : ../../rep34/monFichierDonnees.txt

Lecture d'un fichier

Processing nous offre une fonction permettant de lire directement toutes les lignes d'un fichier et de les charger dans un tableau de String : loadStrings().

```
1  void setup() {
2    String lignes[] = loadStrings("liste.txt");
3    println("il y a " + lignes.length + " lignes");
4    for (int i = 0; i < lignes.length; i++) {
5      println(lignes[i]);
6    }
7  }
```

- Le fichier appelé doit être présent dans le répertoire de données du sketch...
- ...ou être indiqué par son chemin absolu ...
- ...ce peut même être l'adresse d'un fichier distant (url) !

Écriture dans un fichier

Processing nous offre de même une fonction permettant d'écrire directement dans un fichier toutes les lignes d'un tableau de String : saveStrings().

```
1  void setup() {
2      String[] mots = {"pomme", "ours", "chat", "chien"};
3      saveStrings("noms.txt", mots);
4  }
```

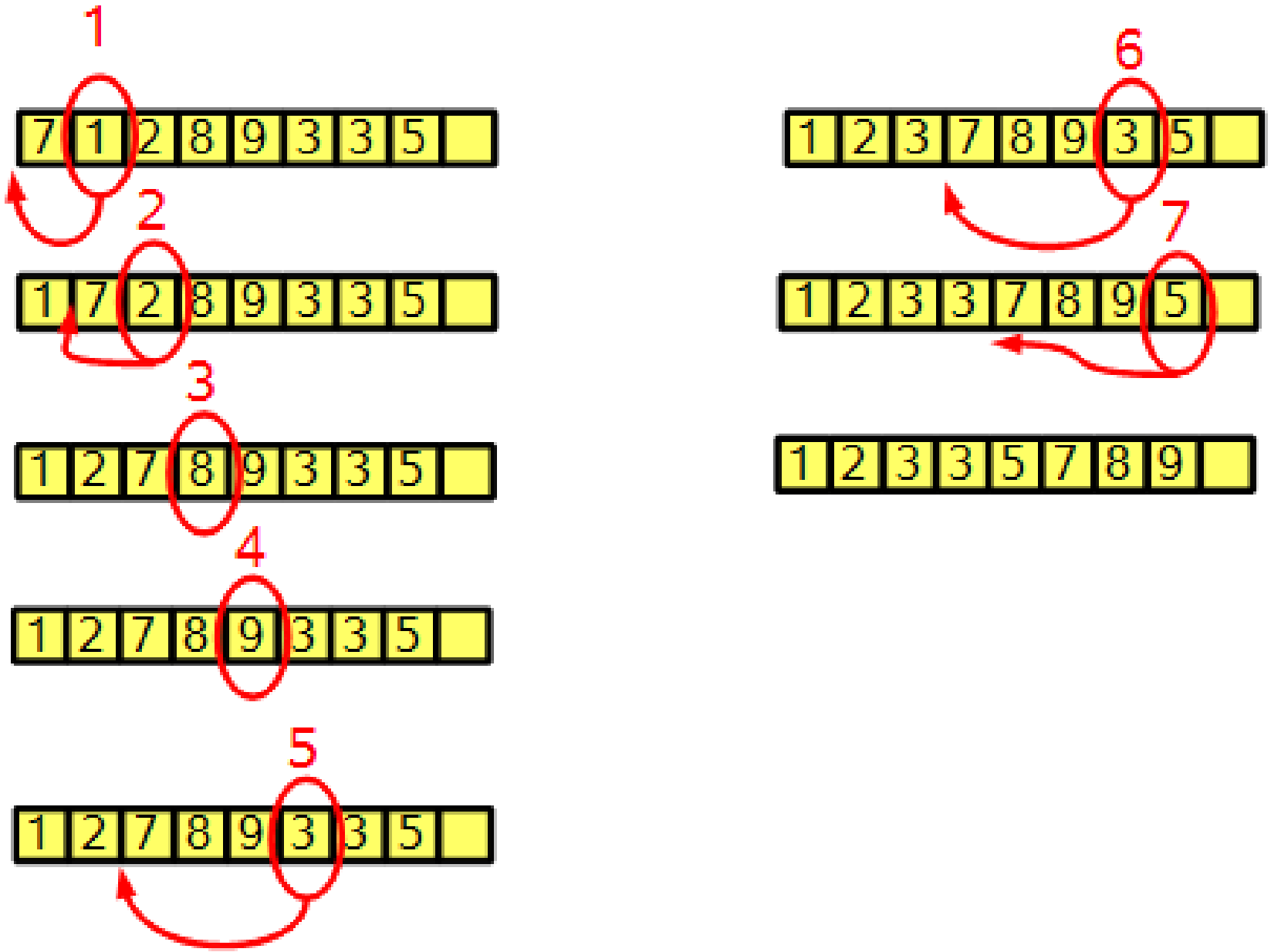
Comportements de loadStrings et saveStrings

- Par défaut, loadStrings va chercher dans le répertoire *data*, alors que saveStrings sauvegarde dans le répertoire du sketch !
- Si le fichier n'est pas disponible ou qu'une erreur survient, loadStrings renvoie *null*. Le programme ne s'arrête pas, c'est à vous de gérer ce cas !
- À partir de processing 0134, seul l'encodage UTF-8 est utilisé.

Tri par insertion du minimum

- Pour chaque élément i du tableau on cherche sa position dans les $i-1$ éléments du début du tableau déjà trié.
- Le i^{eme} élément est comparé au premier, s'il est inférieur il remplace le premier (et on décale les suivants) sinon on le compare au suivant...
- Pour chaque élément on s'arrête lorsque l'on a trouvé un élément plus grand dans le début du tableau ou lorsque le début du tableau a été complètement parcouru (le i^{eme} élément est bien placé).

Tri par insertion du minimum : exemple



Tri par insertion du minimum : programmation

```
 1  void triInsertion(int[] liste) {
 2    int indValeurInseree = 1;
 3    int valeurInseree;
 4    int indice;
 5    while (indValeurInseree != liste.length) {
 6      indice = indValeurInseree - 1;
 7      valeurInseree = liste[indValeurInseree] ;
 8      while ( (indice >= 0)
 9        && (liste[indice] > valeurInseree)) {
10        liste[indice+1] = liste[indice];
11        indice = indice - 1;
12      }
13      liste[indice+1] = valeurInseree;
14      indValeurInseree = indValeurInseree + 1;
15    }
16  }
```

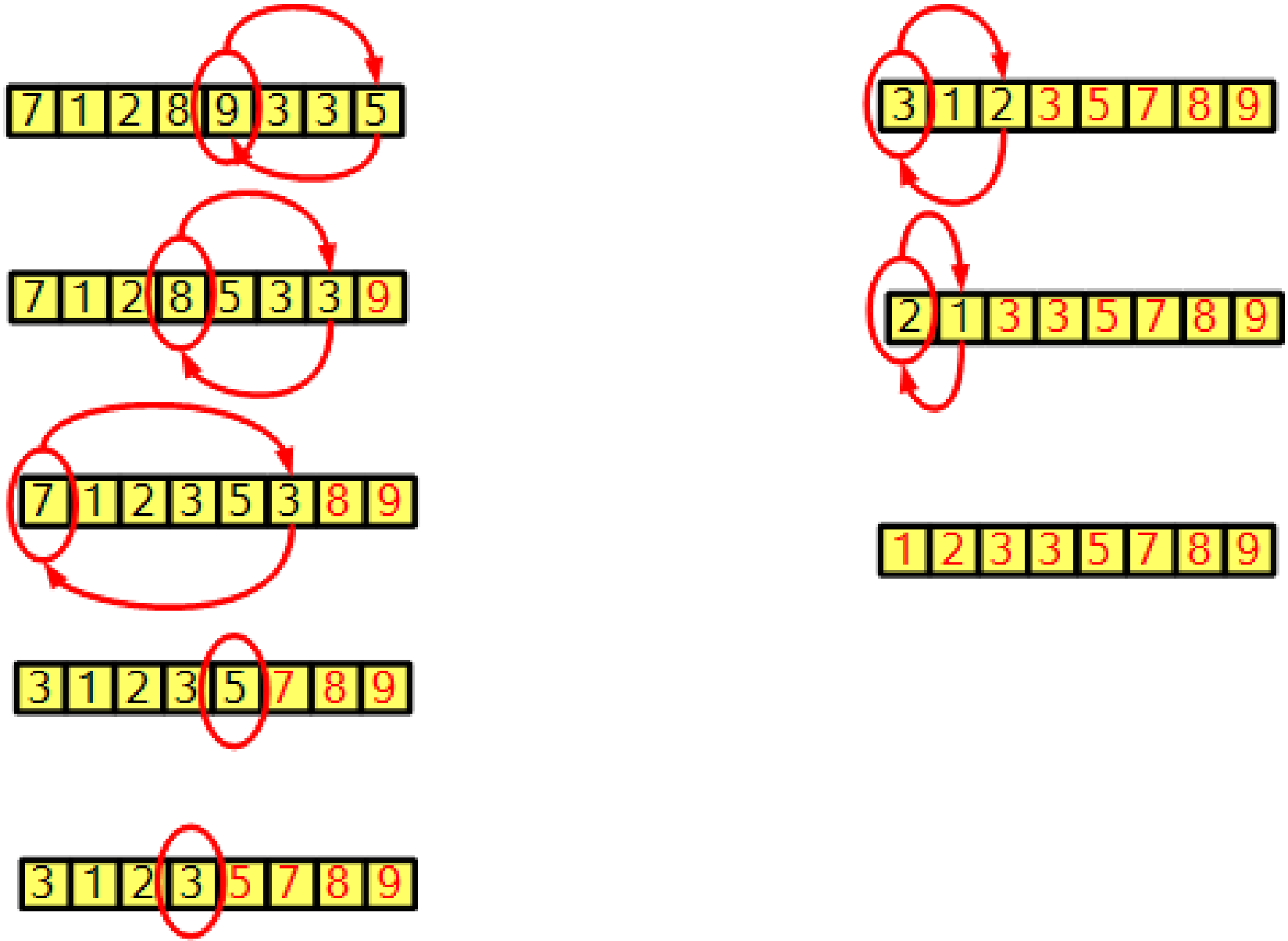
Tri par insertion du minimum : programmation

```
1  void setup() {
2    int[] maListe= {1, 5, 6, 89, 3};
3    triInsertion(maListe);
4
5    for (int i=0; i<maListe.length; i++)
6    {
7      print(maListe[i]+" ");
8    }
9  }
```

Tri par extraction du maximum

- Parmi tous les éléments du tableau rechercher celui qui a la plus grande valeur.
- Échanger cette valeur avec celle de la dernière position (position n).
- Rechercher ensuite le plus grand dans les $n-1$ premiers éléments.
- Échanger cet élément avec celui de la position $n-1$.
- Réitérer le procédé pour toutes les autres positions du tableau (sauf la première).

Tri par extraction du maximum : exemple



Tri par extraction du maximum : programmation

```
1  void permuter(float liste[], int i, int j)
2  {
3      float temp = liste[i];
4      liste[i] = liste[j];
5      liste[j] = temp;
6  }
```

Tri par extraction du maximum : programmation

```
1  void triExtrac(float liste[]){
2      int indMax;
3      for (int i = liste.length-1; i >-1; i--){
4          indMax = i;
5          for (int j = i ; j>-1 ; j--) {
6              if (liste[j] > liste[indMax]){
7                  indMax = j;
8              }
9          }
10         if (indMax != i){
11             permuter(liste, indMax, i);
12         }
13     }
14 }
```

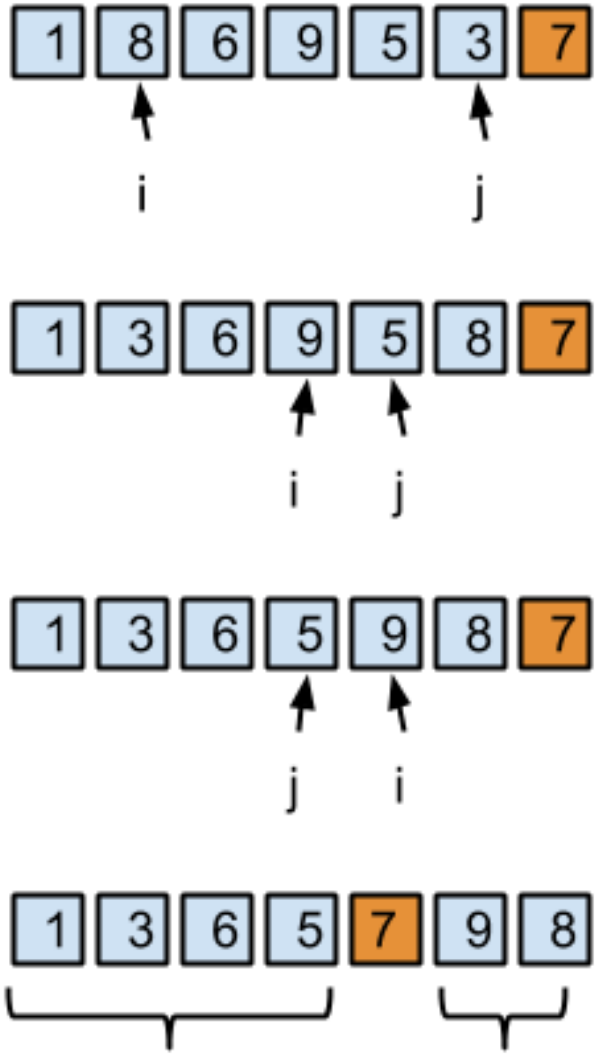
Tri par extraction du maximum : programmation

```
1  void setup(){
2    float[] maListe={1.2,5.8,6.3,89,3};
3    triExtrac(maListe);
4
5    for(int i=0; i<maListe.length; i++)
6    {
7      print(maListe[i]+" ");
8    }
9  }
```

Tri rapide

- Partage du tableau en deux.
- Tri de la partie gauche et de la partie droite.
- Algorithme récursif.

Tri rapide : exemple



Tri rapide : programmation

```
1  void triRapide(float tableau[] , int indDebut, int indFin){
2      int indPivot ;
3      if (indDebut < indFin){
4          indPivot = partitionnement(tableau, indDebut,
5                                                  indFin);
6          triRapide(tableau, indDebut, indPivot-1);
7          triRapide(tableau, indPivot+1, indFin );
8      }
9  }
```

Partitionnement (1/2)

```
 1  int partitionnement(float tableau[], int indDebut, int indFin)
        {
 2      float pivot;
 3      int i, j;
 4      pivot = tableau[indFin];
 5      i = indDebut;
 6      j = indFin - 1;
 7      while (i <= j){
 8          while ((i < indFin) && (tableau[i] <= pivot)){
 9              i++;
10          }
11          while ((j >= indDebut) && (tableau[j] >= pivot)){
12              j--;
13          }
```

Partitionnement (2/2)

```
1           if (i<j) {
2               permuter(tableau, i, j);
3           }
4       }
5       tableau[indFin] = tableau[i];
6       tableau[i] = pivot;
7       return i;
8   }
```

Tri rapide : programmation

```
1  void setup(){
2    float[] maListe={1.2,5.8,6.3,89,3};
3    triRapide(maListe,0,maListe.length-1);
4
5    for(int i=0; i<maListe.length; i++)
6    {
7      print(maListe[i]+" ");
8    }
9  }
```

Complexité d'algorithmes

Complexité en temps et/ou en espace :

- complexité en espace
  - volume de données nécessaire à l'exécution du programme
  - mémoire vive
  - sur disque dur
- complexité en temps
  - nombre moyen d'opérations lors de l'exécution du programme
  - notation $\mathcal{O}(f(n))$

Complexité des tris usuels

	Complexité moyenne	Complexité dans le pire des cas
Tri par insertion du minimum	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Tri par extraction du maximum	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Tri rapide	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$
Tri à bulles	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Tri fusion	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$

Complexité du tri rapide

Démonstration

Les objets

- On appelle *classes* des types complexes.
- Les classes sont définies par un *nom*, dont la première lettre est en majuscules, des *attributs* et des *méthodes*.
- On appelle une instance d'une classe un *objet*. D'où le nom *programmation orientée objet*.
- Les attributs sont des variables, les méthodes des fonctions, propres à la classe.
- Une méthode particulière est appelée le *constructeur*.

Les objets : exemple

```
1  class Geek {
2    String nom;
3    int qi;
4    Geek (String n, int q) {
5      nom = n;
6      qi = q;
7    }
8    void direBonjour() {
9      println("Bonjour, je suis "+ nom+" !");
10   }
11 }
12 void setup(){
13   Geek sc = new Geek("Sheldon Cooper",187);
14   sc.direBonjour();
15 }
```

Les objets : exemple

```
1  void harvard(Geek g){
2    if(g.qi>160){
3      println("Bienvenue  Harvard");
4    }
5    else{
6      println("Desole, votre niveau est insuffisant.");
7    }
8  }
9  void setup(){
10   Geek sc = new Geek("Sheldon Cooper",187);
11   Geek lol = new Geek("Georges Kikoolol",18);
12   harvard(sc);
13   harvard(lol);
14 }
```

Attention, cet exemple ne respecte pas le principe d'*encapsulation*.
Voir le cours de MPOO au second semestre !

Les objets : prenons du recul

- Comment invoque-t-on un attribut ?
- Comment invoque-t-on une méthode ?
- Pourquoi est-ce important que les noms de variables commencent par une minuscule ?

```
1  void setup(){
2    String s="toto";
3    println(s.length());
4
5    int[] tab = {1,2,3};
6    println(tab.length);
7  }
```

Entrées au clavier

Trois variables :

- **key** : variable de type char qui contient la valeur de la dernière touche appuyée
- **keyCode** : entier associé à chaque touche, permet de les différencier
- **keyPressed** : variable de type boolean qui indique si une touche est appuyée

Deux méthodes :

- **keyPressed()** : méthode exécutée suite à l'appui sur une touche
- **keyReleased()** : méthode exécutée si une touche est relachée

Entrées au clavier : les constantes prédéfinies

- **CODED** : pour toutes les touches spéciales, key est égal à la constante CODED (key == CODED est vrai).
- les valeurs de keyCode pour les tocuhes spéciales : LEFT, RIGHT, UP, DOWN, ALT, CONTROL, SHIFT...

Entrées au clavier : exemple

```
1  String s="";
2  int x=50;
3  int y=50;
4  /*
5  Notez que ces variables sont des variables globales !
6  */
7
8  void setup(){
9    size(200,200);
10 }
11
12 void draw(){
13   background(255);
14   fill(0);
15   text(s,x,y);
16 }
```

Entrées au clavier : exemple

```
1  void keyPressed(){
2    if(key==CODED){
3      if(keyCode==LEFT){
4        x -= 5;
5      }
6      else if(keyCode==RIGHT){
7        x += 5;
8      }
9      else if(keyCode==UP){
10       y -= 5;
11     }
12     else if(keyCode==DOWN){
13       y += 5;
14     }
15   }
16   else{
17     s=s+key;
```

Entrées à la souris

Trois variables :

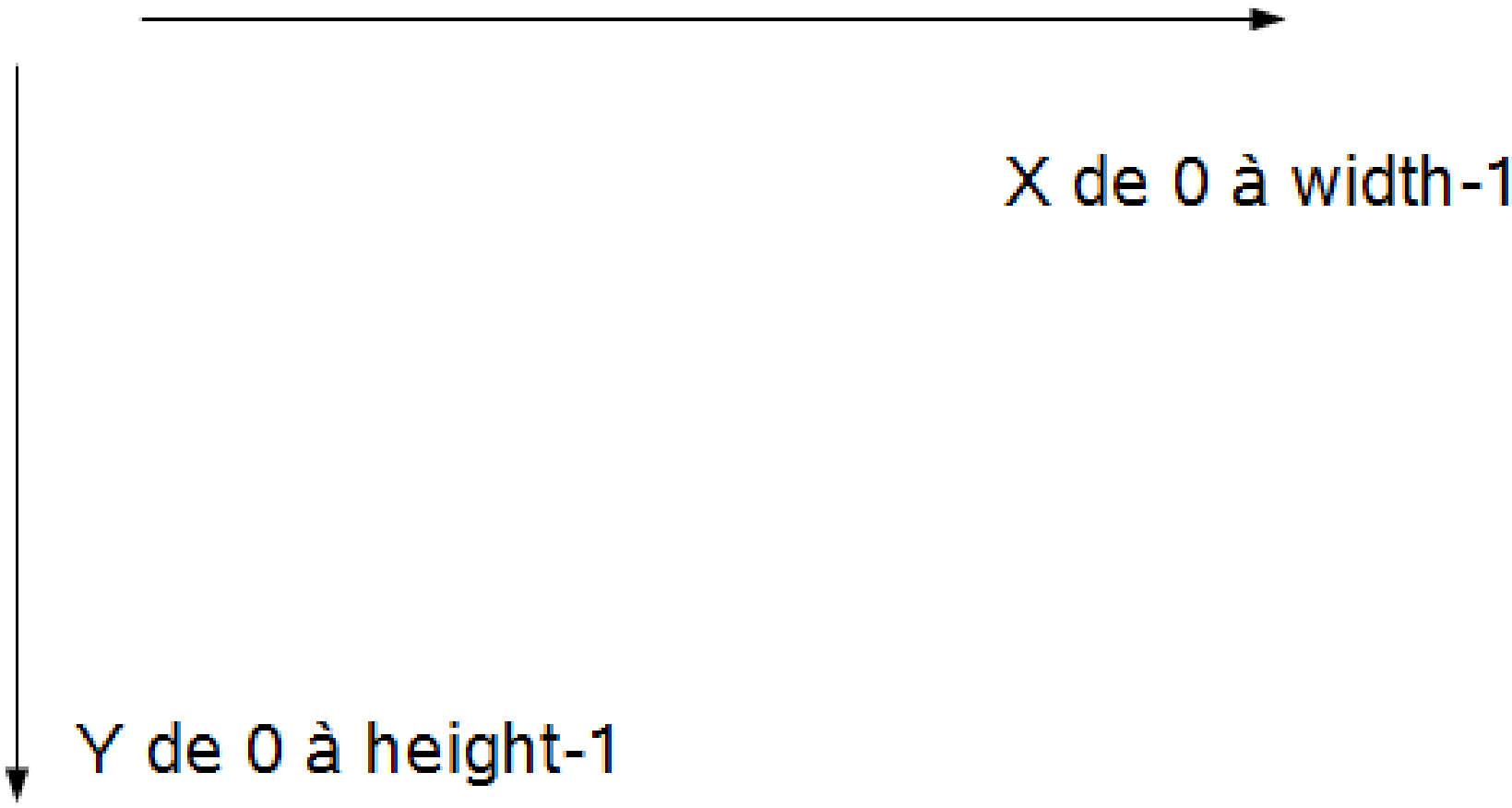
- **mouseX** : position de la souris en x
- **mouseY** : position de la souris en y
- **mouseButton** : identification du bouton appuyé
  - LEFT : bouton gauche
  - RIGHT : bouton droit
  - CENTER : bouton du milieu, s'il existe

Deux méthodes :

- **mousePressed()** : méthode exécutée si clic de souris
- **mouseReleased()** : méthode exécutée si souris relâchée

Le dessin : les coordonnées

Les coordonnées : x et y.



Le dessin : la fonction size

La fonction size définit les valeurs de height et width.

```
1  void setup() {
2    size(640, 360);
3  }
```

Le dessin : setup et draw

- draw est appelé en boucle.
- frameRate paramètre la fréquence d'appel à draw (par défaut, 60 fois par seconde).

```
1  int y = 100;
2  void setup() {
3    size(640, 360);
4    frameRate(30);
5  }
6  void draw() {
7    background(0);
8    y = y - 1;
9    if (y < 0) {
10     y = height;
11   }
12   line(0, y, width, y);
13 }
```

Le dessin : points, lignes, formes

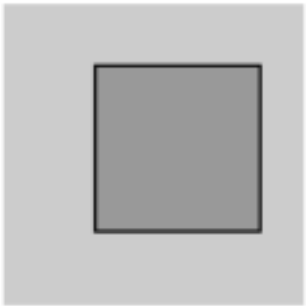
```
1  line(p3, p3, p2, p3);
2  point(p1, p1);
3  triangle(18, 18, 18, 360, 81, 360);
4  rect(81, 81, 63, 63);
5  quad(189, 18, 216, 18, 216, 360, 144, 360);
6  arc(479, 300, 280, 280, PI, TWO_PI);
```

Le dessin : autres

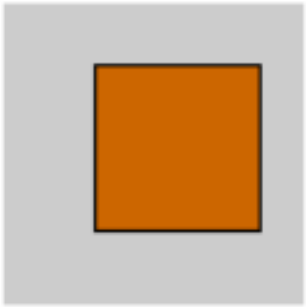
- Il est possible de faire simplement tout polynôme régulier, étoile, graphiques type camembert et bien d'autres...
- ...mais il faut le programmer soi-même.
- Essayez, vous avez le niveau !

Le dessin : les couleurs

- fill(v1,v2,v3) ;
- fill(v1, v2, v3, alpha) ;



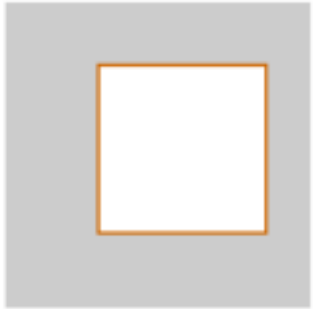
```
fill(153);
rect(30, 20, 55, 55);
```



```
fill(204, 102, 0);
rect(30, 20, 55, 55);
```

- stroke(v1,v2,v3) ;
- stroke(v1, v2, v3, alpha) ;

```
stroke(153);
rect(30, 20, 55, 55);
```



```
stroke(204, 102, 0);
rect(30, 20, 55, 55);
```

Le dessin : contrôler la transparence

```
1  tint(gray)
2  tint(gray, alpha)
3  tint(v1, v2, v3)
4  tint(v1, v2, v3, alpha)
5  tint(255, 127); // Afficher avec une opacite moyenne
```

Le dessin : charger et afficher des images

```
1  PImage img; // Declaration d un objet de type PImage
2  void setup() {
3    size(640, 360);
4    // L image doit etre dans le repertoire data.
5    img = loadImage("moonwalk.jpg");
6  }
7  void draw() {
8    // Affiche l image a sa vraie taille au point (0,0)
9    image(img, 0, 0);
10   // Affiche l image au point (0, height/2) reduite de moitie
11   image(img, 0, height/2, img.width/2, img.height/2);
12 }
```

Le dessin : aller plus loin

- Dégradés
- Courbes paramétrées
- Transformations
  - Tranlations
  - Rotations
- 3D
  - Coordonnées 3D
  - Effets de lumière
- Son
  - Musique
  - Bruits