DUT MMI – IUT de Marne-la-Vallée

M1202 - Algorithmique

Philippe Gambette

Page web du cours : http://tinyurl.com/M1202-2013S1

• Contact

- Courriel : philippe.gambette@gmail.com
  (M1202 dans le sujet du courriel)
- De vive voix avant ou après le cours/TD/TP

• Matériel

- Ordinateur portable : pas pendant les cours, à discuter pour les TD/TP.
- Pas de téléphone portable pendant cours/TD/TP
- Salles informatiques : n'y point manger, n'y point boire, ne pas y débrancher les câbles

• Déroulement des enseignements

- Séparation cours/TP/TD :
  → nouvelles méthodes de travail
  → distinguer ce qui est important, à retenir
  → savoir où retrouver l'information
- En général, distribution de notes de cours à compléter
- En général, distribution de corrigés :
  → refaire les exercices !

• Notes et devoirs

- Interrogations QCM en début de cours ou TD (signalement des absences pour rattrapage)
- Du travail à la maison

• Note finale

- Prévision : environ 2/3 "compétences"
  environ 1/3 "motivation"
- Compétences : 2/3 devoir final (8 janvier 2014)
  1/3 QCM
- Motivation : cours à compléter et/ou devoirs maison note générale de TP

• Exercices supplémentaires d'entraînement

- Sur demande, par courriel
- Sur demande, possibilité d'organiser une séance d'exercices ou de préparation au devoir final.

• Sources

Le livre de Java premier langage, A. Tasso
http://www.pise.info/algo/introduction.htm
Cours INF120 de J.-G. Luque
http://xkcd.com, http://xkcd.free.fr
http://serecom.univ-tln.fr/cours/index.php/Algorithmie
Cours de J. Henriet :
http://julienhenriet.olympe-network.com/Algo.html

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème : ("**étapes**" aussi appelées "**pas de l'algorithme**")
  → de façon structurée et compacte (méthode **facile à comprendre, facile à transmettre**)
  → à partir d'opérations de base (méthode a**daptée aux moyens à disposition, adaptée aux connaissances de celui qui l'utilise**)
  → indépendamment d'un langage de programmation (méthode **adaptée pour des problèmes qui se traitent sans ordinateur, compréhensible sans apprendre un langage de programmation**)

- Problème : données en entrée —*algorithme*→ résultat en sortie

Algorithmes sans ordinateurs :
- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations
- Ada Lovelace (1842) : calcul des nombres de Bernoulli sur la *machine analytique* de Charles Babbage

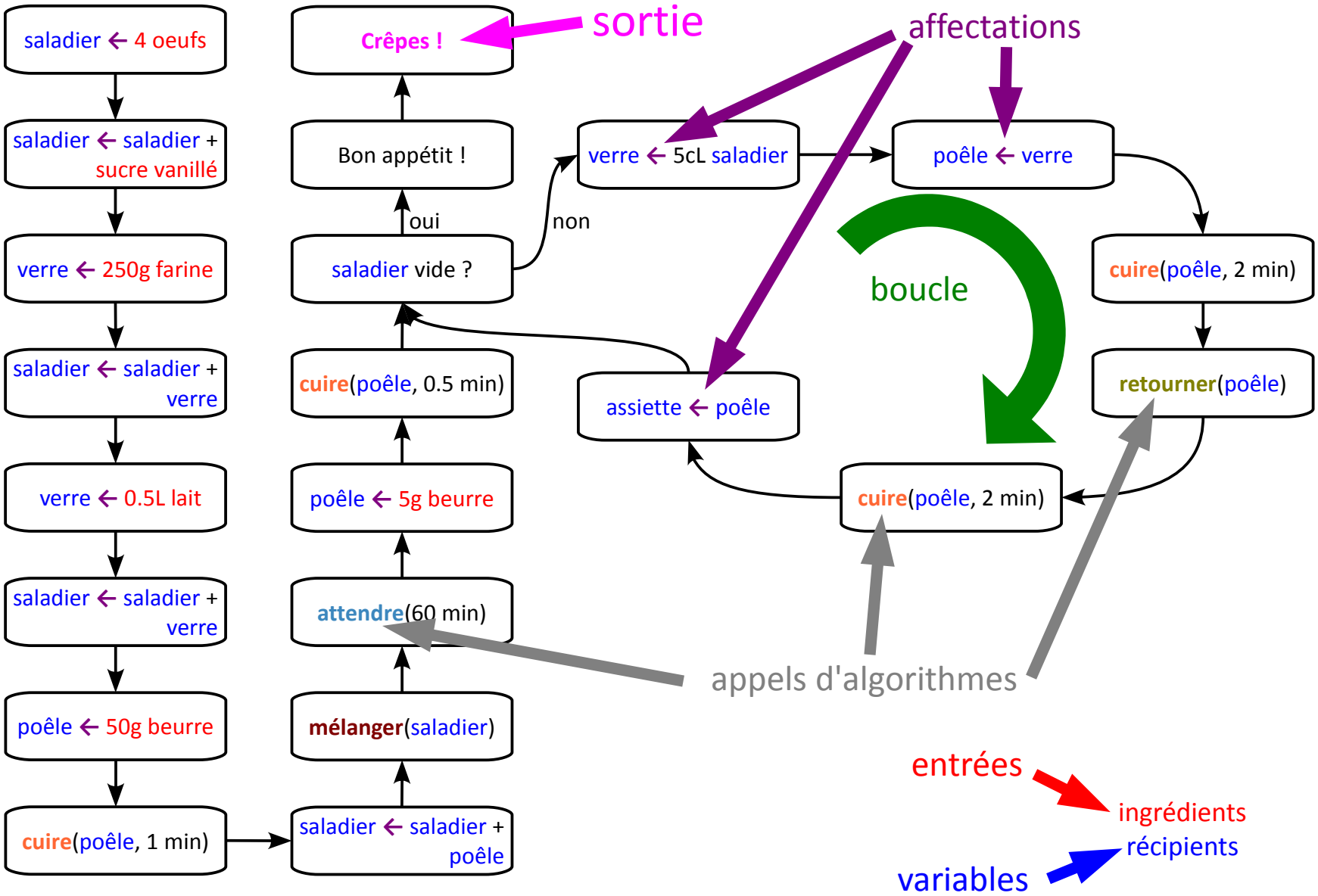
- *L' « algorithme des crêpes »*

Ingrédients : beurre, oeufs, sachets de sucre vanillé, farine, lait, sel
Récipients : saladier, verre mesureur, poêle, assiette,
Opérations de base : ***mettre dans un récipient***, ***mélanger***, ***attendre pendant … minutes***, ***retourner***, ***laisser cuire pendant ... minutes***

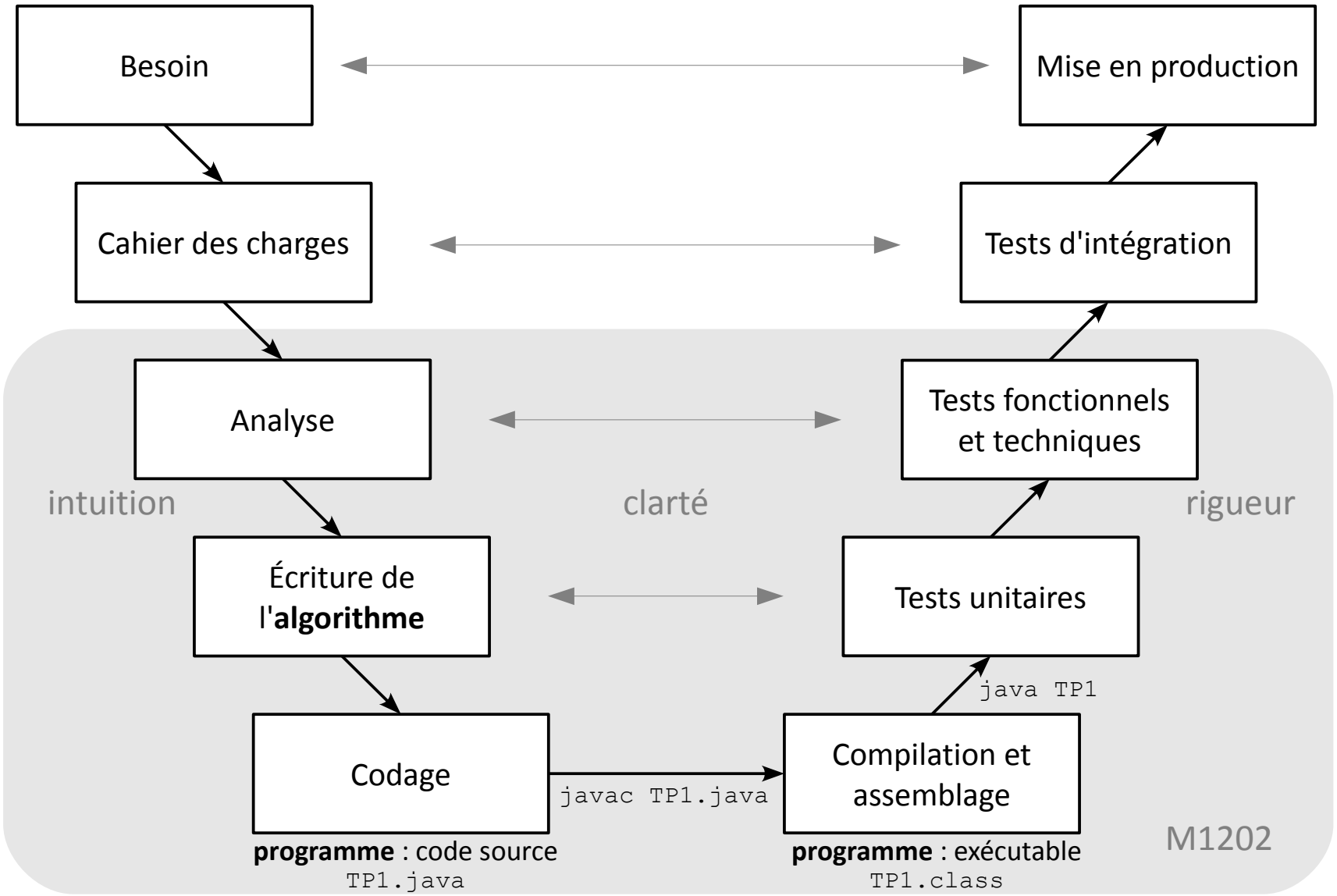
Algorithme des crêpes :

Mettre 4 oeufs **dans** le saladier
Mettre 1 sachet de sucre vanillé **dans** le saladier
Mettre 250 g de farine **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 0,5 litre de lait **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 50 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 1 **minute**
Mettre le contenu de la poêle **dans** le saladier
Mélanger le contenu du saladier
Attendre pendant 60 **minutes**
Mettre 5 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 0.5 **minute**
Tant que le saladier n'est pas vide :
  Mettre 5 cL du contenu du saladier **dans** le verre mesureur
  Mettre le contenu du verre mesureur **dans** la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Retourner le contenu de la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Mettre le contenu de la poêle **dans** l'assiette

Organigramme de la recette des crêpes



De l'algorithme au programme



Enjeux algorithmiques : correction, complexité

Algorithme correct ?

- donne le résultat attendu ? → **preuve de correction**

- quel que soit le type d'entrées ? → **débuggage, tests unitaires**

Preuve de correction :
- « invariant » : propriété vraie tout au long de l'algorithme
    - vraie à la première étape
    - si vraie à une étape, vraie à l'étape suivante
    - ⇨ vrai à la fin

En pratique, pour débuter :
- vérifier sur les “cas de base”
- vérifier sur des exemples aléatoires (attention aux exemples trop « simples »)

Algorithme rapide ?

- se termine ? → **preuve de terminaison**

- en combien de temps ? → **complexité**

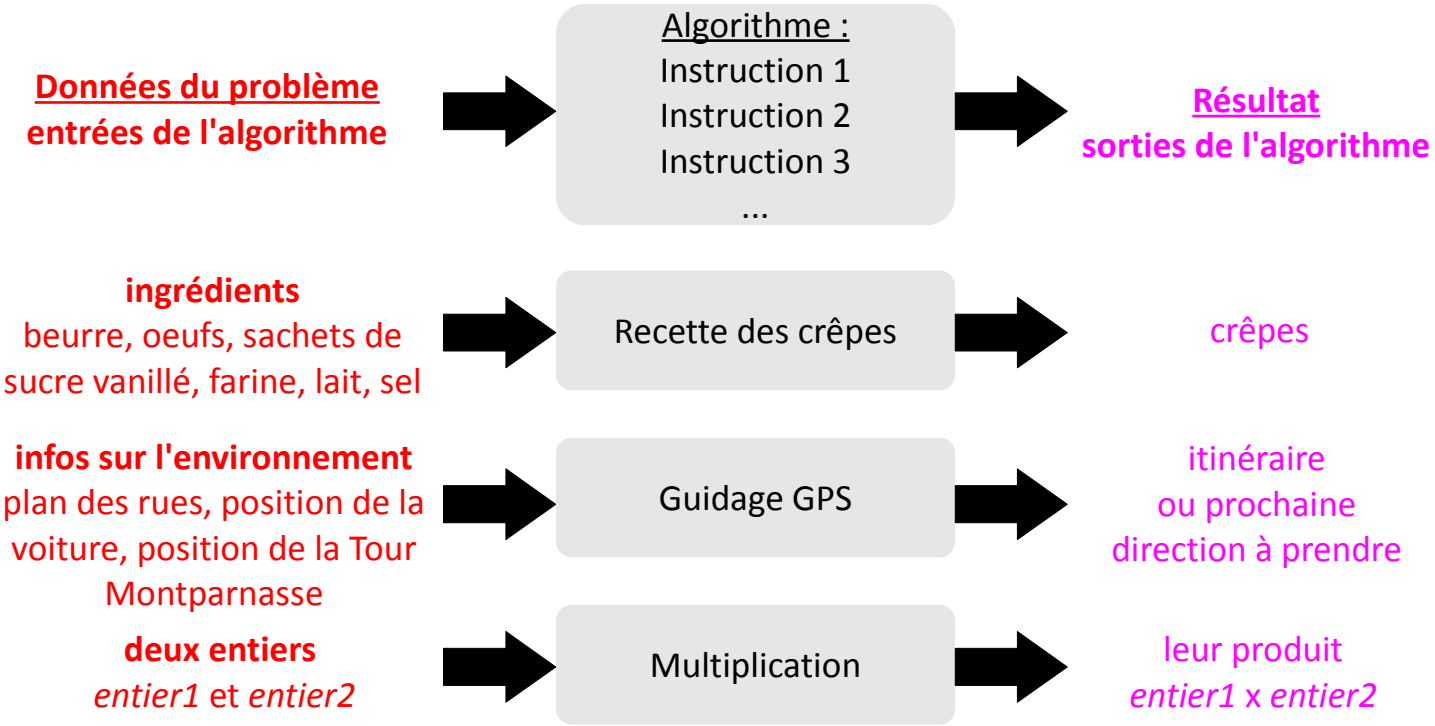
Preuve de terminaison :
L'algorithme des crêpes se termine-t-il ?
→ le saladier sera forcément vide à un moment donné (prouvable mathématiquement), donc **oui**.

Complexité : Combien de temps l'algorithme prend-il pour se terminer ?

Théorie de la complexité :
- nombre d'opérations en fonction de la taille du problème, dans le pire cas
- prouver qu'on ne peut pas utiliser moins d'opérations pour résoudre le problème, dans le pire cas

En pratique, pour débuter **:**
- vérifier sur des exemples aléatoires
- connaître les cas difficiles

Composants d'un algorithme



Divers **types d'instructions** d'un algorithme :
- déclaration d'un algorithme
- appel d'un algorithme
- déclaration d'une variable
- affectation d'une variable
- entrées / sorties
- boucle
- test

Premier programme Java

La **sortie** de l'algorithme `reponseALaQuestion` est "gambette" si l'utilisateur a entré ce nom

```
public static void main(String[]arg){
    String nom, reponse;
    int nombreATrouver, nombreUtilisateur;

    //Identification de l'utilisateur
    nom = reponseALaQuestion("Comment vous appelez-vous ?");

    Affiche(" ");
    Affiche("Bonjour "+nom+" !");

    //Choix du nombre aléatoire
    nombreATrouver = nombreAleatoire(1,10);
    Affiche("L'ordinateur a choisi un nombre entre 1 et 10.");
    Affiche("Essayez de le deviner.");

    //Premier essai de l'utilisateur
    reponse = reponseALaQuestion("Premier essai :");
    nombreUtilisateur = Integer.parseInt(reponse);

    if (nombreUtilisateur == nombreATrouver){
       Affiche("Bravo "+nom+", vous avez trouve !");
    }
}
```

entrées

La **sortie** de l'algorithme `nombreAleatoire` est 3 si l'ordinateur a choisi ce numéro au hasard

variables **affectations** **appel d'algorithme** **test**

Variables et affectation

Dans un algorithme, une **variable** possède :
• un **nom**,
• une **valeur**,
• un **type** (ensemble des valeurs que peut prendre la variable).

La **valeur** d'une variable :
• est **fixe à un moment donné**,
• peut **changer au cours du temps**.

L'**affectation** change la valeur d'une variable :
• *a* ← 5 (pseudo-code) / `a=5` (Java) :
- la variable *a* prend la valeur 5
- la valeur précédente est perdue (“écrasée”)
• *a* ← *b* (pseudo-code) / `a=b` (Java) :
- la variable *a* prend la valeur de la variable *b*
- la valeur précédente de *a* est perdue (“écrasée”)
- la valeur de *b* n'est pas modifiée
- *a* et *b* doivent être de même type
(ou de type compatible)

En revanche, le **nom** et le **type** d'une variable **ne changent pas**.
→ Dans un programme, **déclarer une variable** = fixer son type et son nom.
Pseudo-code : Variables : entiers a et b
Java : `int a,b;`

Dans un **algorithme**, choisir pour les variables :
• un nom composé de **lettres** et éventuellement de **chiffres**
• un nom **expressif**, par exemple :
- *chaine*, *requête1*... pour une chaîne de caractères
- *n*, *a*, *compteur*, *nbOperations*, *longueur*... pour un entier
- *x*, *y*, *température* pour un réel
- *estEntier*, *testEntier*, *trouvé*... pour un booléen
• un nom **assez court** (il faut l'écrire !)
• éviter les **noms réservés** : *pour*, *tant que*, *si*...

Dans un **programme** :
• **éviter** les lettres accentuées et la ponctuation
• préférer l'**anglais** si votre code source est diffusé largement
• être **expressif** et **lisible** :
- *est_entier* ou *estEntier* plutôt que *estentier*

Votre code sera relu, par vous ou par d'autres...

En **Java** le nom de variable doit commencer par une lettre, de préférence minuscule. Il peut contenir des lettres, des chiffres, et les symboles $ et _. Il est sensible à la casse (majuscules/minuscules).

Dictionnaire pseudo-code / Java

	Pseudo-code	Java

Déclaration d'un algorithme	**Addition**
	Entrées : entiers *i* et *j*
	Type de sortie : entier
	Début
	…
	renvoyer …
	Fin
		public static int **Addition**(int i, int j){
		…
		
		return …
		}

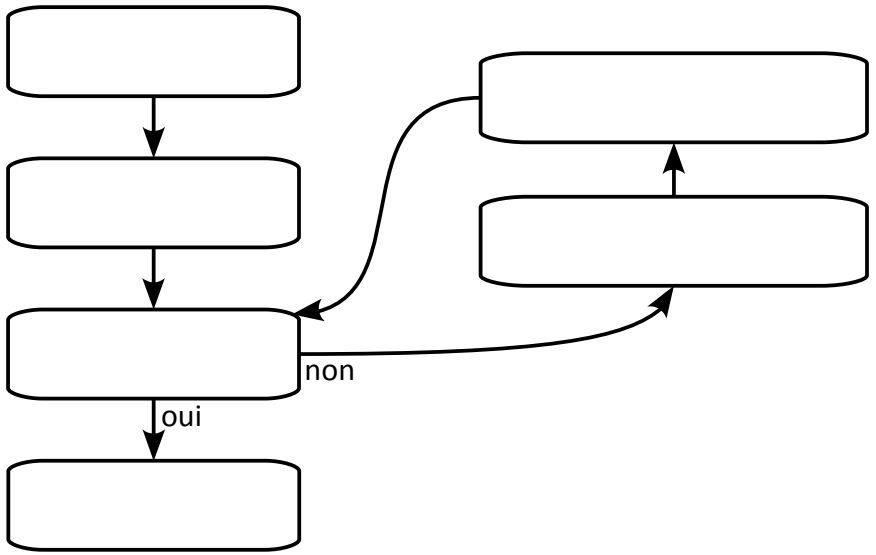
Déclaration d'une variable	Variables : entier *i*	int *i*;
Affectation	*i* ← 1	*i* = 1;

Test	Si *i*=1 alors
	…
	Sinon
	…
	FinSi
		if (i==1){
		…
		} else {
		…
		}

Boucle	Tant que *i*<3 faire
	…
	Fin TantQue
		while (i<3) {
		…
		}

De l'organigramme au code Java

organigramme



pseudo-code

DeuxPuissance :

Entrée :
Sortie :
Variables :
Début

renvoyer
Fin

code Java

public static int DeuxPuissance( ){

return
}

Trace (valeur de chaque variable après chaque instruction de l'algorithme) :

Codage binaire

La "minute votes SMS"

Programme Java :

```
import java.io.*;
public class Boucle{
     public static void main(String [] arg){
          int i=1;
          while(i>0){
               i=i*2;
          }
          System.out.print("J'ai fini !");
     }
}
```

Résultat du programme :

2	
4	131072
8	262144
16	524288
32	1048576
64	2097152
128	4194304
256	8388608
512	16777216
1024	33554432
2048	67108864
4096	134217728
8192	268435456
16384	536870912
32768	
65536	

La "minute mathématique"

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.

Exemple de nombre entier en binaire : 1101100001101

0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

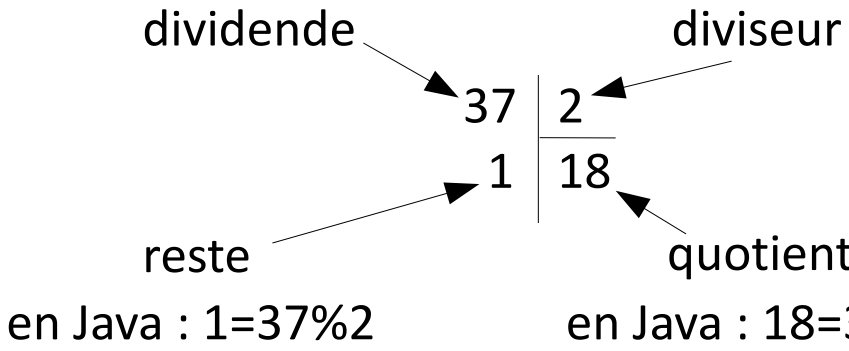
$2^{12} + 2^{11} + 2^{9} + 2^{8} + 2^{3} + 2^{2} + 2^{0} = 4096 + 2048 + 512 + 256 + 8 + 4 + 1 = 6925$

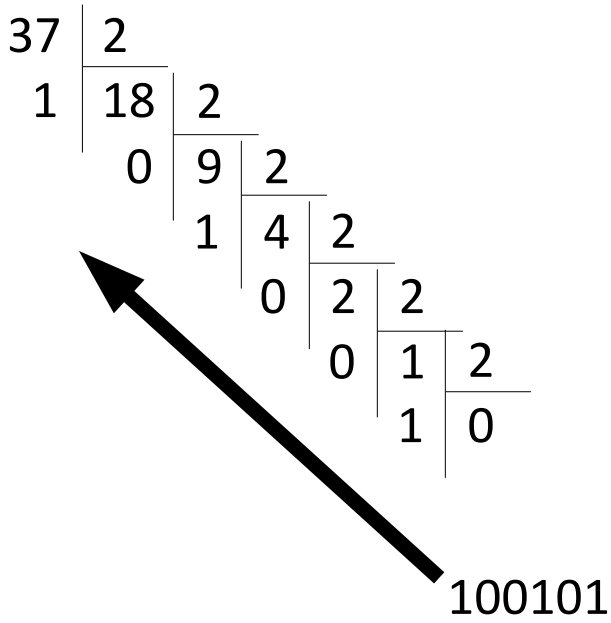
Codage binaire

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.

Exemple : écrire 37 en binaire ?

Division euclidienne :





Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}

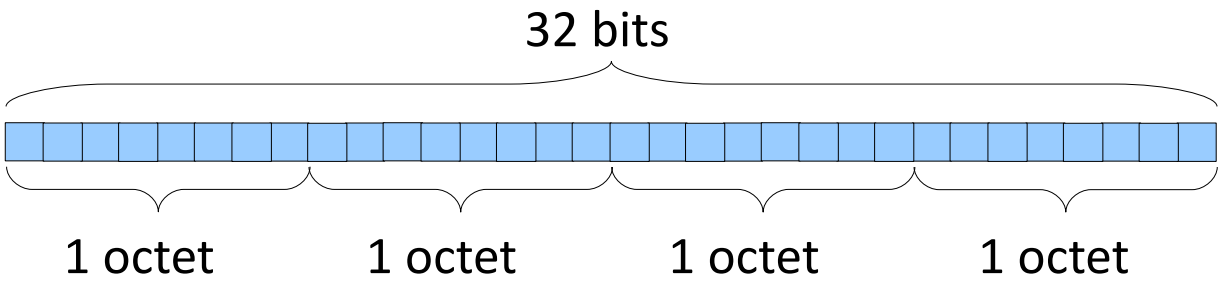
- faire des estimations de nombres données en binaire : $2^{10} = 1024$ donc $2^{10} \approx 1000$

$2^{32} \approx 4$ milliards

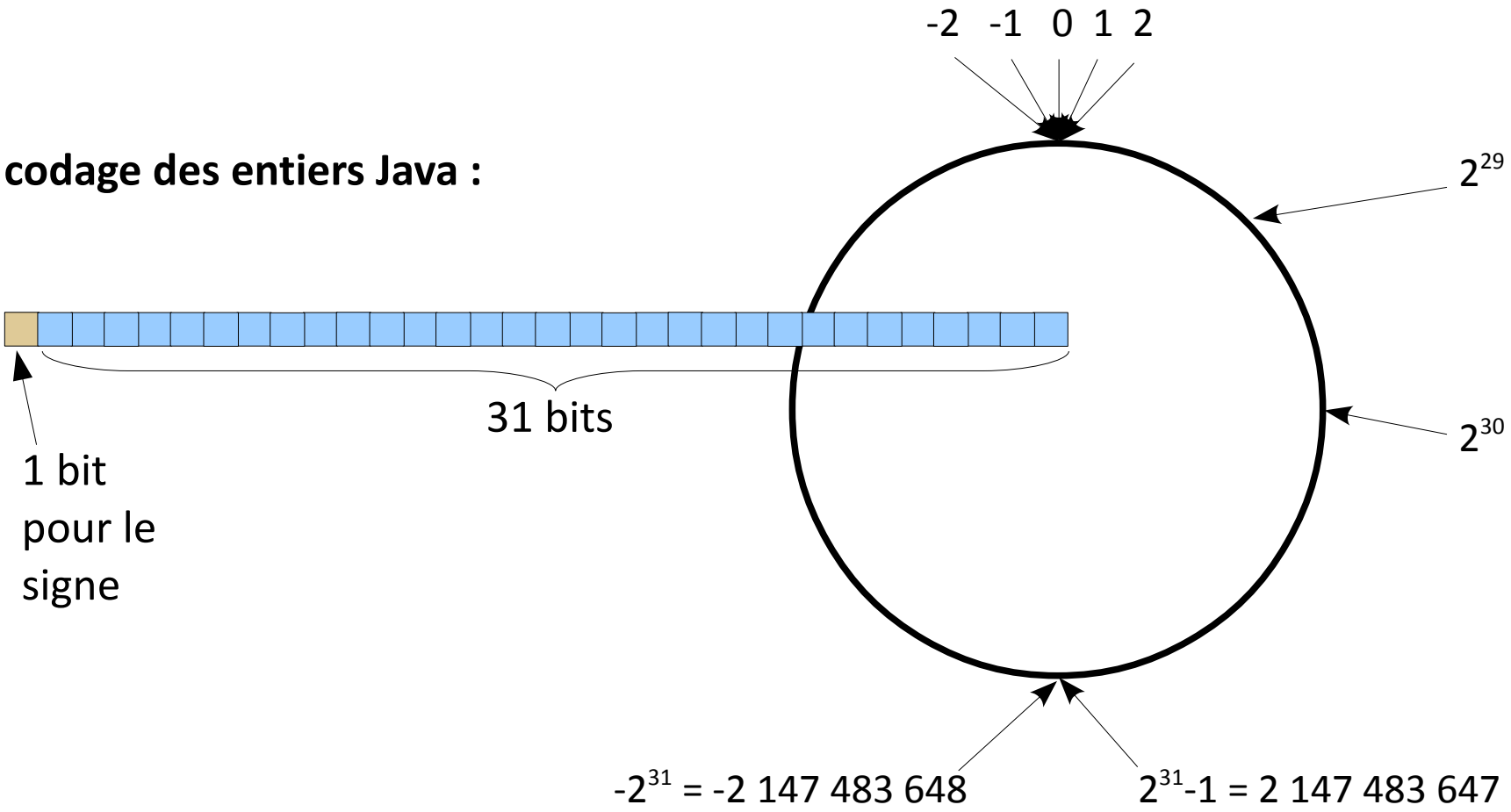
$2^{32} = 4\ 294\ 967\ 296$

Codage des entiers 32 bits

1 bit = 0 ou 1
1 octet = 8 bits
1 Ko (kilooctet) = 1024 octets
1 Mo (mégaoctet) = 1024 Ko (disquette)
1 Go (gigaoctet) = 1024 Mo (carte mémoire, 2h de vidéo en DivX)
1 To (téraoctet) = 1024 Go (disque dur externe)



Le codage des entiers Java :



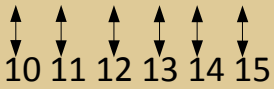
Autres codages

- **Chaînes de caractères**
  - ASCII : 7 bits, caractères simples codés de 32 à 127
  - ANSI : 8 bits, caractères simples codés de 32 à 127, caractères accentués de 128 à 255
  - UTF-8 : de 1 à 4 octets

- **Couleurs d'une image**
  - RGB : “red, green, blue”, 1 octet pour chacun :
    - valeurs entre 0 et 255
    - codage hexadécimal avec 2 symboles

La “minute culturelle”

Hexadécimal : en base 16 (ἑξάς : six, decem : dix)
Codé par les chiffres de 0 à 9 et les lettres A B C D E F



• Deux symboles pour un octet :
16^2 valeurs possibles = 256

• Utilisé pour coder les couleurs en HTML :
couleur="#RRGGBB"
rouge="#FF0000", vert="#00FF00"
#800080 ?

Les booléens

- **Opérations sur les booléens :**

et, ou, non

ET	VRAI	FAUX
VRAI	VRAI	FAUX
FAUX	FAUX	FAUX

OU	VRAI	FAUX
VRAI	VRAI	VRAI
FAUX	VRAI	FAUX

NON	VRAI	FAUX
	FAUX	VRAI

Les opérations de base en Java

- **Type entier `int`**

  `+`, `-`, `*`, `/` (division entière), `%` (reste modulo), `^` (puissance)

- **Type flottant `float`, ou `double` (plus précis)**

  `+`, `-`, `*`, `/`

- **Type booléen `boolean`**

  `false` (faux), `true` (vrai), `&&` (et), `||` (ou), `!` (non)

- **Type chaîne de caractères `String`**

  `+` (concaténation : `"INF"+"120"="INF120"`)

Les tableaux

Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

en pseudo-code

Variables : *tableau1*, un tableau d'entiers,
tableau2, un tableau de chaînes de caractères

Par exemple,

Un **tableau d'entiers** :

4	case 1
5	case 2
1	case 3
23	case 4
8	case 5
9	case 6

longueur 6

tableau1←**NouveauTableau**(6)
Case(*tableau1*,1)←4
Case(*tableau1*,2)←5
Case(*tableau1*,3)←1
Case(*tableau1*,4)←23
Case(*tableau1*,5)←8
Case(*tableau1*,6)←9

Un **tableau de chaînes de caractères** :

“chaine1”	case 1
“chaine2”	case 2
“blabla”	case 3
“toto”	case 4

longueur 4

longueur d'un tableau
= nombre de cases
Longueur(*tableau2*)

tableau2←**NouveauTableau**(4)
Case(*tableau2*,1)←”chaine1”
Case(*tableau2*,2)←”chaine2”
Case(*tableau2*,3)←”blabla”
Case(*tableau2*,4)←”toto”

Les tableaux

Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

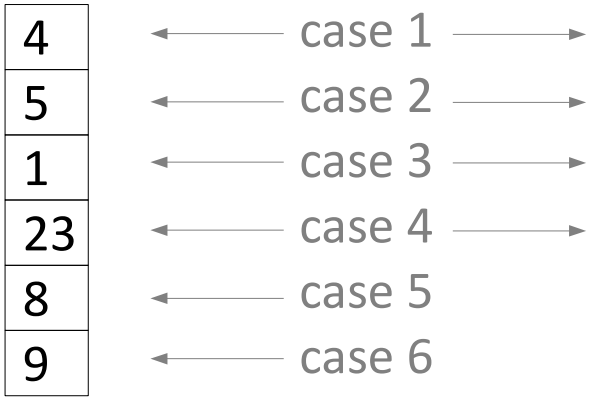
en Java

Déclaration des variables de tableau (tableau d'entiers, et tableau de chaînes de caractères) :

```
int[] tableau1; String[] tableau2;
```

Par exemple,

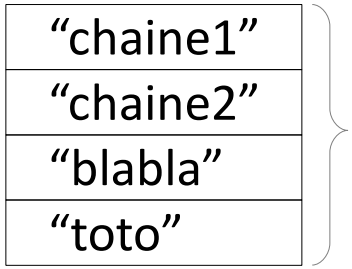
Un **tableau d'entiers** :



longueur 6

```
tableau1=new int[6];
tableau1[0]=4;
tableau1[1]=5;
tableau1[2]=1;
tableau1[3]=23;
tableau1[4]=8;
tableau1[5]=9;
```

Un **tableau de chaînes de caractères** :



longueur d'un tableau
= nombre de cases

```
tableau2.length
```

longueur 4

```
tableau2=new String[4];
tableau2[0]=”chaine1”;
tableau2[1]=”chaine2”;
tableau2[2]=”blabla”;
tableau2[3]=”toto”;
```

Attention, cases du tableau `t` numérotées de `0` à `t.length-1` en Java.

Les tableaux

Pour lire le contenu d'un tableau...
il faut une **boucle pour aller lire chaque case** !

Si le tableau a été prévu trop court au début, **impossible de changer sa longueur**... il faut une boucle pour le recopier dans un tableau plus grand !

Possibilité de créer des **tableaux de tableaux**...

L'algorithme de base, le **parcours du tableau**, pour visiter chaque case :

Variables : tableau d'entiers *tab*, entier *i*
i ← 1
Tant que *i* < **Longueur**(*tab*)+1 faire :
    [des choses avec la *i*-ième case du tableau **Case**(*tab*,*i*)...]
    i ← *i*+1
Fin Tant que

Affichage du contenu d'un tableau d'entiers

Algorithme **AfficheTableau**
Variable d'entrée :
Variable :
Début

Fin

```
public static       AfficheTableau(         tableau1){



}
```

Graphique du nombre d'apparitions des mots dans un texte

J'ai cueilli ce brin de bruyère

L'automne est morte souviens-t'en

Nous ne nous verrons plus sur terre

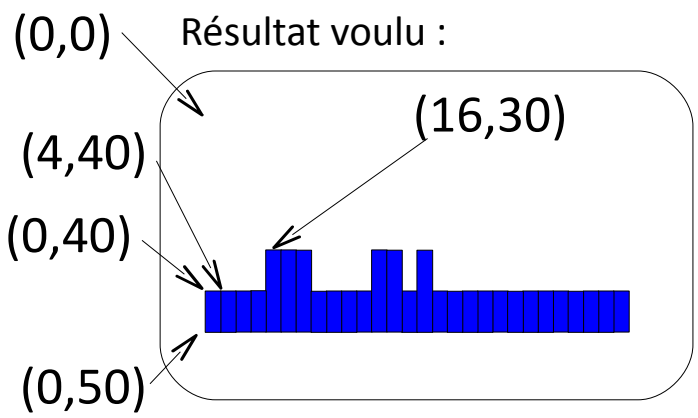
Odeur du temps brin de bruyère

Et souviens-toi que je t'attends

j	1
ai	1
cueilli	1
ce	1
brin	2
de	2
bruyère	2
l	1
automne	1
est	1
morte	1
souviens	2
t	2
en	1
nous	2
ne	1
verrons	1
plus	1
sur	1
terre	1
odeur	1
du	1
temps	1
et	1
toi	1
que	1
je	1
attends	1

tableau de chaînes de caractères *Mots*

tableau d'entiers *NbApparitions*



Graphique du nombre d'apparitions des mots dans un texte

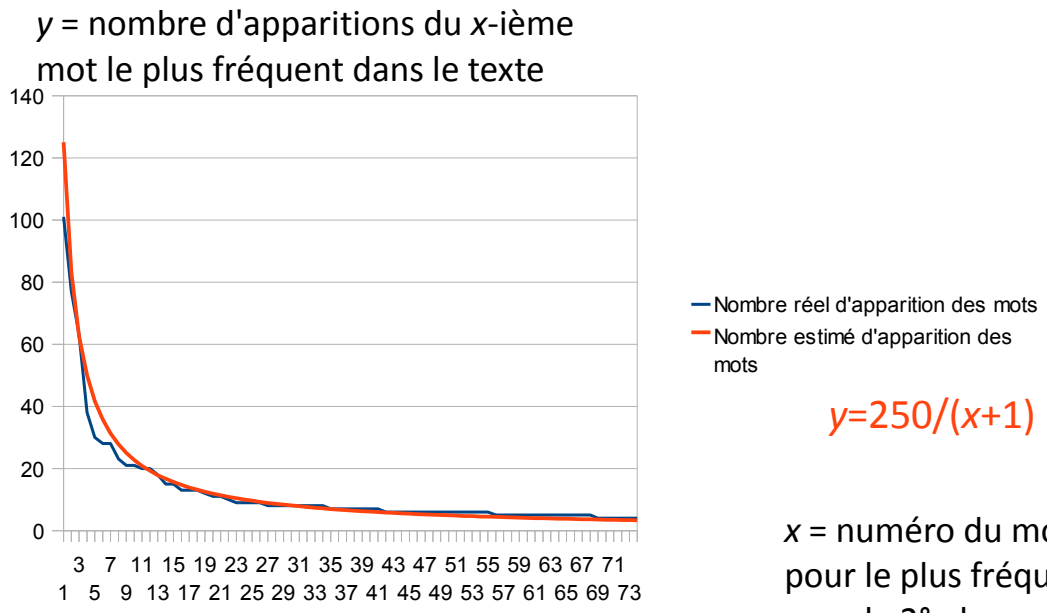
Algorithme **DessineHistogramme**
Entrée : tableau de chaînes de caractères *Mots* et tableau d'entiers *NbApparitions*.
Variable : entier *i*
Début
i ← 1
Tant que faire :

i ← 1 + *i*
Fin TantQue
Fin

Le nombre d'apparitions d'un mot dans un texte

La "minute mathématique"

La loi de Zipf prédit la courbe du nombre d'apparitions des mots les plus fréquents d'un texte.



$y=250/(x+1)$

x = numéro du mot (1 pour le plus fréquent, 2 pour le 2° plus fréquent...)

Fonctionne pour n'importe quel texte assez long...

La boucle “for” / “Pour tout...”

La boucle “for” / “Pour tout”

Une boucle pour **parcourir tous les entiers entre deux valeurs entières**.

Algorithme **DessineHistogramme**
Entrée : tableau de chaînes de caractères *Mots* et tableau d'entiers *NbApparitions*.
Variable : entier *compteur*
Début
compteur ← 1
Tant que *compteur* < **Longueur**(*Mots*)+1 faire :
    dessineRectanglePlein(*compteur**4-4,
        50-10***Case**(*NbApparitions*,*compteur*),
        4,10***Case**(*NbApparitions*,*compteur*),
        couleurRGB(0,0,255))
    compteur ← 1 + *compteur*
Fin TantQue
Fin

Algorithme **DessineHistogramme**
Entrée : tableau de chaînes de caractères *Mots* et tableau d'entiers *NbApparitions*.
Variable : entier *compteur*
Début
Pour *compteur* de 1 à **Longueur**(*Mots*) faire :
    dessineRectanglePlein(*compteur**4-4,
        50-10***Case**(*NbApparitions*,*compteur*),
        4,10***Case**(*NbApparitions*,*compteur*),
        couleurRGB(0,0,255))
    compteur ← 1 + *compteur*
Fin Pour
Fin

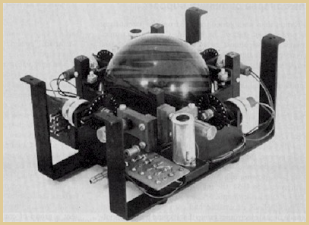
En Java :

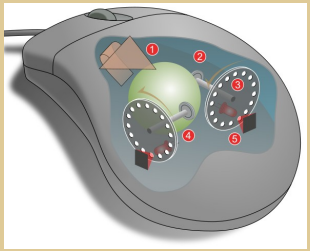
```
int compteur;
compteur=1;
while(compteur<mots.length+1) {
    …
}
```

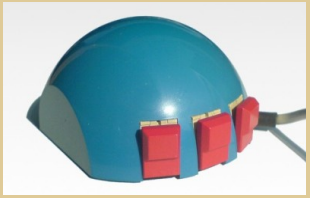
déclaration + initialisation — *condition d'arrêt* — *mise à jour*

```
for(int compteur=1;compteur<mots.length+1;compteur++){
    …
}
```

Souris et autres périphériques d'entrée







La "minute culturelle"

L'invention de la souris

1952 Trackball (boule de commande)
Tom Cranston et Fred Longstaff
(Marine Royale Canadienne)

1963 Souris mécanique
Douglas Engelbart et Bill English
(Stanford Research Institute)

1977 Souris optique
Jean-Daniel Nicoud et André Guignard
(Ecole polytechnique fédérale de Lausanne)

Crédits :
http://en.wikipedia.org/wiki/File:DATAR_trackball.jpg
http://cerncourier.com/cws/article/cern/28358/1/cernbooks2_12-00
Jeremykemp, Pbroks13,
http://fr.wikipedia.org/wiki/Fichier:Mouse_mechanism_diagram.svg
Stéphane Magnenat (User:Nct)
http://en.wikipedia.org/wiki/File:SmakyMouseAG.jpeg

Entrées-sorties dans la **communication ordinateur – utilisateur**
Quel **type de données** utiliser en **algorithmique** pour coder les entrées-sorties ?

Périphérique	**Type de données transmises**
Clavier	chaîne de caractères
Souris à 1 bouton	deux entiers (abscisse et ordonnée) + un booléen (clic ou pas)
Webcam	image, donc tableau de tableaux de couleurs RGB
Kinect	image + tableau de tableaux d'entiers (profondeur)
Ecran	Si ligne de commande : chaîne de caractères Si interface graphique : image, donc tableau de tableaux de couleurs RGB

La "minute xkcd" : 243

Instructions d'entrée-sortie en Java et pseudo-code

Entrées clavier

en pseudo-code

chaîne de caractères

reponseALaQuestion(*questionAAfficher*)
affiche la question *questionAAfficher* et renvoie une chaîne de caractères.

Exemple : **reponseALaQuestion**("Quel est votre nom") me laisse taper mon nom au clavier et renvoie "Gambette"

en Java

```
Scanner lectureClavier = new Scanner(System.in);
String stringLu =
```

LectureClavier.next() renvoie une chaîne de caractères.

```
int intLu =
```

LectureClavier.nextInt() renvoie un entier.

Sorties écran (ligne de commande)

en pseudo-code

chaîne de caractères

Affiche(*chaineAAfficher*)
affiche la chaîne de caractères *chaineAAfficher* et ne renvoie rien.

en Java

```
String chaineAAfficher="blabla";
System.out.println(chaineAAfficher);
```

affiche la chaîne de caractères *questionAAfficher*, puis retourne à la ligne, mais ne renvoie rien.

Les fonctions

La "minute mathématique"

fonction	exemple	entrées possibles	sortie
cosinus	cosinus(1.047)=0.5		
somme	somme(2,3)=5		
opposé	opposé(4)=-4		
inverse	inverse(10)=0.1		
différence	différence(2,3)=-1		
estPositif	estPositif(-5)=FAUX		
partieEntière	partieEntière(5.6)=5		
moyenne	moyenne(2,4,6)=4		
min	min({6,2,4,3})=2		

Attention : point au lieu de virgule dans les flottants (pour différencier de la virgule qui sépare les entrées d'une fonction)

Les fonctions

Les **fonctions** = les **algorithmes**
Zéro, un ou plusieurs paramètres en entrée...
Une sortie (ou aucune : `void` en Java)...

En Java :

Toujours une fonction **`main`** **qui ne renvoie rien** (type de sortie : `void`)
qui prend en entrée les paramètres du programme (tableau `arg`)
et qui est **la seule fonction exécutée quand on exécute le programme**.

Déclaration des autres fonctions **après** la fonction `main` :
ne s'exécutent que si elles sont appelées pendant l'exécution de la fonction `main`

Les paramètres du programme :
Si on exécute dans le terminal la commande
`java TP1 toto 1 10.5`
alors la variable `arg` contient le tableau de chaînes de caractères :

Visibilité des variables : toute variable déclarée à l'intérieur d'une fonction n'est valable **que dans cette fonction** et **ne peut pas être utilisée ailleurs**.
→ Variables « **locales** »

```
public class TP1{
   public static void main(String[] arg){
      int i,j;
      i=5;
      j=34;
      System.out.print("i+1="+inc(i)+",i="+i+",
         j="+j+", somme : "+addition(i,j));
   }
   public static int inc(int i){
      i=i+1;
      return i;
   }
   public static int addition(int i, int j){
      return i+j;
   }
}
```

Méthodo : Connaître les éléments de base d'un algorithme

en pseudo-code

Un algorithme résout un problème, en fournissant un **résultat en sortie** à partir de **données en entrée**.

Entrées : entiers *i* et *j*

Type de sortie : entier

une instruction par ligne !

Pour cela, il utilise plusieurs types d'**instructions** :

- des **affectations** dans des **variables** (mémoires) produit ← entier1+produit (variable / valeur)
- des **boucles** Tant que … faire : … Fin TantQue
- des **appels** à d'autres algorithmes Addition(3,5) — entrées de l'algorithme entre parenthèses, respectant l'ordre de déclaration des entrées
- des **tests** Si ... alors : ... sinon : … FinSi
- des "**lectures**" d'entrées et "**renvois**" de sorties

renvoyer *i*+*j*

Chaque **variable** a un **nom**. On doit :

- la **déclarer** en définissant son **type** (ensemble de valeurs possibles)
- puis l'**initialiser** (lui donner une **valeur** par une affectation)

avant de l'utiliser.

Variables : entiers *a* et *b* (type)

a ← 2

b ← *a*+2

Méthodo : Connaître les éléments de base d'un algorithme

en Java

Un algorithme résout un problème, en fournissant un **résultat en sortie** à partir de **données en entrée**.

public static int Addition(int i, int j){...}
type de sortie — entrées précédées de leur type

Pour cela, il utilise plusieurs types d'**instructions** :

instructions finies par ";"

- des **affectations** dans des **variables** (mémoires)

produit = entier1+produit
variable — valeur

- des **boucles** while(…){...}
- des **appels** à d'autres algorithmes Addition(3,5)

entrées de l'algorithme entre parenthèses, respectant l'ordre de déclaration des entrées

- des **tests** if(...){...}else{...}
- des "**lectures**" d'entrées et "**renvois**" de sorties

return i+j;

Chaque **variable** a un **nom**. On doit :

- la **déclarer** en définissant son **type** (ensemble de valeurs possibles)
- puis l'**initialiser** (lui donner une **valeur** par une affectation)

avant de l'utiliser.

int a, b;
type

a = 2
b = a+2

Méthodo : Lire et comprendre un algorithme

Premiers éléments à identifier :
- qu'est-ce que l'algorithme **prend en entrée** ? **Combien** de variables, de quel **type** ?
- qu'est-ce que l'algorithme **renvoie en sortie** ? **Rien** ? Ou bien **une** variable ? De quel **type** ?

Ensuite :
- quels sont les autres **algorithmes appelés** par l'algorithme ?

Enfin :
- faire la **trace** de l'algorithme, c'est-à-dire l'essayer sur un **exemple**
(… ou plusieurs pour passer au moins une fois par toutes les instructions de l'algorithme)
et voir ce que valent **toutes les variables à chaque étape** (et noter ces valeurs dans un tableau contenant une ligne par variable et une colonne par étape),
- noter en particulier le **résultat obtenu en sortie** pour une **entrée testée.**

Méthodo : Concevoir un algorithme

Premiers éléments à identifier :
- quels sont les **outils à disposition ?** (pour ces outils : données en entrée, type de données en entrée, résultat en sortie, type de résultat en sortie, résultat attendu sur un exemple...)
- quel est le **comportement attendu** pour mon algorithme ? (données en entrée, type de données en entrée, résultat en sortie, type de résultat en sortie, résultat attendu sur un exemple...)

Ensuite, résoudre le problème en utilisant ces outils :
- comment résoudre le problème **étape par étape** ? (essayer sur l'exemple testé)
- est-ce que les **outils à disposition** sont **utilisables** pour réaliser chaque étape ?

Enfin :
- comment **structurer** l'utilisation des outils à disposition ? (**combinaison** des différents outils à l'intérieur de structure de **boucles**, de **tests**, utilisation d'un **organigramme**...)
- comment **décomposer** le problème ? (et **reformuler** chaque sous-problème pour le résoudre avec les outils à disposition, écrire un algorithme par sous-problème)

+ fiche méthodo : Vérifier un algorithme !