DUT MMI – IUT de Marne-la-Vallée

M1202 - Algorithmique

Philippe Gambette
Samy Dindane

Page web du cours : http://tinyurl.com/M1202-2018S1

- **Contact**

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

- **Matériel**

- Ordinateur portable : - pas pendant les cours a priori ;
  - 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 (les demander s'ils ne sont pas disponibles) :
  → **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 (26 novembre 2018)
  1/3 tests
- Motivation : cours à trous à compléter
  travail maison + note générale de TP
  bonus pour erreurs détectées

- **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
Cours INF120 de J.-G. Luque
http://www.pise.info/algo/introduction.htm
JavaScript et les données du Web, R. Jeansoulin
http://serecom.univ-tln.fr/cours/index.php/Algorithmie
Cours de J. Henriet :
http://julienhenriet.olympe-network.com/Algo.html

Sommaire

Introduction
p. 3 A quoi sert un algorithme ?
p. 4 Organigramme de la recette des crêpes
p. 5 De l'algorithme au programme
p. 6 Enjeux de l'algorithmique : complexité, correction

Premiers algorithmes
p. 7 Composants d'un algorithme
p. 8 Premier programme Javascript
p. 9 Variables et affectation
p. 10 Dictionnaire pseudo-code / Javascript
p. 11 De l'organigramme au code Javascript

Les types en Javascript
p. 12 Codage binaire
p. 14 Codage des entiers 32 bits
p. 15 Autres codages
p. 16 Les booléens
p. 17 Les opérations de base en Javascript

Les tableaux
p. 18 Les tableaux en pseudo-code
p. 19 Les tableaux en Javascript
p. 20 Manipulation de tableaux
p. 21 Affichage du contenu d'un tableau d'entiers
p. 22 Graphique du nombre d'apparitions des mots dans un texte
p. 24 La boucle for/pour tout

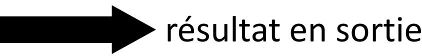
Périphériques d'entrée/sortie
p. 25 Souris et autres périphériques d'entrée
p. 26 Instructions d'entrée/sortie en Javascript et pseudo-code

Les fonctions
p. 27 Introduction aux fonctions
p. 28 Les spécificités de Javascript sur les fonctions
p. 29 Exemples de fonctions

Concepts et compétences du cours M1202
p. 31 Concepts
p. 32 Compétences

À 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 

 résultat en sortie

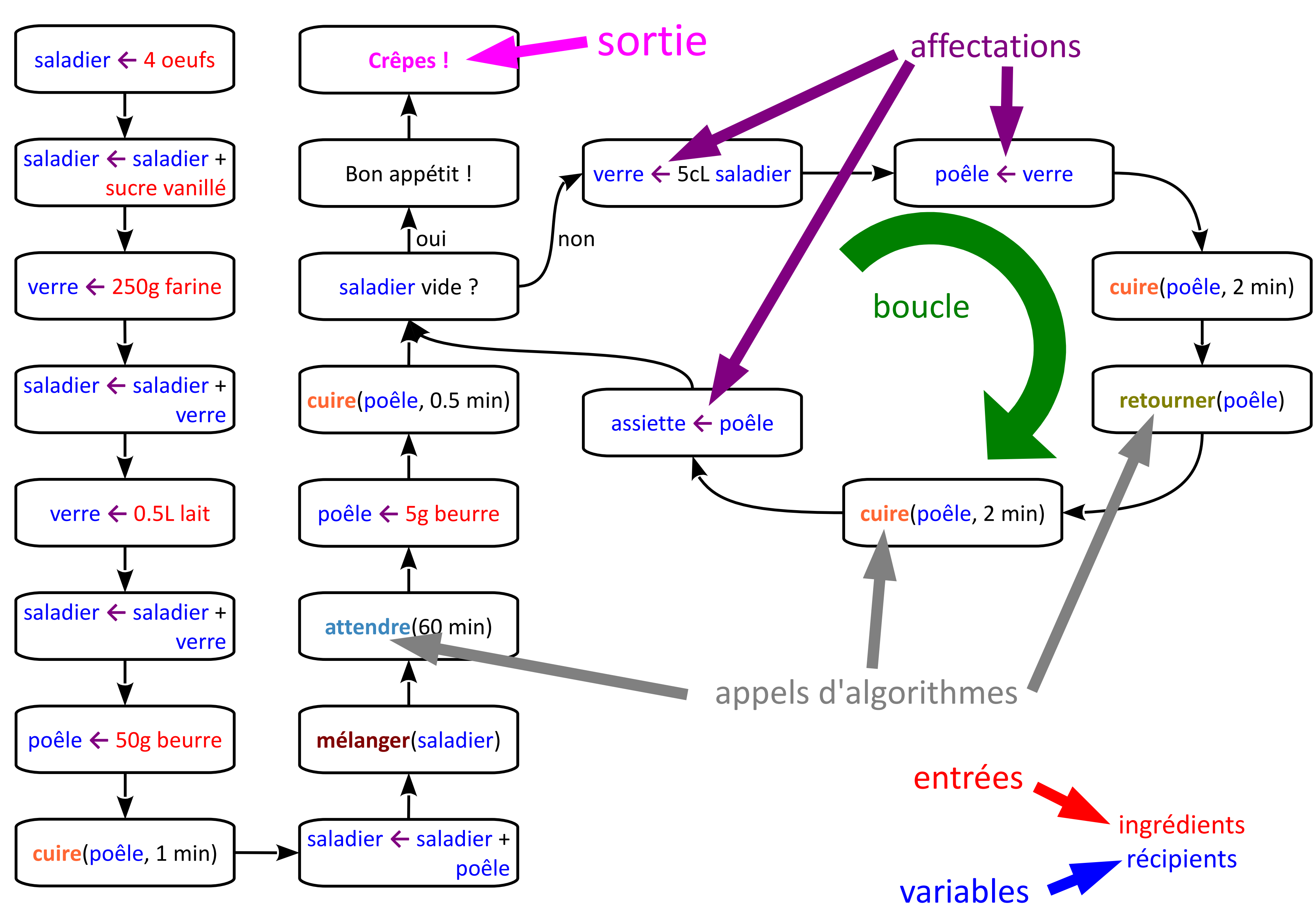
Algorithmes sans ordinateur :
- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations
- Ada Lovelace (1842) : calcul des nombres de Bernoulli sur la *machine analytique* de Charles Babbage

- *L' « algorithme des crêpes »*

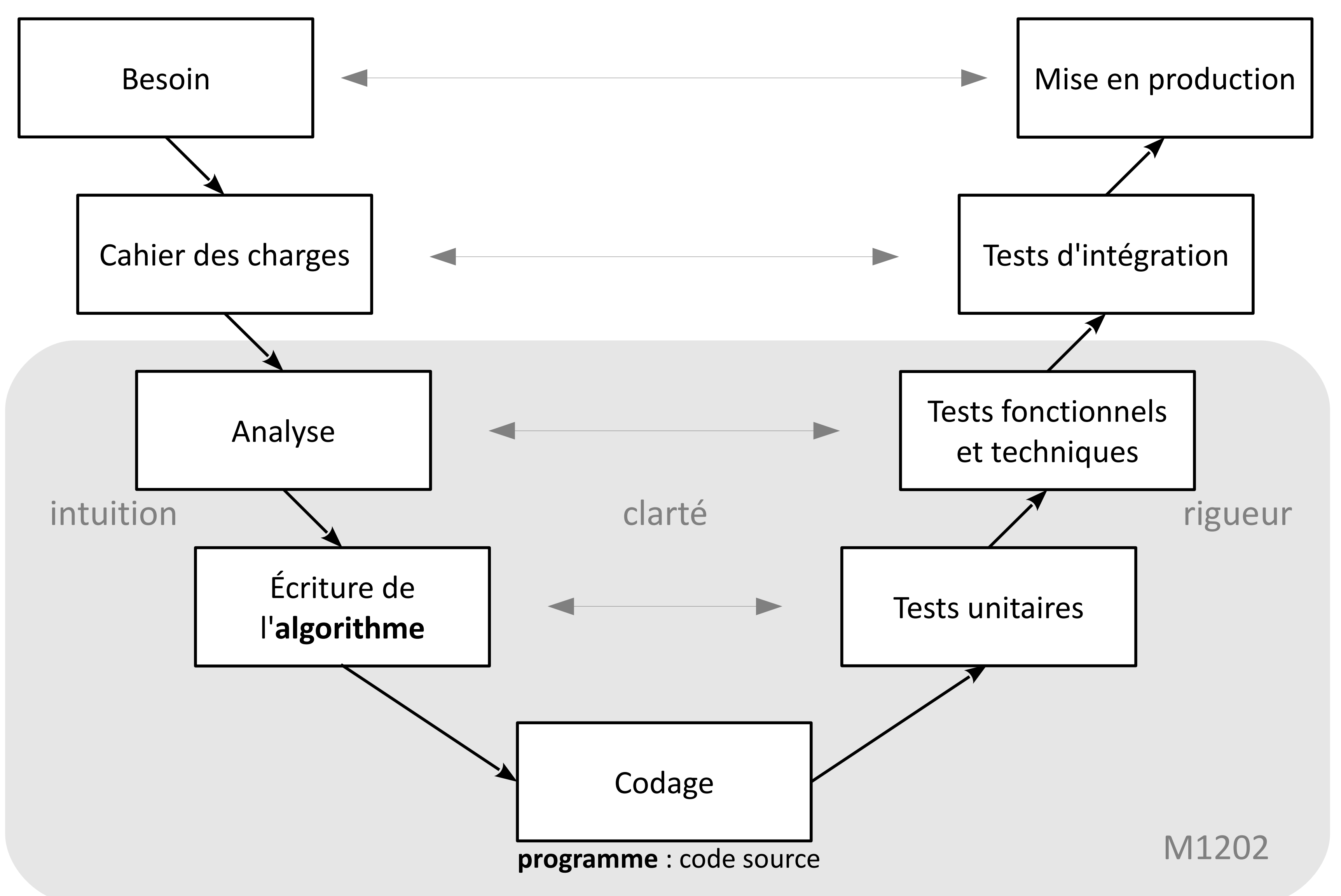
Ingrédients : beurre, oeufs, sachets de sucre vanillé, farine, lait, sel
Récipients : saladier, verre mesureur, poêle, assiette,
Opérations de base : ***mettre dans un récipient***, ***mélanger***, ***attendre pendant … minutes***, ***retourner***, ***laisser cuire pendant … minutes***

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 de l'algorithmique : 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**

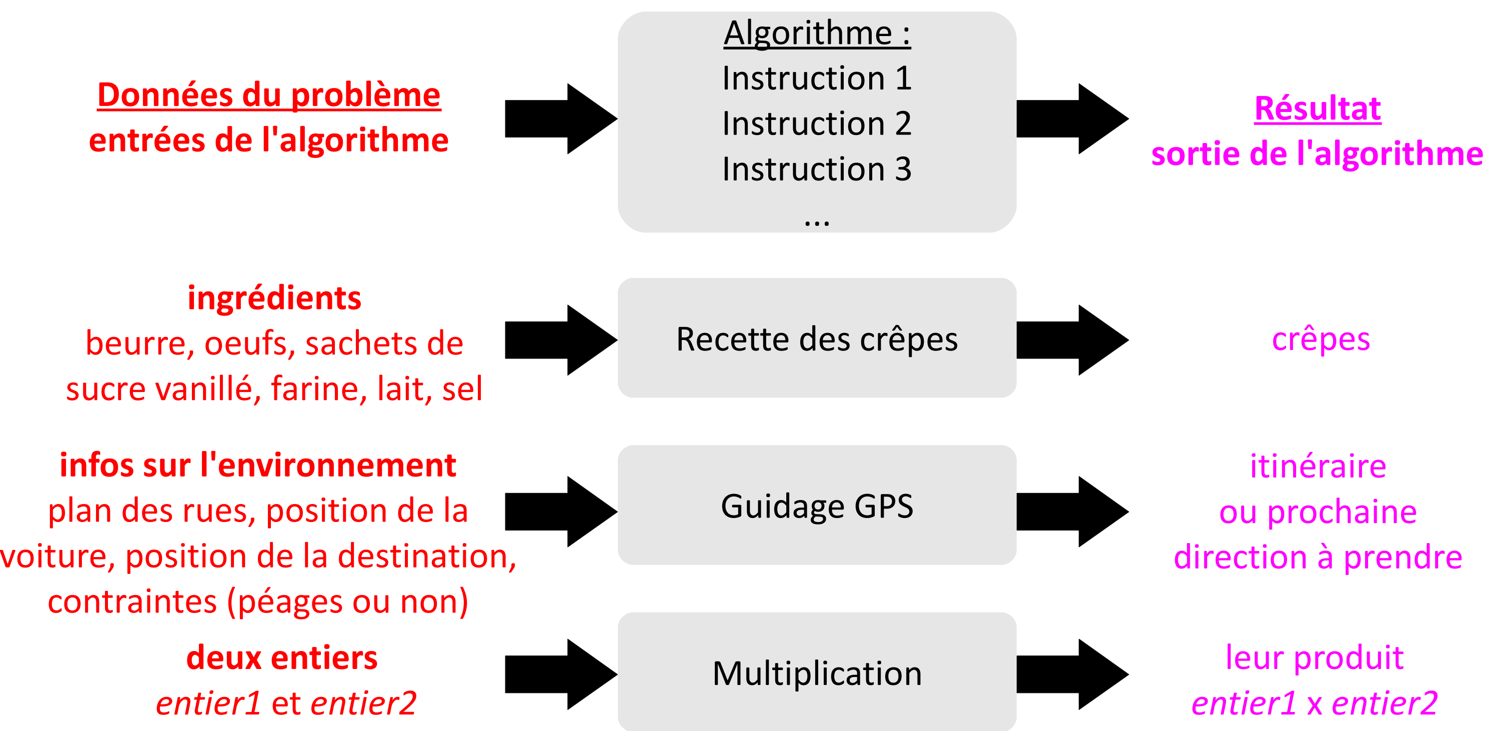
Preuve de terminaison :
L'algorithme des crêpes se termine-t-il ?
→ le saladier sera forcément vide à un moment donné (prouvable mathématiquement), donc **oui**.

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'utilisatrice ou utilisateur a entré ce nom

```
async function fonctionPrincipale(){
   //Identification de l'internaute
   var nom = await reponseALaQuestion("Comment vous
            appelez-vous ?");
   afficheConsole("Nom entré : "+nom)

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

   //Choix du nombre aléatoire
   var 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
   var reponse = await reponseALaQuestion("Premier essai :");
   var nombreUtilisateur = convertitEnEntier(reponse);

   if (nombreUtilisateur == nombreATrouver){
      affiche("Bravo "+nom+", vous avez trouvé
      du premier coup !");
   }
}
```

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` (Javascript) :
  - la variable *a* prend la valeur 5
  - la valeur précédente est perdue ("écrasée")
• *a* ← *b* (pseudo-code) / `a=b` (Javascript) :
  - 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* devraient être de même type
    (ou de type compatible)

En revanche, le **nom** d'une variable **ne change pas**.
Il est **possible mais pas recommandé** de changer son **type**.
→ Dans un programme Javascript, **déclarer une variable** = fixer son nom.
Pseudo-code : Variables : entiers a et b
Javascript : var `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 **Javascript** 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).
Il ne peut pas correspondre à un nom réservé du langage (if, for, etc.)

Dictionnaire pseudo-code / Java

	Pseudo-code	Javascript

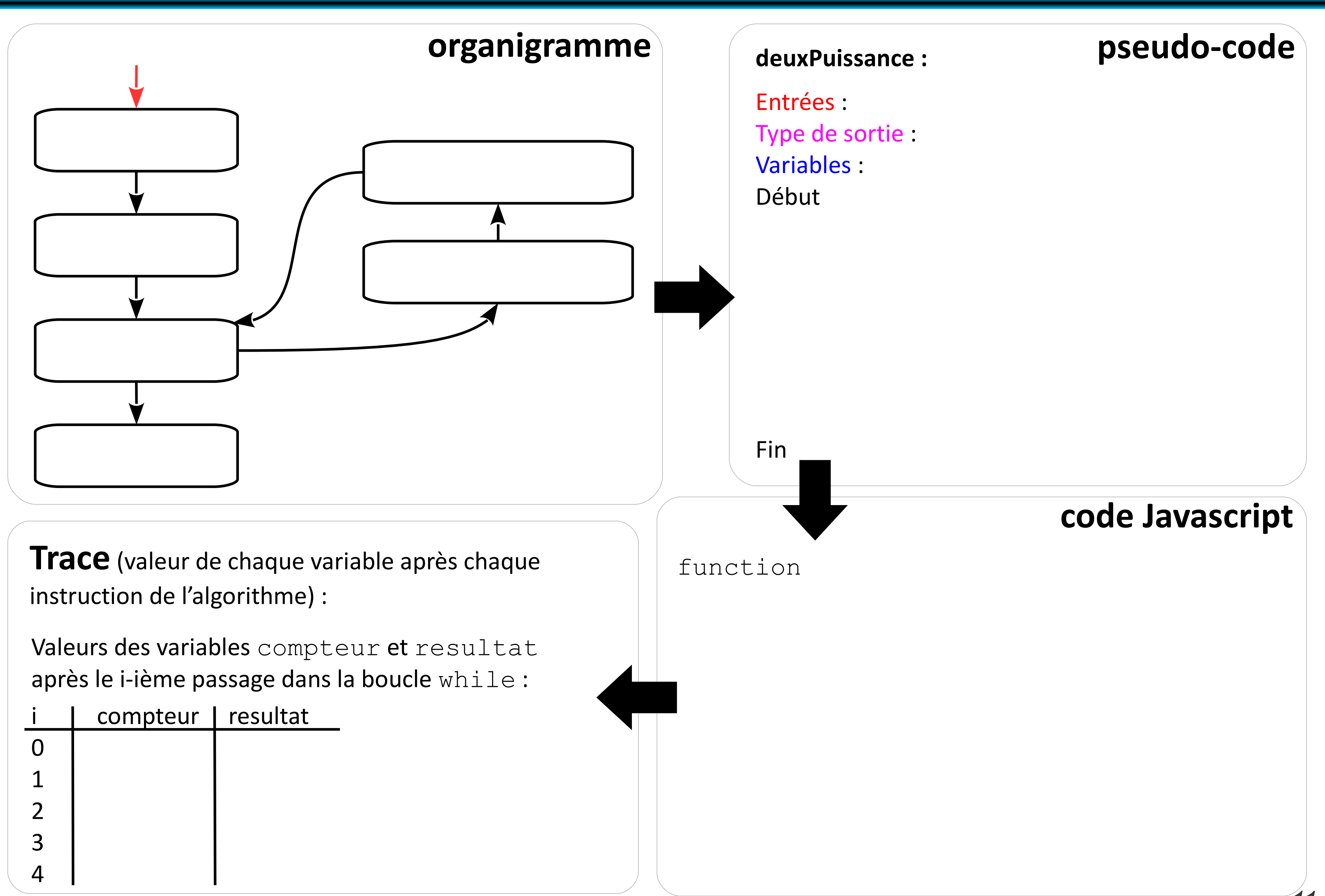
Déclaration d'un algorithme	**addition**
	Entrées : entiers *i* et *j*
	Type de sortie : entier
	Début
	…
	renvoyer …
	Fin
		function **addition**(i, j){
		…
		return …
		}

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

Test	Si *i*=1 alors
	…
	Sinon
	…
	Fin Si
	Le bloc du « Sinon » est facultatif
		if (i==1){
		…
		} else {
		…
		}
		Le bloc du « else » est facultatif

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

De l'organigramme au code Javascript



Codage binaire

La “minute votes SMS”

Programme Javascript :

```
var i=2;
var k=0;
while(k<10){
   i=i*i;
   k=k+1;
   // on affiche i dans la console
   // avec console.log :
   console.log(i)
}
if(i<i+1){
   console.log("Tout va bien.");
} else {
   console.log("i n'est pas inférieur à i+1 !?");
}
```

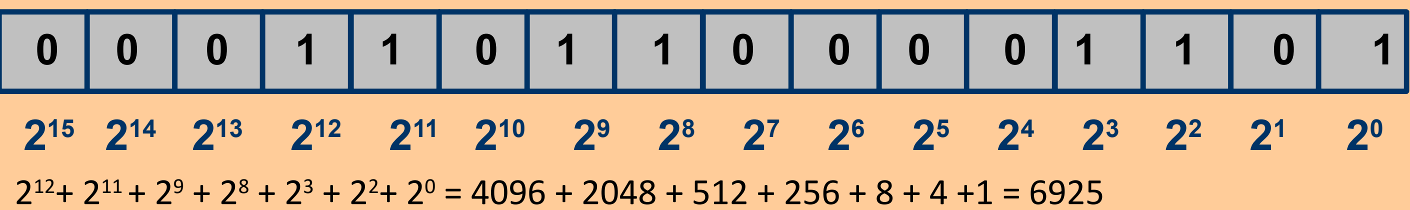
Affichage dans la console :

4
16
256
65536
4294967296
18446744073709552000
3.402823669209385e+38
1.157920892373162e+77
1.3407807929942597e+154

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

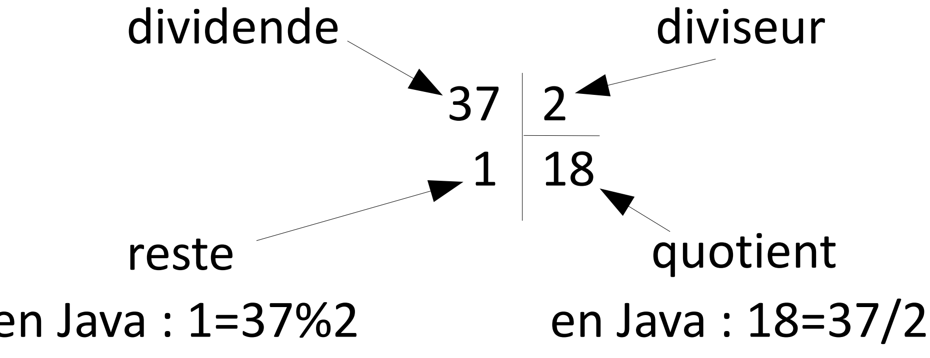


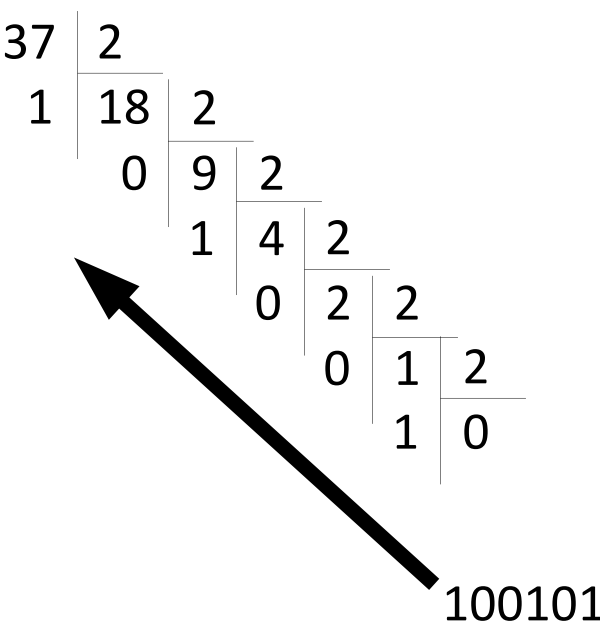
Codage binaire

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 :

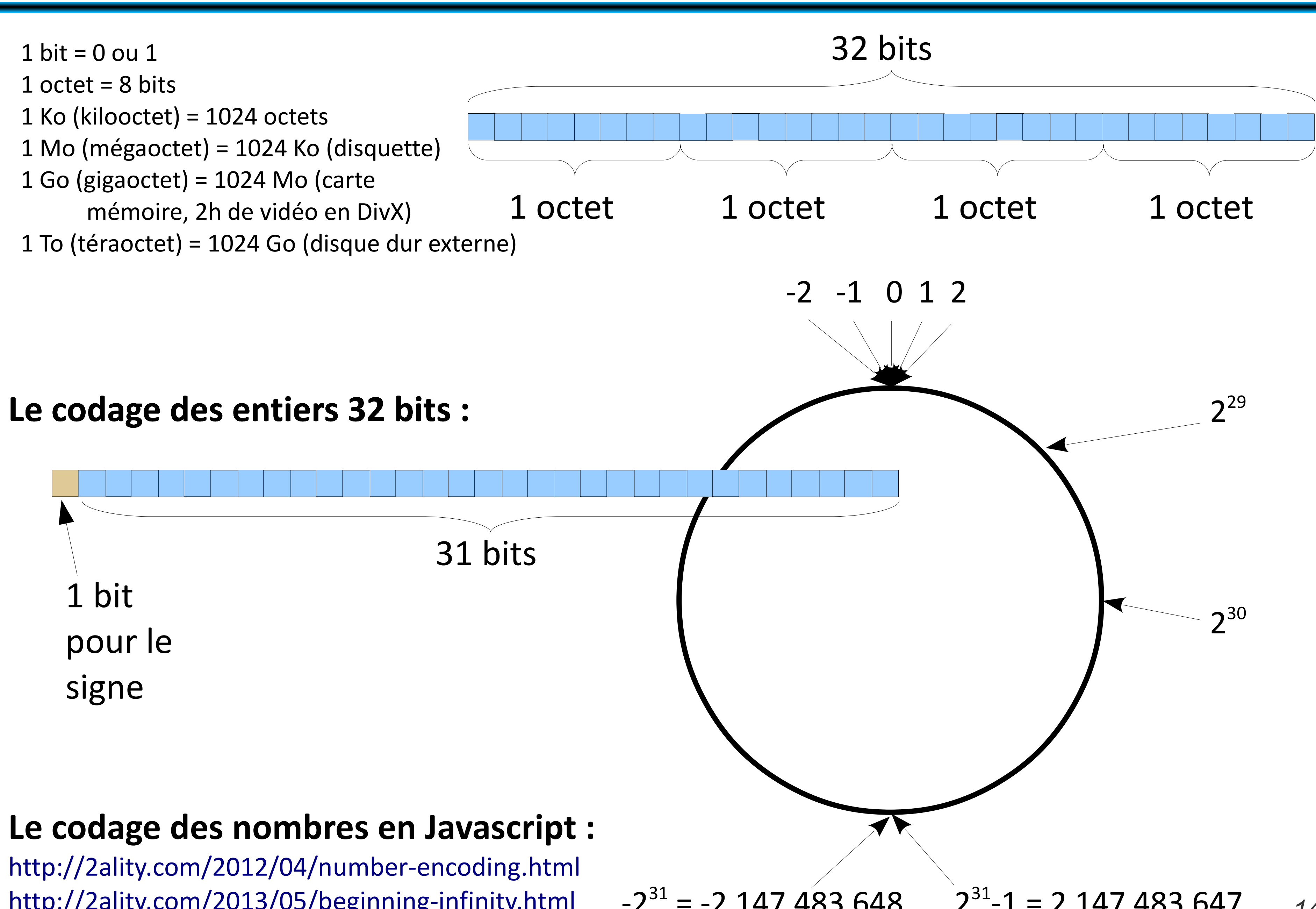
- compter sur ses doigts en binaire, jusqu'à 2^{10}

- 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



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

A	B	C	D	E	F
↕	↕	↕	↕	↕	↕
10	11	12	13	14	15

- 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

- Une **égalité** est un **booléen** : (i=4) est soit VRAI, soit FAUX
- Une **inégalité** est un **booléen** : (i>10) est soit VRAI, soit FAUX

→ **on peut placer utiliser les opérations sur les booléens pour des inégalités, des égalités, etc.**
Exemples : Si (i=4) OU (i>10) alors … / Si NON (i=4) alors …

Les opérations de base en Javascript

- **Type nombres à virgule** (`float` en anglais)

`+` (addition), − (soustraction), `*` (multiplication), `/` (division),
`%` (reste modulo), `**` (puissance), == (égalité), < et > (inégalité stricte),
<= et >= (inégalité large), != (non égalité)

- **Type booléen** (`boolean` en anglais)

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

- **Type chaîne de caractères** (`string` en anglais)

`+` (concaténation : `"M"+"1202"` est équivalent à `"M1202"`,
tout comme `"M"+1202`)

Les tableaux en pseudo-code

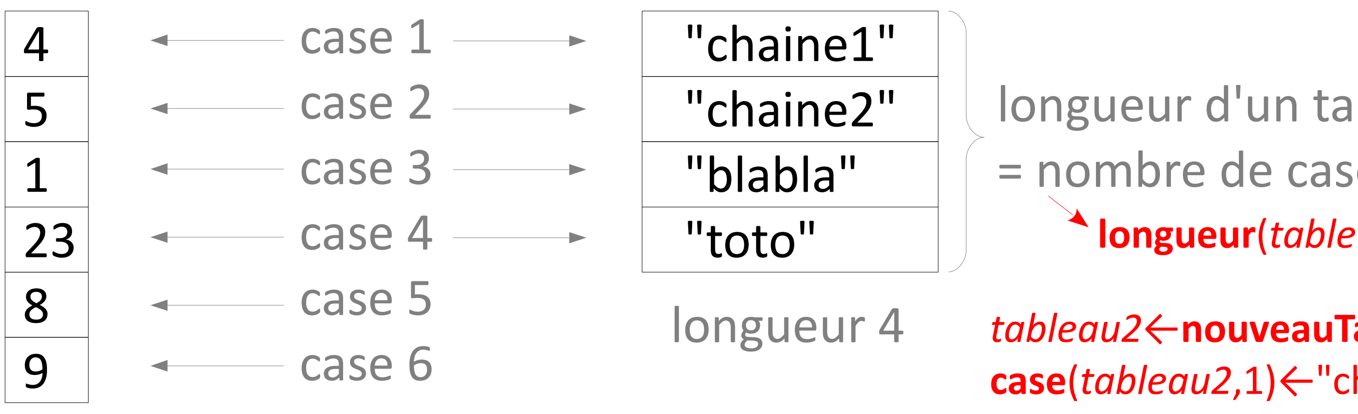
Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

en pseudo-code

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

Par exemple,

Un **tableau d'entiers** : Un **tableau de chaînes de caractères** :

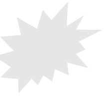


longueur(*tableau2*)

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

tableau2←**nouveauTableau**(4)
case(*tableau2*,1)←"chaine1"
case(*tableau2*,2)←"chaine2"
case(*tableau2*,3)←"blabla"
case(*tableau2*,4)←"toto"



Les tableaux en Javascript

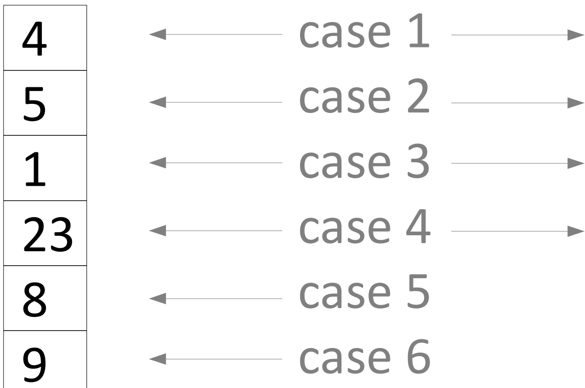
Les tableaux sont des variables qui contiennent **plusieurs variables**, stockées chacune dans une des cases du tableau.

en Javascript

Par exemple,

Un **tableau d'entiers** :

Un **tableau de chaînes de caractères** :



"chaine1"
"chaine2"
"blabla"
"toto"

longueur 4

longueur d'un tableau = nombre de cases

`tableau2.length`

longueur 6

```
var tableau1 = [];
tableau1[0] = 4;
tableau1[1] = 5;
tableau1[2] = 1;
tableau1[3] = 23;
tableau1[4] = 8;
tableau1[5] = 9;
```

```
var tableau2 = [];
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 Javascript.

Manipulation de tableaux

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

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

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

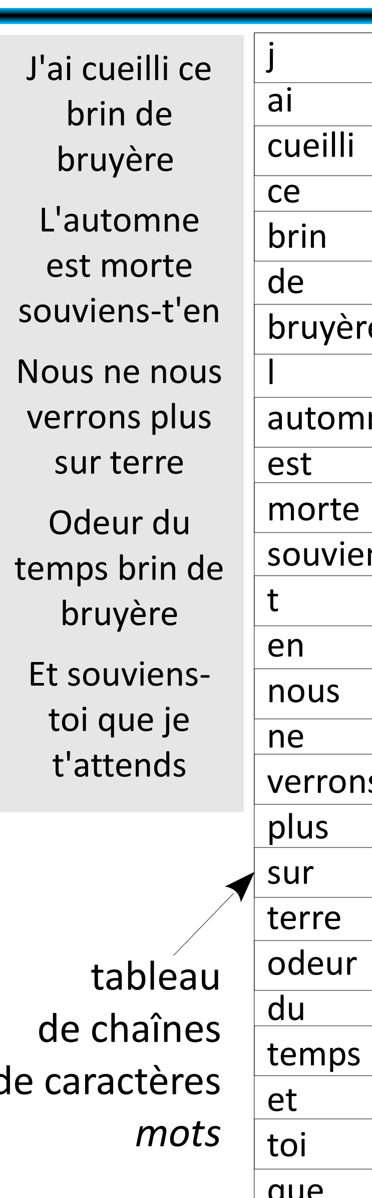
Variables : tableau d'entiers *tab*, entier *i*
i ← 1
Tant que *i* < **longueur**(*tab*)+1 faire :
    [des choses avec la *i*-ième case du tableau **case**(*tab*,*i*)...]
    i ← *i*+1
Fin Tant que

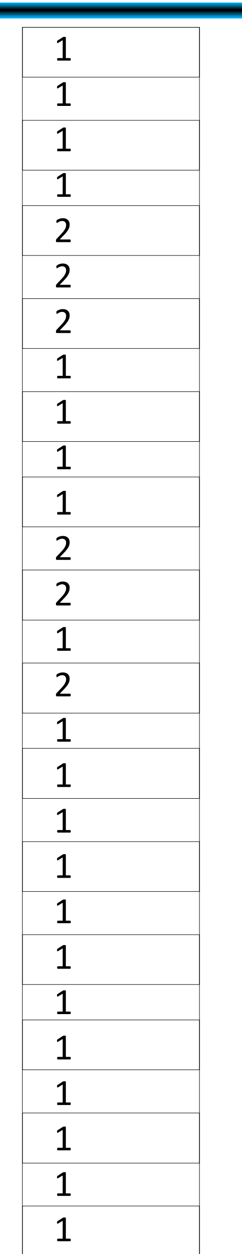
Affichage du contenu d'un tableau d'entiers

Algorithme **afficheTableau**
Variable d'entrée : tableau d'entiers *tableau1*
Variable :
Début
    i ← 1
    Tant que *i* < **longueur**(*tableau1*)+1 faire :
        affiche(**case**(*tableau1*,*i*))
        i ← *i*+1
    Fin Tant que
Fin

```
function afficheTableau(tableau1){
    //Afficher dans la console les cases du tableau tableau1
    var i;
    i = 0;
    while (i<tableau1.length){
        console.log(tableau1[i]);
        i = i+1;
    }
}
```

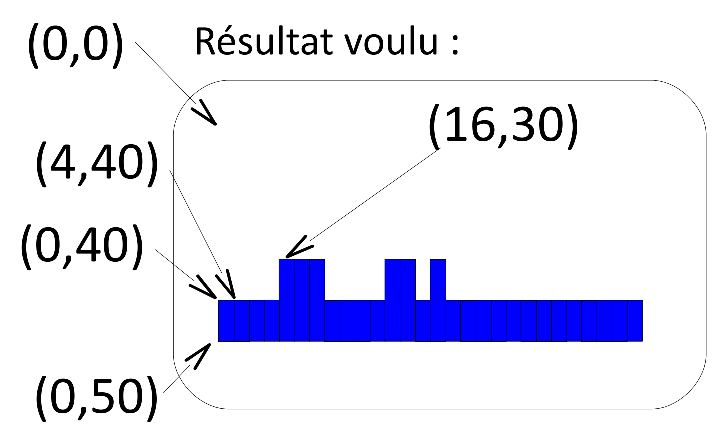
Graphique du nombre d'apparitions des mots dans un texte





nbApparitions
1
1
1
1
2
2
2
1
1
1
1
2
2
1
2
1
1
1
1
1
1
1
1
1
1
1
1
1

tableau d'entiers *nbApparitions*



Graphique du nombre d'apparitions des mots dans un texte

Algorithme **dessineHistogramme**

Variables d'entrée : chaîne de caractères *imageDeFond*, entiers *abscisseBG*, *ordonneeBG* et *hauteurMin*, tableau d'entiers *nbApparitions*, chaîne de caractères *couleur*

Variable : entier *i*

Début

i ← 1

Tant que   faire :

    i ← 1 + *i*

Fin Tant que

Fin

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*),
        "blue")
    compteur ← 1 + *compteur*
Fin Tant que
Fin

Algorithme **dessineHistogramme**
Entrée : tableau de chaînes de caractères *mots* et tableau d'entiers *nbApparitions*.
Variable : entier *compteur*
Début
Pour tout *compteur* de 1 à **longueur**(*mots*) faire :
    dessineRectanglePlein(*compteur**4-4,
        50-10***case**(*nbApparitions*,*compteur*),
        4,10***case**(*nbApparitions*,*compteur*),
        "blue")
Fin Pour
Fin

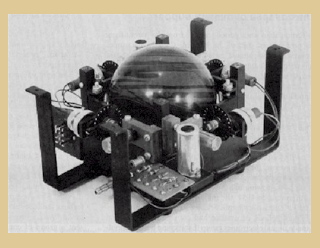
En Javascript :

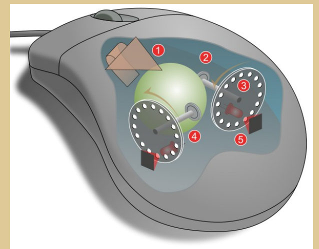
```
var compteur;
compteur=1;
while(compteur<mots.length+1) {
   …
   compteur++;   ⟷  compteur+=1
}                ⟷  compteur=compteur+1
```

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

```
for(var 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 pseudo-code et Javascript

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"

Sorties écran (ligne de commande)

en pseudo-code

chaîne de caractères

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

en Javascript / jQuery depuis une page web

Charger jQuery dans la page web :

```
<SCRIPT TYPE="text/javascript"
 SRC="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></SCRIPT>
```

```
<!-- Code HTML -->
<input type="text" id="js-reponse1">
<script type="text/javascript">
// Code Javascript qui utilise jQuery :
$(document).ready(function(){
   $("#js-reponse1").on("change",function(){
      var chaine = $("#js-reponse1").val();
   }
}
</script>
```

Stocke, quand on change le contenu du champ texte dont l'*id* est *js-reponse1*, ce contenu dans la chaîne de caractères *chaine*.
En TP : appel possible de l'algorithme ***reponseALaQuestion*** en utilisant `await reponseALaQuestion("Texte de la question")`

```
var chaine = "blabla";
$(document).ready(function(){
   $("#zoneDInteraction").append(chaine);
}
```

ajoute la chaîne de caractères *chaine* à la fin du bloc HTML dont l'*id* est *zoneDInteraction*, dès que le code de la page est chargé dans le navigateur.

```
console.log(chaine);
```

affiche la chaîne de caractères *chaine* dans la console du navigateur.

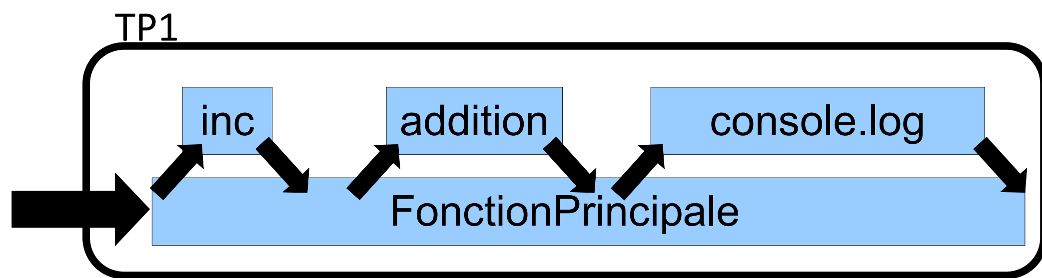
Les fonctions – Introduction

Algorithme pour **construire une maison** ?

→ faire appel à un **maître d'oeuvre**

Que fait le maître d'oeuvre ?
→ il réunit les informations sur la maison qui va être construite
→ il trouve des artisans
→ à chaque artisan, il donne des informations sur ce qu'il attend, et récupère le résultat
→ chaque artisan peut lui-même sous-traiter une partie de son travail à un autre artisan : il donne les informations sur ce qu'il attend, et récupère le résultat
→ la maison se construit

La construction de la maison : le programme TP1
Le maître d'oeuvre : la fonction main
L'artisan 1 : la fonction inc
L'artisan 2 : la fonction addition
L'artisan 3 : la fonction console.log




```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
function fonctionPrincipale(){
   var i = 5;
   var j = 34;
   console.log("i+1=" + inc(i) + ",i=" + i +
      ",j=" + j + ", somme : " + addition(i,j));
   }
function inc(i){
   i=i+1;
   return i;
}
function addition(i, j){
   return i+j;
}
$(document).ready(function(){
   fonctionPrincipale();
</script>
```

Les fonctions – Spécificités en Javascript

Les **fonctions** = les **algorithmes**

Zéro, un ou plusieurs paramètres en **entrée** (entre parenthèses après le nom de la fonction)...
Une **sortie** (ou aucune si pas de `return`)...
Écrites **les unes après les autres**, les fonctions peuvent s'appeler entre elles par un **appel de fonction**.

En Javascript :
Le code qui se situe entre les balises `<script type="text/javascript">` et `</script>` s'exécute s'il n'est pas contenu dans une fonction. S'il est à l'intérieur d'une fonction, il va falloir appeler cette fonction.

```
<html><head><title>Ma page web avec un code Javascript</title></head><body>
<script type="text/javascript" src="jquery.js"></script><!-- chargement de jQuery sur cette ligne -->
<script type="text/javascript">
function fonctionPrincipale(){
   var i = 5;
   var j = 34;
   console.log("i+1=" + inc(i) + ",i=" + i +
      ",j=" + j + ", somme : " + addition(i,j));
   }
function inc(i){
   i=i+1;
   return i;
}
function addition(i, j){
   return i+j;
}
$(document).ready(function(){ // sur cette ligne, détection par jQuery que la page est chargée.
   fonctionPrincipale(); // sur cette ligne, appel de fonctionPrincipale quand la page est chargée.
}
</script>
</body></html>
```

appels d'algorithme

variables d'entrée

pas la même variable *i* même si elles ont la même valeur !

ligne de déclaration d'un algorithme / d'une fonction

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** »

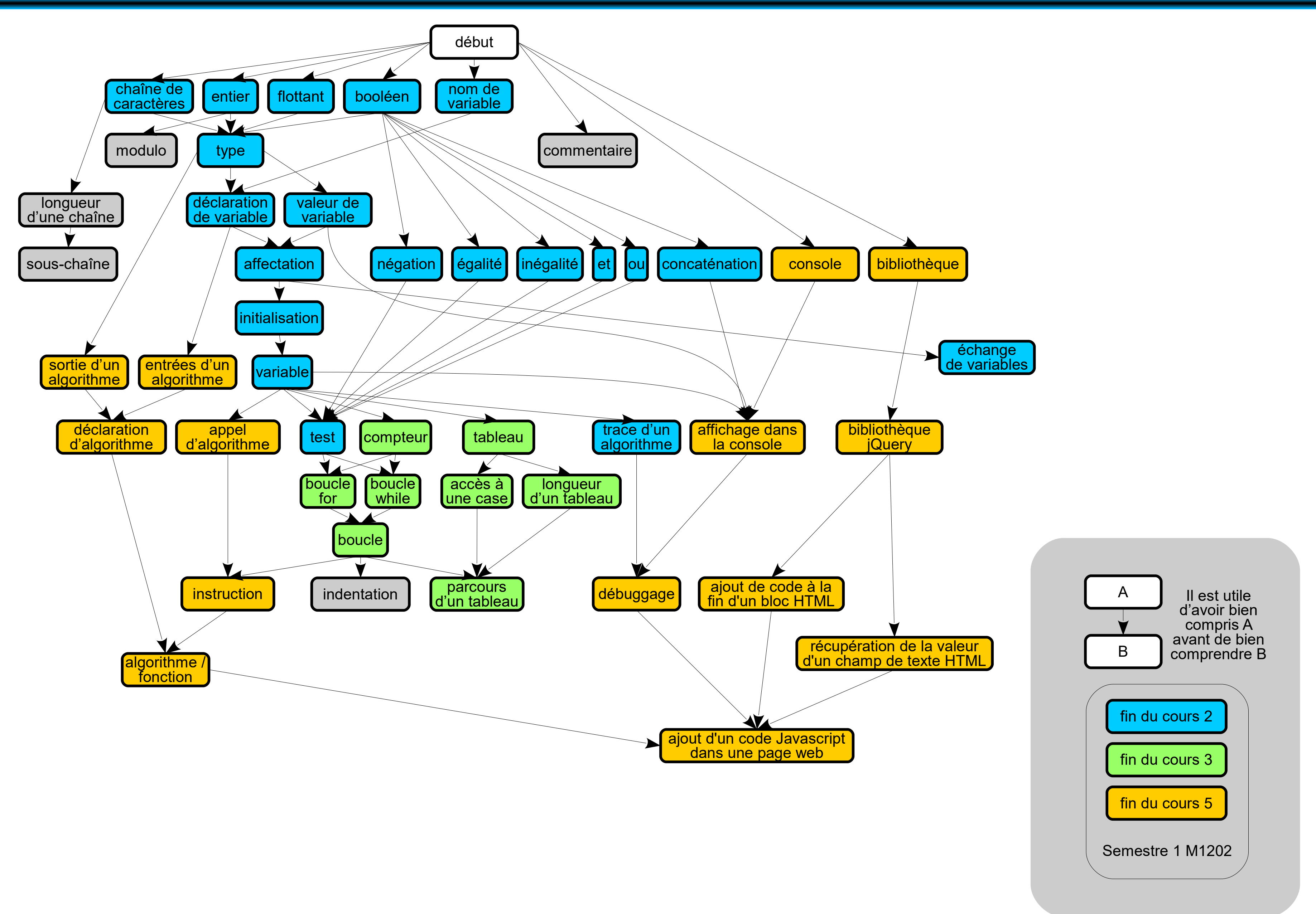
Les fonctions – Exemples

La “minute mathématique”

fonction	exemple	entrées possibles	sortie
cosinus	cosinus(1.047)=0.5	flottant	flottant
somme	somme(2,3)=5	2 entiers 2 flottants	entier flottant
opposé	opposé(4)=-4	entier flottant	entier flottant
inverse	inverse(10)=0.1	flottant	flottant
différence	différence(2,3)=-1	2 entiers 2 flottants	entier flottant
estPositif	estPositif(-5)=FAUX	entier	booléen
partieEntière	partieEntière(5.6)=5	flottant	entier
moyenne	moyenne(2,4,6)=4	3 flottants	flottant
min	min({6,2,4,3})=2	tableau d'entiers tableau de flottants	entier flottant

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

Concepts du cours M1202



Compétences du cours M1202

Être capable de :

C1) comprendre le fonctionnement d'un algorithme :

C1a) en identifiant les différents éléments de base de l'algorithme

C1b) en simulant son comportement à l'aide d'une trace

C1c) en interprétant une courte description en français de ses spécifications

C2) concevoir un algorithme pour résoudre un problème :

C2a) en analysant le problème :

- pour comprendre les besoins
- pour le découper éventuellement en sous-problèmes

C2b) en écrivant un algorithme correctement structuré :

- avec des entrées et des sorties correctement définies
- en utilisant des structures conditionnelles
- en utilisant des répétitions d'opérations (à l'aide de boucles)
- en faisant appel à d'autres algorithmes dont les spécifications sont connues, notamment pour interagir avec l'utilisateur
- en découpant le code de manière modulaire en divers composants indépendants et réutilisables

C3) comprendre le fonctionnement d'un code Javascript :

C3a) en identifiant les divers éléments de structure du code Javascript

C3b) en identifiant les éventuelles erreurs qu'il contient par un processus de débuggage

C4) écrire un programme Javascript :

C4a) en respectant la syntaxe Javascript

C4b) en choisissant ou en respectant des conventions de nommage appropriées

C4c) en testant le code obtenu (par compilation puis exécution)

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 !

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 (produit : variable ; entier1+produit : valeur)
- des **boucles**   Tant que … faire : … Fin Tant que
- 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 : … Fin Si
- des "**lectures**" d'entrées et "**renvois**" de sorties   renvoyer *i*+*j*

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

- la **déclarer** en définissant son **nom** et son **type** (ensemble de valeurs possibles)   Variables : entiers *a* et *b* (entiers : type)
- puis l'**initialiser** (lui donner une **valeur** par une affectation) avant de l'utiliser.

a ← 2
b ← *a*+2

Méthodo : Connaître les éléments de base d'un algorithme

en Javascript

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

function addition(nb1, nb2){...}
entrées

instructions finies par “;”

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

- des **affectations** dans des **variables** (mémoires) produit = entier1+produit;
  (produit : variable ; entier1+produit : 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 **nom** var a, b;
- puis l'**initialiser** (lui donner une **valeur** par une affectation) avant de l'utiliser.

```
a = 2;
b = a+2;
```