En cours

Développement de programmes

Rôle de l'IA

Bases fondamentales

Introduction à la Programmation : Premiers pas en programmation

Laboratoire d'Intelligence Artificielle
Faculté I&C

Support MOOC

Vidéo et Quiz de :

https://www.coursera.org/learn/initiation-programmation-java/home/week/1

- ☞ Semaine 1 du MOOC
- ☞ Notes de cours et BOOC de la même semaine

Pendant l'heure de cours

Résumé et approfondissements sur :

- le cycle de développement d'un programme
  - ☞ (section « Développement de programmes»)
- la notion de typage fort
  - ☞ On disserte un peu sur le calcul de moyennes
- quelques détails en plus sur `int` et `double`
- Operateurs et expressions :
  - les notations abrégées
    - ☞ infixé vs postfixé
- Fiches résumé : Variables et Opérateurs

Les instructions de l'ordinateur

Concrètement, quelles sont les instructions et les données « adaptées » à l'ordinateur ?

Ordinateur $\simeq$

microprocesseur
: détermine l'ensemble des instructions élémentaires que l'ordinateur est capable d'exécuter ;

mémoire centrale
: détermine l'espace dans lequel des données peuvent être stockées en cours de traitement

périphériques
: permettent l'échange ou la sauvegarde à long terme des données

Les instructions de l'ordinateur

Concrètement, quelles sont les instructions et les données « adaptées » à l'ordinateur ?

Ordinateur $\simeq$

microprocesseur

mémoire centrale

périphériques

☞ C'est donc **le microprocesseur** qui **détermine le jeu d'instructions** (et le type de données) à utiliser.

On les appelle Instructions Machine, Langage Machine,
$\simeq$ Assembleur

On peut programmer directement le microprocesseur en langage machine...
...mais c'est un peu fastidieux et de plus, chaque processeur utilise ses propres instructions

Exemple d'instructions-machine

Programme en Assembleur
1: LOAD 10 5
2: CMP 10 0
3: JUMP +3
4: DECR 10
5: JUMP -3
6: END

mettre 5 dans la mémoire 10
comparer le contenu de la mémoire 10 à 0
si tel est le cas sauter 3 instructions plus loin
décrémenter la mémoire 10 (de 1)
sauter 3 instructions en arrière
fin

Instructions machine :

Instructions	Code Machine	données	Code Machine
CMP	00000000	-3	10000011
DECR	00000001	0	00000000
END	00000010	2	00000010
JUMP	00000011	3	00000011
LOAD	00000100	5	00000101
		6	00000110
		10	00001010

Le programme ci-dessus correspond donc physiquement en machine à la séquence :

```
000001000000101000000101000000000000101000000000000001100000011
000000010000101000000111000001100000010
```

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

La notion de langage de programmation

Cependant, ces instructions-machine sont **trop élémentaires** pour pouvoir être efficacement utilisées (par les humains) pour l'écriture de programmes...

. . . il faut donc fournir au programmeur la possibilité d'**utiliser des instructions de plus haut niveau**, plus proches de notre manière de penser et de conceptualiser les problèmes . . .

Exemples de langage de programmation de haut niveau :

« *vieux* » BASIC	C

`1 N=5`
`2 IF (N>0) THEN PRINT N;`
`  N=N-1; GOTO 2`
`3 END`
	`main() {`
	`int n;`
	`for (n=5; n>0; --n)`
	`  printf("%d\n", n);`
	`}`

La notion de langage de programmation (2)

Comment rendre les instructions plus sophistiquées compréhensibles par l'ordinateur ?

- ☞ **traduire** les séquences d'instructions de haut niveau en instructions-machine directement exécutables par le microprocesseur

Selon ses caractéristiques, un tel traducteur est appelé **compilateur** ou **interpréteur**

L'ensemble des instructions de plus haut niveau qu'un compilateur ou un interpréteur est capable de traiter constitue un **langage de programmation**.

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

La notion de langage de programmation (3)

Un **langage de programmation** est donc un moyen formel permettant de décrire des *traitements* (i.e. des tâches à réaliser) sous la forme de *programmes* (i.e. de séquences d'instructions et de données de « haut niveau », compréhensibles par le programmeur) pour lesquels un **compilateur** ou un **interpréteur** est disponible pour permettre l'exécution effective par un ordinateur.

Exemples de langages de programmation : C, ADA, C++... et **Java**

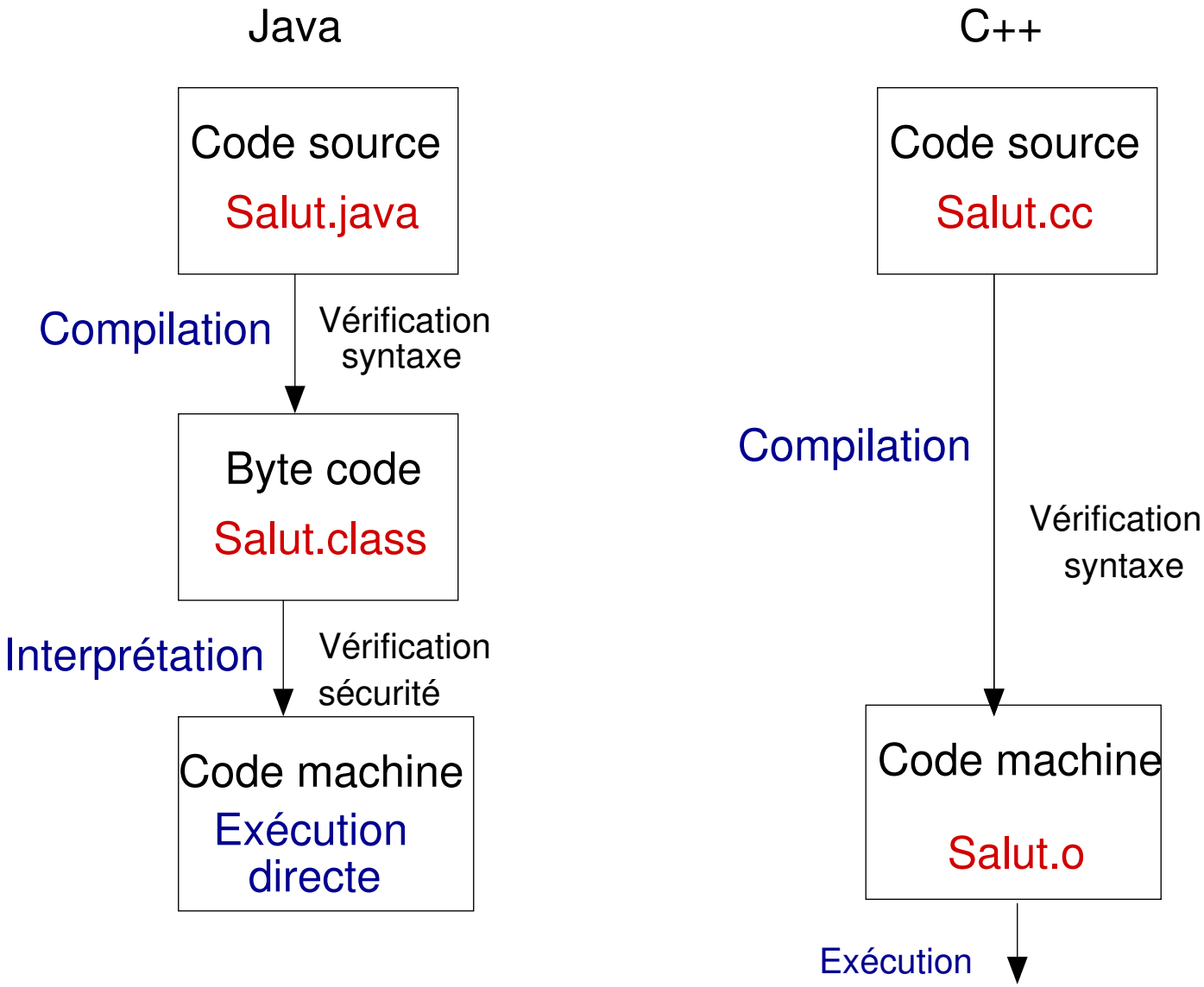
Interpréteur/Compilateur

- Le compilateur traduit un programme, écrit dans un langage de haut niveau, en un fichier binaire (exécutable) spécifique à une architecture matérielle (ARM, x86, amd64, ...). Chaque plateforme a son format de fichiers binaires (ELF, COFF, ...)

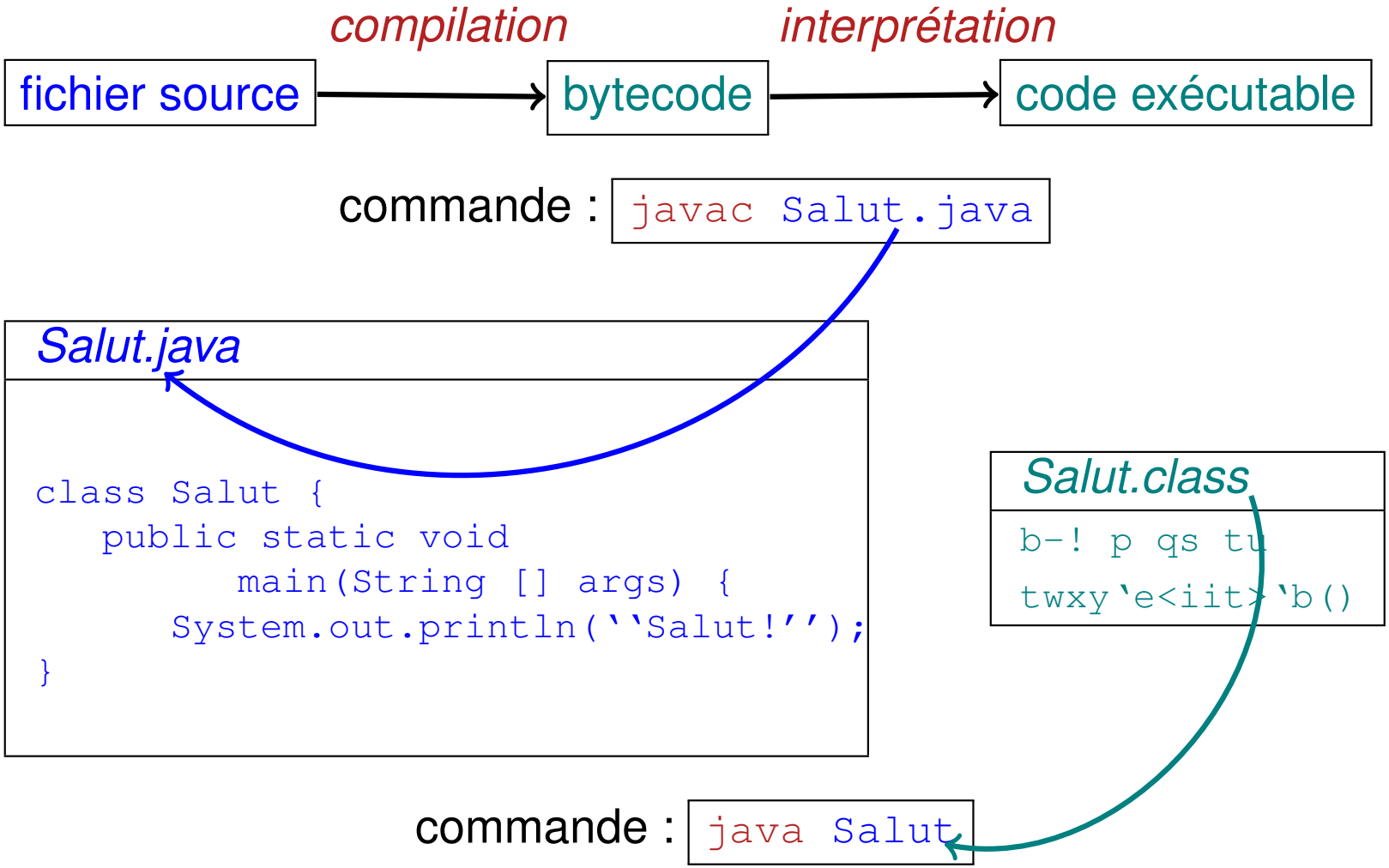
- L'interpréteur exécute un programme, écrit dans un langage de haut niveau, sans étape intermédiaire. Un programme interprété de manière naïve est plus lent qu'un programme compilé, mais indépendant de l'architecture.

☞ Java utilise les deux !

Java vs C++



Compilation d'un programme Java



L'interpréteur Java s'appelle la **machine virtuelle Java** (Java Virtual Machine (JVM))

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

Notion de bytecode [1]

Bytecodes Java ⇔ "langage intermédiaire" entre compilation et exécution :

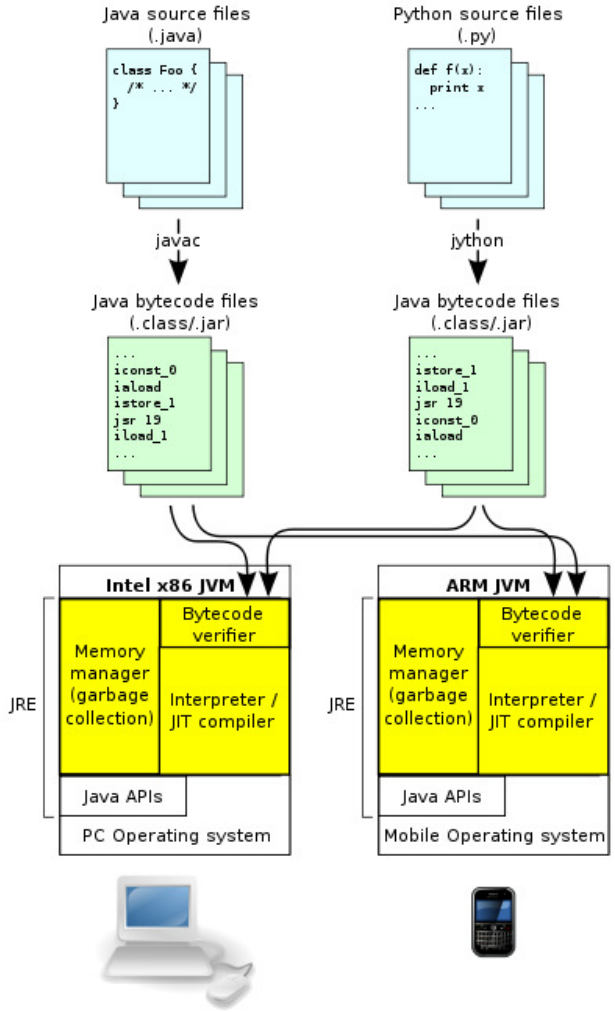
- Langage dans lequel est compilé un programme Java :
  - `javac Salut.java` produit `Salut.class` contenant des bytecodes.
- Pas vraiment "humainement lisible" et dans ce sens, il ressemble à du code assembleur. . . mais attention, les bytecodes sont **interprétés** par la machine virtuelle et donc pas executés directement par le processeur.
  - C'est à la machine virtuelle de traduire ces bytecodes en code machine : `java Salut`
    - La machine virtuelle va ouvrir le fichier `Salut.class` et interpréter les bytecodes contenus !

1. Le terme bytecode vient de byte car chaque instruction de la JVM est codée à l'aide d'un byte ⇒ à chaque instruction correspond un opcode de 8 bits. Exemple : l'addition de deux entiers est représenté par l'opcode `IADD` et correspond à `60` en hexadécimal.

Indépendance de la plateforme

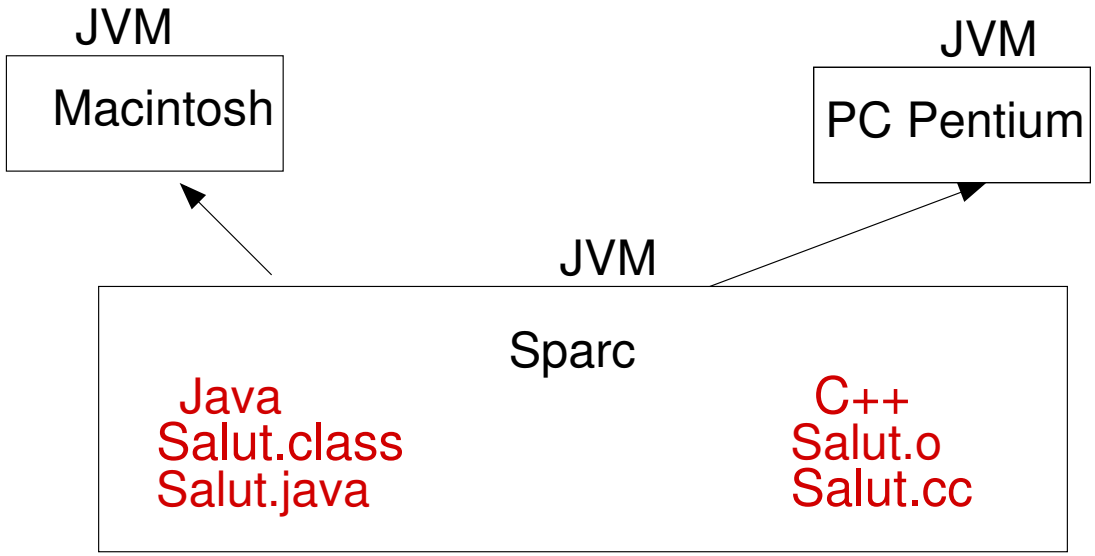
- Le bytecode rend le programme **indépendant de la plateforme**
  - Possibilité de démarrer le programme sur un autre processeur sans recompilation
  - Pour autant que la JVM soit installée
- L'idée de la "platform independence" (et de Java) :
  - "Write once, run anywhere"
- `Salut.java` $\Rightarrow$ autre processeur
  - Compilation + interprétation nécessaire
  - La compilation est lente
  - Le compilateur est un gros programme
- `Salut.class` $\Rightarrow$ autre processeur
  - Seulement l'interprétation est nécessaire
  - L'interprétation est rapide
  - La JVM est petite

Indépendance de la plateforme (2)



(emprunté à wikipédia)

Indépendance de la plateforme (2)



Java : La JVM pour processeurs Intel ou AMD peut interpréter du bytecode généré sur Sparc

C++ : le code machine généré sur Pentium est inutilisable sur Sparc (sauf éventuelle cross-compilation)

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

Cycle de développement

Programmer c'est :

① réfléchir au problème ; **concevoir l'algorithme**

② traduire cette réflexion en un *texte* exprimé dans un langage donné (écriture du programme source)

③ traduire ce texte sous un format interprétable par une JVM (**compilation**, en Java production de bytecode, javac)

④ exécution du programme (en Java **interprétation du bytecode, java**)

En pratique :

- erreurs de compilation (mal écrit)
- erreurs d'exécution (mal pensé)

$\Longrightarrow$ **correction(s)** ☞ d'où le(s) **cycle**(s) !

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

Cycle de développement (2)

En mode "terminal" :

- Les phases d'**écriture** et de **correction** se fait à l'aide d'un éditeur de texte (emacs, geany, gedit, notepad, . . .) ou d'une EDI (voir plus loin)
  Exemple : geany Salut.java

- La phase de **compilation** se fait à l'aide de la commande javac.
  Exemple : javac Salut.java
  - La compilation produit un fichier .class.
    Exemple : le fichier Salut.class est généré par le compilateur

- La phase d'**interprétation** se fait à l'aide de la commande java.
  Exemple : java Salut

J. Sam

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

Cycle de développement (EDI)

Avec un environnement de développement intégré (EDI), par exemple IntelliJ, le programmeur n'appelle pas **explicitement** les commandes javac ou java, mais elles sont appelées par l'EDI.

- La phase de **compilation** se fait à la volée par l'EDI (il y a donc bel et bien une compilation qui est faite **implicitement** !) lorsque des changements ont été effectués sur les fichiers sources.
- La phase d'**interprétation** se lance graphiquement à l'aide du bouton **Run**. L'EDI lance l'interprétation (c'est-à-dire la commande java !) lors du clic sur le bouton **Run**. Le résultat apparaît dans une console intégrée à l'EDI.

En cours
Développement de programmes
Langages de programmation
Cycle de développement (Java)
Rôle de l'IA
Bases fondamentales

Interpréteur/Compilateur (suite et fin)

- Des optimisations sont généralement appliquées au temps de la compilation (élimination de code inutile, mise à profit de jeux d'instructions spéciaux etc . . .)

- Cependant, la JVM (environnement d'exécution de Java) effectue de la **compilation et des optimisations pendant l'exécution !** Ces techniques, appelés JIT (Just-In-Time) permettent par exemple aux programmes Java de rivaliser avec des programmes C++ (uniquement compilé) en terme de performance.

Bien apprendre à programmer / Rôle de l'IA

[Une partie du matériel de ces 4 slides est partagé par plusieurs cours de programmation (J-C. Chappelier, J. Sam, C. Salzmann et S. Doeraene)]

Pour apprendre à programmer, il faut **pratiquer par soi-même** (y compris le debugging)

Se confronter aux difficultés, y réfléchir, reformuler, etc. sont nécessaires pour (bien) apprendre

- ☞ Lire une solution (IA ou autre), sans avoir au préalable **FAIT par soi-même**, est peu utile

« *On a l'impression de devenir bête. On ne réfléchit plus, car on peut demander directement à une IA.* »

Utilisation de GenAI et apprentissage

Certes, les outils à base d'IA générative (ChatGPT, Github-Copilot, Claude, etc.) révolutionnent désormais l'approche au codage et sont déjà considérés, à juste titre, comme des supports indispensables

Mais ils ne sont pas faits pour l'apprentissage si l'on ne maîtrise pas les bases !

Utiliser une calculatrice nous apprend-il les bases du calcul ?

Pouvez-nous comprendre pourquoi une calcul (sur calculette) est faux sans connaître les bases de l'arithmétique ?

Utilisation de GenAI et apprentissage

Derrière l'écriture d'un programme, il y a des enjeux de **modélisation** !

Si l'on ne comprend pas ce qui caractérise une bonne modélisation, on ne saura pas interroger une IA de façon adéquate

Si vous êtes débutant.e, utiliser des outils GenAI à mauvais escient peut sensiblement péjorer votre efficacité et vos méthodes d'apprentissage ainsi que vos capacités d'analyse

- ☞ à terme **ce ne sera pas un gain de temps**
- ☞ et notre but est de former des ingénieur(e)s, pas uniquement des codeuses/codeurs :)

Pourquoi les IA disent des bêtises (surtout aux débutant.es)

Étudiant(e)s :

- question (« *prompt* ») mal posée :
  mauvais ou manque de contexte, mauvaise compréhention de l'erreur
- manque de recul critique pour évaluer la pertinence de la réponse

  ☞ gardez un **esprit critique**

GenAI :

- mélange de différents langages de programmation
  (en raison de fortes similitudes mais subtiles différences)
- ne compile pas et n'exécute pas le code, ne fait que le **prédire** (prédit la suite la plus probable suivant ses données d'entraînement)
- génération d'erreurs par reformulation (p.ex. troncature de parties de code)

Conseils d'utilisation d'IA dans ce cours

Pour ce cours, du point de vue de l'enseignement, nous allons faire comme si les outils d'IA n'existaient pas

- ☞ **Le but n'est pas de vous apprendre à interroger ces outils, mais de vous enseigner ce qui constitue un bon programme, techniquement et méthodologiquement.**

De votre côté, l'usage de ces outils est toléré pour des tâches simples et comme support à la compréhension

- ☞ Il est indispensable de vous **re-approprier** le matériel ou les explications produites par ces outils en vous montrant capable de les comprendre, de les re-expliquer par vous même et d'y porter un regard critique.

Conseil : si vous débutez, utilisez les outils IA pour chercher des réponses à vos questions
mais **pas** pour produire de grandes quantités de code ni pour déboguer

☞ https://www.epfl.ch/education/teaching/index-html/ai-teaching/ai-in-student-learning/

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Le langage Java

Le langage Java est un langage orienté-objet fortement typé.

☞ un des langages objets les plus utilisés !

Parmi les avantages de Java, on peut citer :

- langage populaire (bonne documentation, communauté active),
- une grande bibliothèque de fonctionnalités prédéfinies,
- un typage fort, ce qui permet au compilateur d'effectuer des vérifications sur la correction du programme,
- langage indépendant de la plateforme,
- langage adapté aux systèmes de communications (programmation réseau).

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Structure générale d'un programme Java

Le programme Java le plus simple... et le plus inutile (car il ne fait rien !) est le suivant :

```
class Rien {
    public static void main(String[] args) {
    }
}
```

Ce programme déclare la **classe** `Rien` contenant la **méthode** `main`.
Il déclare que :

1. cette méthode est vide : son **corps** (délimité par `{` et `}`) est **vide**,
2. cette méthode ne retourne rien : c'est le `void` devant le nom de la méthode.

☞ La méthode `main` doit nécessairement être présente dans tout programme Java. C'est la **première méthode exécutée au lancement d'un programme**.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Classes, mots-clés et identificateurs

- `class`
  - Marque le début d'une classe
  - Mot-clé = mot réservé de la syntaxe

- Classe = "brique de base" (description d'un type de données)
  - Un programme possède souvent plusieurs classes
  - Pour commencer : 1 classe/programme

- `Rien`
  - Nom de la classe (du programme)
  - Identificateur = nom choisi (presque) librement par le programmeur

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Fichiers, blocs et méthodes

- Convention (obligatoire !) :
  - Réutilisation de l'identificateur qui décrit le nom de la classe
  - Nom du fichier qui stocke le programme = Nom de la classe + `.java`
  - `Rien.java` (qu'on édite dans IntelliJ, par exemple)
- Première et dernière accolade
  - Bloc `{ }`
  - Marque les limites de la classe
- Contenu d'une classe :
  - Diverses instructions
  - Pour commencer : 1 méthode/classe
- Méthode :
  - En-tête spécifiant l'identité de la méthode
  - Bloc d'instructions à exécuter

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

La méthode main

- En-tête de la méthode `main` :

```
public static void main(String[] args)
```

- Pour commencer :
  - En-tête standard à apprendre par coeur
  - Chaque programme possède une méthode `main`
- Lors du démarrage du programme :
  - La méthode `main` est recherchée
  - Son bloc d'instructions est exécuté
  - S'il n'y a pas de méthode `main`, le programme ne démarre pas

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Premier exemple de programme (qui fasse qqchose)

On veut réaliser un programme qui résout (dans $\mathbb{R}$) une équation du second degré de type :

$$x^2 + b\,x + c = 0$$

Pour b et c fixés, les solutions réelles d'une telle équation sont :

$$\begin{cases} \left\{ \frac{-b+\sqrt{\Delta}}{2}, \frac{-b-\sqrt{\Delta}}{2} \right\} & \text{si } \Delta > 0 \\ \left\{ \frac{-b}{2} \right\} & \text{si } \Delta = 0 \\ \emptyset & \text{sinon} \end{cases}$$

avec $\Delta = b^2 - 4\,c$

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Premier exemple de programme :

Formalisation des traitements

☞ **Algorithme**

facile dans un cas aussi simple (déjà bien formalisé au départ)
mais peut devenir (très) complexe

saisir les données b et c
$\Delta \leftarrow b^2 - 4\,c$
Si $\Delta < 0$
  afficher "pas de solution"
Sinon
  Si $\Delta = 0$
    $x \leftarrow -\frac{b}{2}$
    afficher x
  Sinon
    $x \leftarrow \frac{-b-\sqrt{\Delta}}{2}$, $y \leftarrow \frac{-b+\sqrt{\Delta}}{2}$
    afficher x et y
  fin du si
fin du si

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Premier exemple de programme en Java

```
import java.util.Scanner;

class Degre2 {
  public static void main (String[] args) {

    Scanner keyb = new Scanner(System.in);
    double b=0.0;
    double c=0.0;
    double delta=0.0;

    b = keyb.nextDouble();
    c = keyb.nextDouble();
    delta = b*b - 4.0*c;
      if (delta < 0.0) {
      System.out.println( "Pas de solutions réelles");
       } else if (delta == 0.0) {
      System.out.println ("Une solution unique : " + -b/2.0);
    } else {
      System.out.println("2 solutions: " + (-b-Math.sqrt(delta))/2.0
           +  " et " + (-b+Math.sqrt(delta))/2.0);
    }
  }
}
```

données
traitements
structures de contrôle

J. Sam

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Qu'allons nous voir en programmation ?

Programmer c'est décomposer une **tâche** à automatiser en une séquence d'instructions (**traitements**) et des **données**

Algorithme	S.D.A.
Traitements	Données
Expressions & Opérateurs	Variables
Structures de contrôle	Types de base
Modularisation (méthodes)	Portée
Passage de paramètres par valeur	Chaînes de caractères
Récursivité	Tableaux statiques et dynamiques
Complexité	Structures de données abstraites

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Données et traitements

Comme dans tout langage de programmation évolué, on a en Java la possibilité de définir des **traitements** mis en œuvre sur des **données**. Pour être utilisée dans un programme Java, une **donnée** doit être stockée quelque part :

- ☞ dans une **variable** :
  objet informatique manipulable par le programme.

Les **traitements** sont associés dans le programme à la notion d'**instructions** et d'**expressions**.

- Une instruction indique à la machine comment manipuler les données
- En Java, elle se terminera toujours par un *point-virgule* : `;`

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

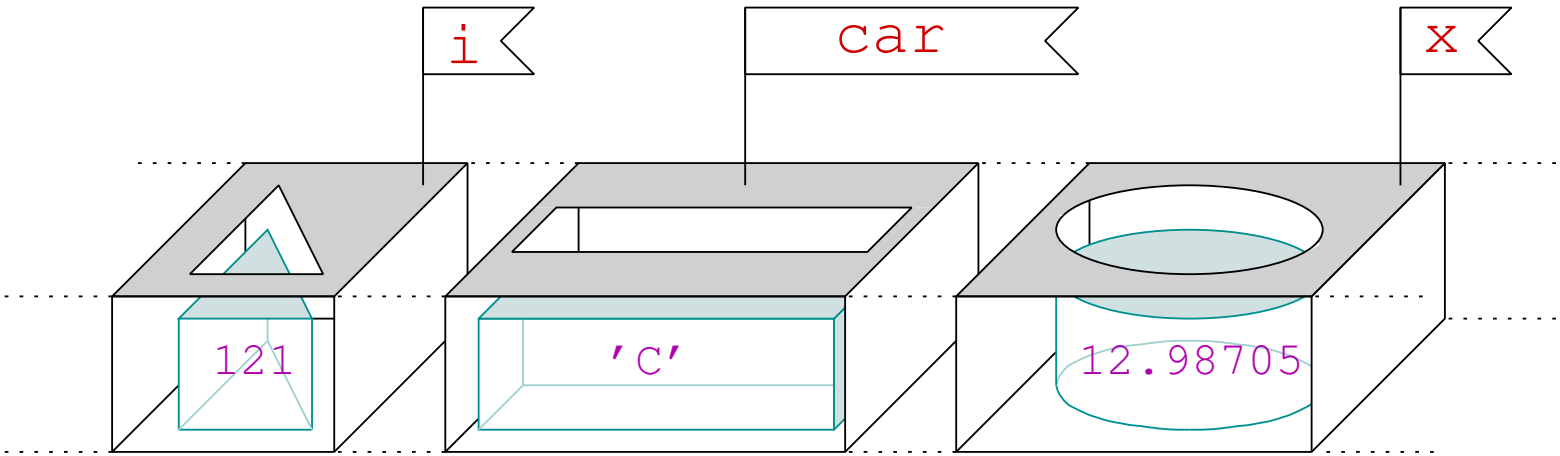
Variables : Définition

Une variable est décrite à l'aide de trois caractéristiques :

- Son **identificateur** qui est le nom par lequel la donnée est désignée.
  Exemples : `b`, `delta`, `myWindow`, ...
- Son **type** qui définit de quel genre est la donnée associée à la variable, en particulier, quels traitements elle peut (et ne peut pas) subir.
  Cette caractéristique sera généralisée plus tard en la notion de classe.
  Exemples : `double`, `int`, `String`, ...
- Sa **valeur littérale** qui permet de définir sa *valeur*. Par exemple, si la donnée est un nombre, sa représentation pourra être (selon les conventions de représentation) : 123, -18, 3.1415, 2e-13, ...

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Variable = zone mémoire




```
int i = 121;

char car = 'C';

double x = 12.98705;
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Déclaration et initialisation de variables

En Java, une variable doit être **déclarée** avant d'être utilisée.

La syntaxe de la déclaration d'une variable est :

```
type identificateur ;
```

où *type* est l'identificateur de **type** de la variable déclarée et *identificateur* est une chaîne de caractères permettant de **référer** la variable créée et donc de l'utiliser dans un programme.

Exemples :

```
int val;
double delta;
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Déclaration (2)

- Les principaux **types élémentaires** définis en Java sont :

`int`	:	les nombres entiers
`double`	:	les nombres réels
`char`	:	les caractères
`boolean`	:	les valeurs logiques "*vrai*" (`true`) et "*faux*" (`false`)

  Note : nous verrons plus tard d'autres types : les types composés et la généralisation des types par la notion de **classe**

- Un **identificateur** de variable peut être n'importe quelle séquence composée de lettres, de chiffres ou du caractère '_' et commençant par une lettre ou par '_'.

Remarque : un identificateur ne peut pas correspondre à un mot réservé du langage (if, else, while, ..., voir plus tard au long du cours).

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Conventions de nommage en Java

- Nom de classe :
  - Commencez chaque mot par une majuscule

```
HelloWorld
MonPremierProgramme
```

- Nom de variable ou méthode :
  - Commencez chaque mot par une majuscule, sauf le premier

```
maPremiereVariable
maPremiereMethode
nb1
main
```

- Usage non recommandé :

```
mapremierevariable
ma_premiere_variable
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Le type `int`

- Une variable de type **`int`** peut contenir un nombre entier :
  - max 4 octets (bytes) = 32 bits
    `[-2147483648, +2147483647]`
    $[-2^{31}, 2^{31}-1]$
- D'autres types pour les nombres entiers :
  - **`byte`** (1 octet)
  - **`short`** (2 octets)
  - **`long`** (8 octets)
- Plusieurs variables de même type peuvent être déclarées ensemble :

```
int maPremiereVariable, x;
int y;
short nb1, nb2;
```

. . . mais il est conseillé de ne déclarer qu'une variable par ligne.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Promotion entière

Le code suivant ne compile pas :

```
short i = 1, j = 2;
i = i + j; // erreur
```

- Les opérandes et l'expression sont convertis en **`int`**
- But : éviter un débordement de capacité sans que le programmeur ne soit conscient du risque

Conversion explicite nécessaire :

```
short i = 1, j = 2;
i = (short)(i + j);
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Le type `double`

- Une variable de type **`double`** peut contenir un nombre décimal :
  - max 8 octets = 64 bits
    $[\ \pm 1.7976931348623157 \cdot 10^{308}\ ]$
- Écriture d'une valeur double :
  - Point au lieu de virgule,
  - `e` au lieu de 10 en notation scientifique
- Exemples :

```
double x = 4.712;
double y = 53.17e15;
```

- Autre type pour les nombres décimaux :
  - **`float`** (4 octets)

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Plus de détails sur les types de base

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Initialisation

En même temps qu'elle est déclarée, une variable peut être **initialisée**, c'est-à-dire qu'on lui donne une première valeur avant même toute utilisation.

Note : Java ne vous laissera pas utiliser une variable non initialisée. Initialisez toujours vos variables... vos programmes gagneront en clarté et vous perdrez moins de temps à compiler

La syntaxe de la **déclaration/initialisation** d'une variable est :

```
type identificateur = valeur_d'initialisation;
```

où *valeur_d'initialisation* est n'importe quelle **constante** (i.e., *valeur littérale*) ou **expression** du type indiqué.

Exemples :

```
int val = 2;
double pi = 3.1415;
char c = 'a';
int j = 2*val+5;
```

Les expressions vont être définies et détaillées plus tard

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Valeurs Littérales

- Valeurs littérales de type entier : `1, 12, ...`
- Valeurs littérales de type réel : `1.23, ...`

  Remarque :

  `12.3e4` correspond à $12.3 \cdot 10^4$ (soit `123000`)
  `12.3e-4` correspond à $12.3 \cdot 10^{-4}$ (soit `0.00123`)
- Valeurs littérales de type caractère : `'a', '!', ...`

  Remarque :

  le caractère apostrophe (`'`) se représente par `'\''`
  le caractère `\` se représente par `\\`
- Valeurs littérales de type booléen : `true, false`

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Données modifiables/non modifiables

Par défaut les variables en Java sont modifiables.
Mais on peut vouloir imposer que certaines variables ne le soient pas : définir des **constantes**

La nature **modifiable** ou **non modifiable** d'une donnée peut être définie lors de la déclaration par l'indication du mot réservé **`final`**.

Elle ne pourra donc plus être modifiée par le programme (toute tentative de modification produira un message d'erreur lors de la compilation).
Exemple :

```
final double PI = 3.141592653;
```

Convention de nommage pour les constantes : on utilise des lettres majuscules, au besoin séparées par les "_".

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Affectation

L'opération d'affectation affecte (!) une valeur à une variable.

En Java, la syntaxe d'une affectation est :

```
identificateur = valeur ;
```

où `valeur` est une constante ou une **expression** (voir plus loin) du même type que la variable référencée par `identificateur`.

Exemple : `i = 3 ;`

On dit aussi que l'on assigne une valeur à une variable.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Bien comprendre l'affectation

Exemple :

```
x = x + 1;
```

C'est une instruction qui décrit **un processus dynamique** :

① lire le contenu de la case mémoire correspondant à `x`
② ajouter 1 à ce contenu (sans modifier `x` ici) ;
③ mettre le résultat dans la case mémoire correspondant à `x`

Notez bien qu'il n'y a pas de cycle ! L'expression n'est évaluée qu'une seule fois.

Pour résumer :
si la valeur de la variable `x` est 4 avant l'expression `x = x + 1;`
sa valeur après cette instruction est 5.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Réutilisation d'une variable

Que se passe-t-il si l'on réutilise une variable ?

- L'affectation remplace l'ancienne valeur
- Comme vu précédemment, la partie droite est évaluée avant l'affectation

```
int x;
int y;
x = 1;          // x vaut 1
y = 2;          // y vaut 2
x = 3;          // x vaut 3
y = x;          // y vaut 3
x = x + 1;      // x vaut 4
x = x - y;      // x vaut 1
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Typage fort

- Pourquoi des types différents ?
  - Vérification de la cohérence du programme
  - Utilisation efficace de la mémoire
- Java est un langage à typage fort :
  - Chaque variable a un type
  - La valeur doit être du même type que la variable
  - On ne peut pas mélanger des variables de types différents
- Les règles pour `int` et `double` sont plus souples :
  - Transtypage automatique : `int` $\Rightarrow$ `double`
  - Transtypage explicite (explicit cast) : `double` $\Rightarrow$ `int`

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Typage fort (2)

- Un **`int`** à droite peut devenir un **`double`** à gauche mais pas vice versa :

```
double d;
int i;
d = 1.7;   // Oui
d = 1;     // Oui, devient 1.0
i = 1;     // Oui
i = 1.7;   // Non !!!
```

- On peut forcer une conversion **`double`** $\Rightarrow$ **`int`** :

```
int i;
i = (int) 1.7; //Oui, troncature, i vaut 1
i = (int) d;   //Oui, idem
```

Attention : Valeur tronquée ne veut pas dire valeur arrondie !

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Typage fort (3)

- Si des **`int`** et **`double`** interviennent dans un calcul, les **`int`** deviennent des **`double`** :

```
int i;
double d;
i = 5 / 2;              // Oui, 2
d = 5.0 / 2.0;          // Oui, 2.5
d = 5.0 / 2;            // Oui, 2.5
d = 5 / 2.0;            // Oui, 2.5
d = 5 / 2;              // Oui, 2.0
i = 5 / 2.0;            // Non
i = (int) (5 / 2.0); // Oui, 2
```

Variables

En Java, une donnée est stockée dans une variable caractérisée par :

- son **type** et son **identificateur** (définis lors de la **déclaration**),
- sa **valeur**, définie la première fois lors de l'**initialisation** puis éventuellement modifiée par la suite.

Rappels de syntaxe :

```
type id ;
type id = valeur;

id = expression ;
```

Types élémentaires :

```
int
double
char
boolean
```

Exemples :

```
int val = 2 ;
final double PI = 3.141592653;
i = j + 3;
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Opérateurs et expressions

La syntaxe des expressions en Java est la même qu'en Python.
Seuls les symboles utilisés pour identifier les différents opérateurs sont différents :
Java fournit des **opérateurs** permettant d'écrire des **expressions**.

Les opérateurs sont associés au type des données (arguments) sur lesquels ils peuvent opérer.
Ainsi, les opérateurs *arithmétiques* (`+`, `-`, `*`, `/`, ...) sont définis pour des arguments de types numériques (entiers et réels), les opérateurs *logiques* (`&&`, `||`, `!`, ...) pour des arguments de type booléen, etc...

Les **expressions** sont des séquences (*bien formées* au sens de la syntaxe) combinant des opérateurs et des arguments (variables ou valeurs).

Exemple d'expression numérique : `(2*(13-i)/(1+4))`

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Opérateurs et expressions (2)

Les opérateurs arithmétiques sont :

Opérateur	Opération	Exemple
`+`	Addition	`z = x + y;`
`-`	Soustraction	`z = x - y;`
`*`	Multiplication	`z = x * y;`
`/`	Division	`z = x / y;`
`%`	Modulo (reste div)	`z = x % y;`
`-`	Négation	`y = -x;`

L'évaluation d'une expression conduit (naturellement) à sa valeur.
Exemple : l'évaluation de l'expression `(2*(13-3)/(1+4))` correspond à la valeur `4`
Par le biais de la notion d'évaluation, une expression peut donc également être considérée comme une représentation alternative de l'objet correspondant à son évaluation. Ainsi l'expression `(2*(13-3)/(1+4))` est une représentation alternative de l'entier 4.

Note : nous avons déjà précédement rencontré l'opérateur `=`, l'opérateur d'affection, qui est universel : s'applique à tout type

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Notations abrégées

- Opération très courante :
  - Additionner/soustraire 1 d'une variable

Opérateur	Opération	Exemple
`++`	Incrémentation	`++x` ou `x++;` `x = x + 1;`
`--`	Décrémentation	`--x` ou `x--;` `x = x - 1;`

```
int x = 1, y = 1;
x = x + 1;
y++;
y--;
```

- Evitez l'utilisation de `++` et `--` dans des formules :

```
z = x++ * y--;      // Mauvais style !
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Opérateurs et expressions (3)

Autres exemples d'expressions, utilisées ici pour des affectations :

```
z = (x + 3) % y;
z = (3 * x + y) / 10;
```

Java fournit un certain nombre de **notations abrégées** pour des affectations particulières.

`x = x + y` peut aussi s'écrire `x += y`

(idem pour `-`, `*`, `/` et `%`)

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

ATTENTION PIÈGE !

Remarque sur l'opérateur de division en Java :

- si `a` et `b` sont des *entiers*, `a/b` est le quotient de la division **entière** de `a` par `b`
  Exemple : `5/2 = 2`
  (et `a%b` est le reste de la division entière de `a` par `b`
  Exemple : `5%2 = 1`)
- si `a` ou `b` sont des *réels*, `a/b` est le résultat de la division réelle de `a` par `b`
  Exemple : `5.0/2.0 = 2.5`

Note : dans une expression constante, on distingue un réel d'un entier en lui ajoutant `.` à la fin. En général pour la lisibilité on préfère ajouter `.0` :

`5.0` (réel) $\longleftrightarrow$ `5` (entier)

C'est un point.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Priorité entre Opérateurs

Il est largement préférable de parenthéser ses expressions (ne serait-ce que pour la lisibilité !). Par exemple écrire `(a * b) % c` plutôt que `a * b % c` En l'absence de parenthéses, l'évaluation sera dans l'ordre suivant des opérateurs :

`* / % + -`

Tous ces opérateurs sont associatifs à gauche :

`a + b + c` = `(a + b) + c`

En cas d'ambiguité entre opérateurs de même ordre de priorité, c'est la règle d'associativité qui s'applique

Exemples :
```
a * b % c = (a * b) % c
a % b * c = (a % b) * c
a + b * c % d = a + ((b * c) % d)
```

En résumé :

1. Parenthèses
2. - (négation)
3. `*` , `/`
4. `+` , - (soustraction)

Opérateurs

Operateurs arithmétiques

`*`	multiplication	
`/`	division	
`%`	modulo	
`+`	addition	
`-`	soustraction	
`++`	incrémentation	*(1 opérande)*
`--`	décrémentation	*(1 opérande)*

Operateurs de comparaison

`==`	teste l'égalité logique
`!=`	non égalité
`<`	inférieur
`>`	supérieur
`<=`	inférieur ou égal
`>=`	supérieur ou égal

Operateurs logiques

`&&`	"et" logique	
`||`	ou	
`^`	ou exclusif	
`!`	négation	*(1 opérande)*

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) : `!` `++` `-`, `*` `/` `%`, `+` `-`, `<` `<=` `>` `>=`, `==` `!=`, `^` `&&`, `||`

Notation abrégée : `x = x <op> y`, où `<op>` est un opérateur arithmétique, peut aussi s'écrire : `x <op>= y` (exemple : `x += y`)

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Entrées-Sorties

Un minimum d'interactivité...

Affichages :

```
System.out.print(...)
```

ou

```
System.out.println(...)
```

☞ définis dans Java

Lecture (d'un entier) :

```
import java.util.Scanner;
Scanner keyb = new Scanner(System.in);
int i = keyb.nextInt();
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

classe Scanner

- `import java.util.Scanner;`
  - ☞ on importe la classe `Scanner` pour la rendre visible
- `Scanner keyb = new Scanner(System.in);`
  - ☞ on crée un objet `Scanner` et on l'affecte à la variable `keyb`
- `int i = keyb.nextInt();`
  - ☞ On lit un entier et on le stocke dans la variable `i`
- La méthode `nextInt()` :
  1. Arrête le programme momentanément
  2. L'utilisateur peut entrer un nombre entier
  3. Affectation activée par la touche `r`eturn
- Utilisation courante :
  1. Afficher un texte avec `System.out.print()`
  2. Lire une valeur avec `nextInt()`
     - Notez la paire de parenthèses, nécessaire !
  3. Affecter la valeur lue à une variable

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

classe Scanner (2)

- Idem pour `nextDouble()`, qui lit un `double`
  - `nextFloat()`, `nextLong()`, ...

- `next()` retourne une `String` (s'arrête au premier délimiteur)
  - délimiteurs : espace / retour à la ligne / tabulation / ...

  $\Rightarrow$ Une `String` est une chaîne de caractères. Un caractère `char` est délimité par des apostrophes, comme 'i', par exemple. Une `String` est delimitée par des guillemets, comme "test", par exemple.
- `nextLine()` retourne le reste de la ligne courante (`String`) jusqu'au terminateur de ligne (`'\n'`).

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

classe Scanner (3)

Attention avec `nextLine()` !

```
int i = keyb.nextInt();
String s1 = keyb.nextLine();
String s2 = keyb.nextLine();
```

input 1
25 francs
23 francs
30 francs
...

input 2
14
euros
43
...

input 1 : `i` ←25, `s1` ←"francs", `s2` ←"23 francs"
input 2 : `i` ←14, `s1` ←"", `s2` ←"euros"

- Pourquoi s1 est vide ("") ? Parce que le `nextLine()` lit la `String` **vide** qui suit l'entier jusqu'au terminateur...

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

System.out.println

- Lors de l'exécution d'une méthode :
  1. Recherche de la définition de la méthode
  2. Mise à disposition des arguments
  3. Exécution du bloc d'instructions
  4. Retour au point initial
  5. Exécution de la prochaine instruction

- La méthode `System.out.println(...)` :
  1. Affichage de l'argument à l'écran
  2. Affichage d'un saut de ligne
  3. Si pas d'argument, seulement saut de ligne

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Exemples...

- Mettre une variable dans l'argument de la méthode à l'aide de l'opérateur de concaténation `+` :

```
System.out.println
      ("La moyenne vaut : " + laMoyenne);
```

- Plusieurs `+` sont possibles :

```
System.out.println
      ("x vaut " + x + " et y vaut " + y);
```

- On peut mettre le calcul directement dans l'instruction d'affichage :

```
System.out.println
      ("Le produit est : " + x * y);
```

- Utiliser des parenthèses si le calcul est une addition :

```
System.out.println
      ("La somme est : " + (x + y));
```

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Commentaires

- Code complexe = commentaires indispensables
- Deux types de commentaires :
  1. `//` le reste de la ligne est ignoré
  2. `/* .. */` le code entouré est ignoré

```
/* Ce programme calcule les notes du semestre
   La moyenne est 4.0 et la note maximale 6.0
*/
```

```
// nextInt() permet de lire un entier
```

- Commenter est un devoir du programmeur.

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Premières bases (résumé)

- **Traitements** et **données** sont les deux facettes complémentaires de la programmation .
- Avant de programmer il faut réfléchir à la conception du programme (**algorithmes**).
- Mes premières bases de programmation :
  - les variables,
  - expressions et opérateurs arithmétiques,
  - entrée-sorties de base.

☞ Je peux commencer à écrire mon premier programme Java

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

La suite

- Exercices :
  - Nouveaux tutoriels IntelliJ
  - Ecriture de vos premiers programmes
- Le prochain cours :
  - Expressions logiques
  - Structures de contrôle en Java

En cours
Développement de programmes
Rôle de l'IA
Bases fondamentales
Structure d'un programme (Java)
Variables
Opérateurs/Expressions
Entrées/sorties

Pour préparer le prochain cours

- Vidéos et quiz du MOOC semaine 2 :
  - Branchements conditionnels [13 :39]
  - Conditions [12 :13]
  - Erreurs de débutant, Le type boolean [14 :08]
- Vidéos et quiz du MOOC semaine 3 :
  - Itérations : introduction [13 :29]
  - Boucles [21 :37]
  - Blocs d'instructions [05 :59]
  - les autres vidéos sur les itérations (approfondissement et quiz peuvent être vues plus tard)
- Le prochain cours :
  - de 14h15 à 15h (résumé et quelques approfondissements)