Un algorithme est un ensemble ordonné d’instructions et opérations finies dans le but de produire un résultat. Cela signifie qu’il s’agit d’une chaîne d’instructions précises qui doivent être suivies dans l’ordre. Son objectif est de résoudre un problème, c'est-à-dire il a un objectif délimité.

Lorsqu’un développeur crée un programme, il crée essentiellement un ensemble d’algorithmes. Un programme informatique est un ensemble de commandes données à la machine, écrites dans un langage spécifique, pour effectuer une série d’opérations déterminées afin d’obtenir un résultat. La programmation permet de résoudre un problème de manière automatisée grâce à l’utilisation des algorithmes. Les instructions utilisées dans le programme représentent le code source.

Langage

Pour que le programme puisse être exécuté par la machine, il faut utiliser un langage que celle-ci peut comprendre. Le langage binaire est un langage à deux états représentés par des 0 et des 1 que l’ordinateur peut comprendre.

Lorsque le développeur écrit son programme, il le fait grâce à un langage de programmation qui est un ensemble de règles compréhensible par un humain dans le but de donner des instructions à une machine. Pour que la machine arrive à comprendre ces instructions, il faut que ce code soit traduit en langage binaire par un compilateur ou un interpréteur.

Un langage de bas niveau est un langage qui est considéré comme plus proche du langage machine que du langage humain. Il est en général plus difficile à apprendre et à utiliser mais offre plus de possibilités d'interactions avec le hardware de la machine. Un langage de haut niveau se rapproche plus du langage humain et est par conséquent plus facile à appréhender. Cependant les interactions se voient limiter aux fonctionnalités que le langage met à disposition.

La compilation (programme compilateur) consiste à transformer toutes les instructions en langage machine avant que le programme puisse être exécuté. Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source. S’il n’y a pas d’erreur dans le code, tout sera interprété et exécuté en un bloc, alors que s’il y a une erreur, l’interprétation n’aboutira pas et rien ne sera exécuté.

L’interprétation (programme interpréteur) consiste à traduire les instructions en temps réel. Le code source est donc lu à chaque exécution et les changements apportés au code seront pris en compte directement. S’il n’y a pas d’erreur dans le code, l’interprétation et l’exécution se fera requête après requête, alors que s’il y a une erreur, les requêtes seront interprétées et exécutées jusqu’à la requête comportant une erreur qui ne pourra pas être interprétée donc exécutée et rien ne se passe pour les requêtes suivantes.

L’avantage de l’interprétation est qu’il sera plus facile de trouver les erreurs dans le code mais ces interprétation et exécution successives sont plus lentes que pour la compilation où tout est fait en un seul bloc.

Le CPU est le microprocesseur, l’unité de traitement ou « cerveau » de l’ordinateur qui exécute les instructions données par les programmes. La mémoire RAM est le « mémoire de travail » qui va stocker les données et les instructions, tandis que le disque dur (HDD pu SSD) réalise un stockage permanent. Le CPU va alors rechercher les données et instruction dans la RAM (qui est plus proche du CPU que le disque dur) qui sont déjà en langage machine, exécuter donner un résultat.

Même si les langages sont différents, l’objectif principal d’un langage de programmation reste celui d’instruire la machine pour qu’elle produise des résultats conformes aux objectifs de l’application. On résume souvent la programmation comme une fonction mathématique : F(X) = Y, où X représente l’input (entrée), Y l'output (sortie) et F la fonction qui permet de transformer X en Y.

Eléments de programmation

Déclarer une variable c’est réserver un espace en mémoire pour stocker des données. L’espace dans la mémoire RAM n’est pas illimité, et s’il n’y a plus de place, le CPU va aller chercher les informations dans le disque dur qui est « plus loin ». Par opposition aux constantes, la valeur contenue dans une variable peut changer au cours du temps.

Cette variable va d’abord avoir un nom, qui lorsqu’il contient plusieurs mots doit respecter la convention camelCase (marionPerez ~~marion\_perez~~), et un contenu dont la nature doit être spécifiée lors de sa création. Les types de données pouvant être contenus dans une variable sont les suites de caractères, les chiffres, les valeurs booléennes, les tableaux et les objets pouvant inclure des sous-objets ou encore des fonctions.

L’affectation est l’opération qui consiste à attribuer une valeur à une variable. L’affectation « = » (en programmation) est donc différente de l’égalité « == » (en mathématiques) pour laquelle la relation est maintenue au cours du temps.

Sur java :

* Un seul caractère : **char nom=’a’ ;**
* Suites de caractères : **String nom=’’abcd’’ ;**
* Nombres entiers (±125) : **byte nombre=10 ;**
* Nombres entiers (±30 milles) : **short nombre=1000 ;**
* Nombres entiers (±2 milliards) : **int nombre=100000 ;**
* Nombres réels peu précis (6 décimales max) **: float nombre=0.01f ;**
* Nombres réels précis (15 décimales max) : **double nombre=0.0000001 ;**
* Valeurs booléennes (vrai/faux) : **boolean …=true ;**

La valeur de la variable peut être directement saisie par l’utilisateur ou être calculée par le processeur. On peut alors utiliser des opérateurs arithmétiques que sont l’addition (+), la soustraction (-), la division (/) et la multiplication (\*) dans l’affectation des variables.

Les structures de contrôle permettent d’exécuter seulement certaines instructions d’un programme selon la vérification d’une ou plusieurs conditions. La version la plus répandue des structures de contrôle est « si…alors » ici «**if…else** ». Il est souvent nécessaire de formuler des conditions en combinant des conditions simples au moyen d’opérateurs logiques : **&&** (et) **||** (ou).

Je peux aussi avoir besoin de fonctions mathématiques à ne pas retenir mais à savoir chercher dans la documentation. Elles sont sous la forme **Math.** …

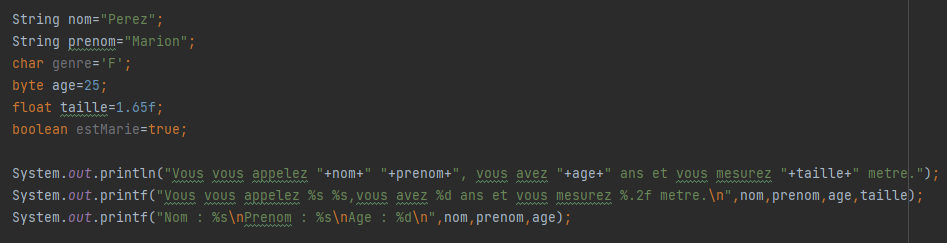
Affichage

Après l’affection je peux vouloir que l’algorithme affiche une solution écrite dans le terminal. Pour cela on utilise :

**System.out.print(‘’abcd‘’) ;**

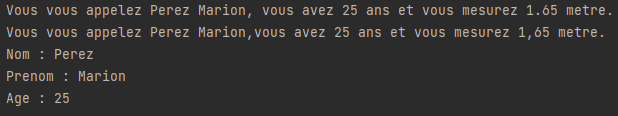
**System.out.println(‘’abcd’’) ;** si je veux en plus revenir à la ligne

Si je veux que le terminal affiche une solution comprenant des valeurs attribuées à des variables en amont, je dois les intégrer.

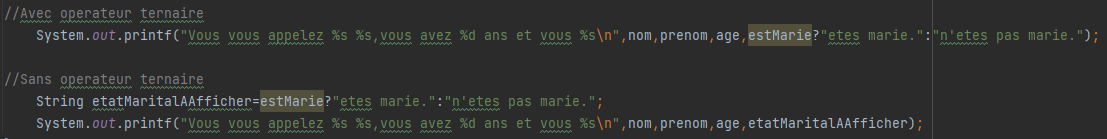


On utilise **%s** pour les chaines de caractères, **%d** pour les nombres entiers et **%f** pour les nombres décimaux. Il faut ensuite renseigner dans l’ordre à quelles variables se rapportent les insertions faites dans le texte. Si je souhaite revenir à la ligne je dois ajouter **\n**.

Ce qui donne :



Pour une écriture comprenant une valeur booléenne :



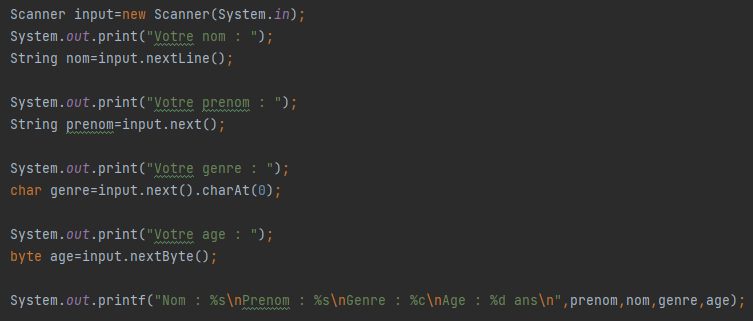
Lecture

Je peux aussi demander à l’utilisateur de donner les valeurs affectées aux variables plutôt que le déterminer en amont. Pour cela on utilise la ligne de code « Scanner ».

**Input.next()** permet de prendre la première chaine de caractère (avant un espace)

**Input.next().charAt(x)** permet de prendre un seul caractère dont la position dans la chaine est x

**Input.nextByte()** permet de prendre un nombre au format byte et **input.nextFloat()** au format float



Ce qui donne (les caractères en vert ont été renseignés par l’utilisateur) :

