Cursus JAVA

M2I Formations 2022

Olivier Blaivie

Introduction à la programmation

Olivier Blaivie

1.

Introduction à la programmation

Introduction à la programmation

- La programmation permet de **résoudre un problème** de manière automatisée grâce à l'application d'un algorithme :

Programme = Algorithme + Données

- **Algorithme** : Suite d'instructions évaluées par le processeur sur lequel est exécuté le programme.
- **Code source**: Ensemble des instructions
- Un programme s'appuie sur un langage de programmation

Langage bas niveau vs haut niveau

- Langage **Bas Niveau**:
  - Plus proche du langage machine (binaire)
  - Plus difficile à apprendre et à utiliser
  - Plus de possibilité d'interactions (peu de contraintes)

- Langage **Haut Niveau**:
  - Plus proche du langage humain (C, Python, Java ...)
  - Plus facile à appréhender
  - Interactions limités aux fonctionnalités proposées par le langage de programmation

Compilation vs Interprétation

- **Compilation** d'un programme :

  Transformation des instructions en langage machine avant qu'il ne soit exécuté.

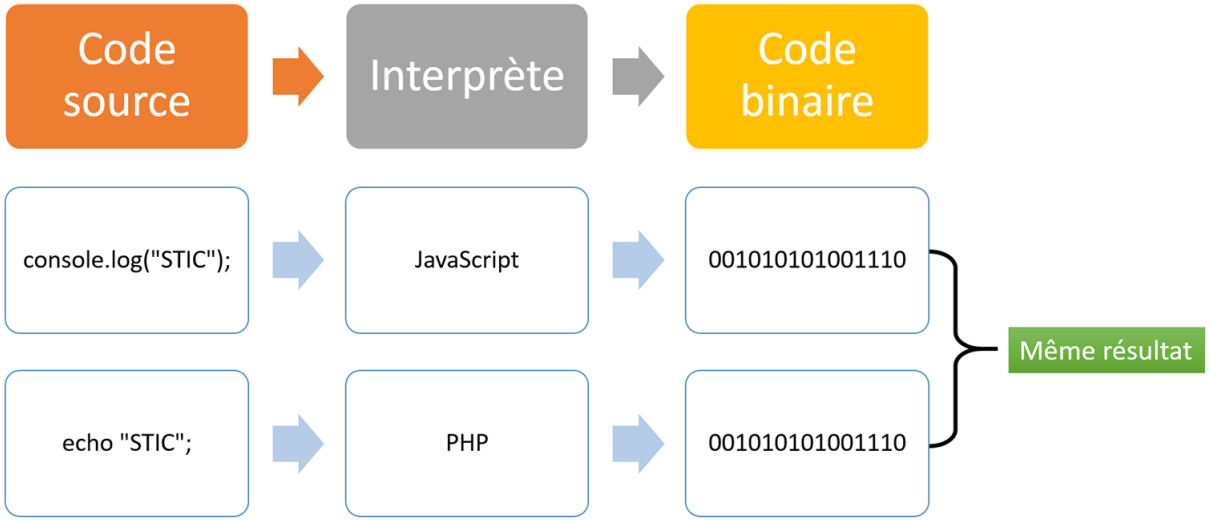
- **L'interprétation** d'un programme :

  Traduction en temps réel des instructions (à l'aide d'un interpréteur)

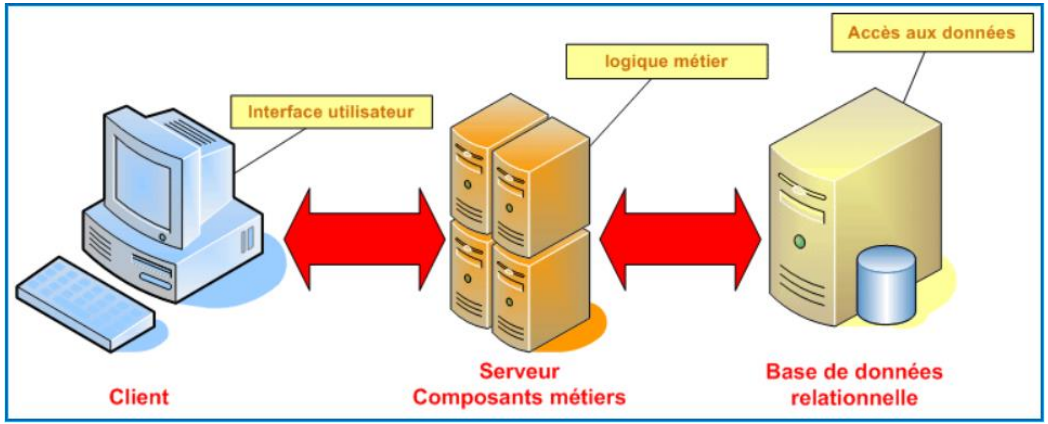
  Seule contrainte : La machine hôte doit disposer de l'interpréteur adapté au langage

Introduction à la programmation

- Alors c'est pas des 0 et des 1 le code ?

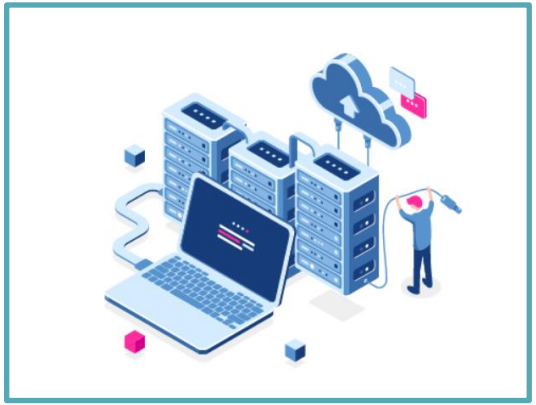


Client léger vs client lourd (1/2)



Client Léger

Client Lourd

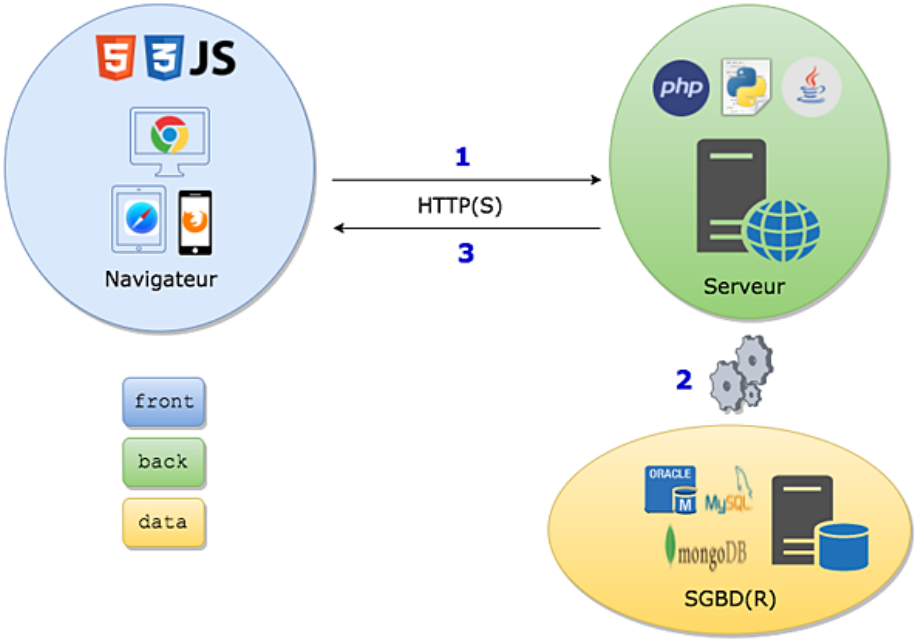


Client léger vs client lourd (2/2)

	CLIENT LOURD (OU RICHE)	CLIENT LEGER
Graphique	Capacité a fournir des interfaces de haute qualité	Graphisme Web (souvent ressemblant d'un site à un autre)
Connexion	Utilisation Hors ligne possible	Connexion Internet indispensable
Utilisation	Nécessite une installation / Utilisation sur un seul poste	Utilisation partout et depuis n'importe quel type d'appareil (smartphone, tablette ...)
Manipulation	Local donc plus permissif (sauf si droit Admin demandé)	Téléchargement nécessaire pour manipuler en local
Update	Mise à jour par installation...	Transparent et non contraignant pour le client
Performances	Selon la machine hôte...	Selon server (ou Cloud)
Exemples	Pack Office, Calculatrice, Discord, Fifa...	Google Maps, Discord, Eurosport...

Architecture 3 tiers

- Anatomie d'une application interactive



Vocabulaire de développement

- **Les éléments fondamentaux :**
  - Les variables
  - Les opérateurs
  - Les structures de contrôle (Conditions)
  - Les boucles
  - Les fonctions
  - Les tableaux (ou array)
  - Les objets

Les variables

- Une variable permet de déclarer une entité avec une **valeur** avec un **nom**.

- Utilité des variables :
  - **Déclaration** : Permet de créer une instance
  - **Affectation** : Permet de donner une valeur à une instance

- Les types de variables :
  - Les **suites de caractères** (**String**): elles sont utilisées pour représenter du texte ;
  - Les **chiffres** (*nombre entier, à virgule flottante, etc.)* : ils sont utilisés surtout avec des opérateurs mathématiques ;
  - Les **valeurs booléennes (Booleans)**: elles sont des valeurs dichotomiques (soit vrai, soit faux) ;
  - Les **tableaux** (**Array**): ils sont utilisés pour créer des listes et regrouper plusiers données
  - Les **objets** (**Object**): ils sont des conteneurs qui peuvent inclure souvent tout type de données, y compris de sous-objets, des variables (i.e. des propriétés), ou des fonctions (i.e. méthodes)

Les opérateurs

Les opérateurs permettent de **manipuler** ou **comparer** des valeurs, notamment des variables. Parmi les opérateurs utilisés fréquemment dans tout langage de programmation vous trouverez :

- Les opérateurs **mathématiques**: addition, soustraction, multiplication, division, etc.
- Les opérateurs de **comparaison** (égalité, différence, majeur ou mineur)
- Les opérateurs **logiques** (AND et OR)

Les structures de contrôle

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la **vérification** d'une ou plusieurs **conditions**.

La version sémantique la plus répandue des structures de contrôle est « **si… alors…** ».

- Si la note d'un examen est mineur de 4, alors la note est insuffisante
- Si le joueur n'a pas débloqué une porte, alors il ne peut pas accéder au niveau supérieur
- Si le mot de passe choisi contient moins de 6 caractères, alors il est trop court
- Si l'extension du fichier n'est pas .pdf, alors ne le téléverse pas sur le serveur

En général elles sont utilisées pour déterminer si certaines instructions doivent s'exécuter ou non :

- Si cette condition s'avère, alors…
  Exécute l'instruction #1
  Exécute l'instruction #2
  Exécute l'instruction #9
- Si non…
  Exécute l'instruction #7
  Exécute l'instruction #10

Les boucles

Les boucles sont à la base d'un concept très utile en programmation: l'**itération.**

L'itération permet d'exécuter de manière **récursive** (plusieurs fois) des instructions. Elles peuvent être très utile par exemple pour appliquer un traitement à une liste d'éléments.

- Exemple d'utilisation :

Tant que la liste n'est pas totalement parcouru :

Modifier un élément de la liste

- Ou encore :

Tant que ce chiffre ne dépasse pas 16 :

Réalise un calcul

Algorithmie

- Un algorithme est une **représentation** de la **logique** d'un programme

- Un algorithme peut être écrit en langage **naturel** ou en langage de **programmation**

- Un algorithme est défini en 3 parties :

  La **déclaration** des informations nécessaires au programme

  Les **instructions**

  Le **résultat**

- Les variables ont un type à déclarer : **entier**, **chaine**, **boolean**, **list**

Exemple 1

- **Problème :**

Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

- **Solution :**

```
Variables
  n entier
Début
  n <- 0
  Ecrire "Entrez un nombre entre 1 et 3"
  Saisir n
  TantQue n<1 ou n>3
    Lire n
    Si n <1 ou n>1 Alors
      Ecrire "Saisie erronée. Recommencez"
      Saisir n
    FinSi
  FinTantQue
Fin
```

Exemple 2

- **Problème :**

Ecrire un algorithme qui affiche tous les nombres de 0 à un nombre positif saisi par l'utilisateur

- **Solution :**

```
Variables
  n,i entier
Début
  n <- 0
  Ecrire "Entrez un nombre positif"
  Saisir n
  Lire n
  Si n < 0 Alors
    Ecrire "Saisie erronée. Recommencez"
  SinonSi n > 0 Alors
    Pour i <- 0 a n
      Ecrire i
    i Suivant
  FinSi
Fin
```

Les fonctions

Les fonctions représentent une sorte de "*programme dans le programme*".

Elles sont la première forme **d'organisation** du code.

On utilise des fonctions pour **regrouper des instructions** et les appeler sur demande: chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions. Pour accomplir ce rôle, le cycle de vie d'une fonction se divise en deux :

1. **Une phase unique dans laquelle la fonction est déclarée**
   On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité

2. **Une phase qui peut être répétée une ou plusieurs fois dans laquelle la fonction est exécutée**
   On demande à la fonction de mener à bien toutes les instructions dont elle se compose à un moment donnée dans la logique de notre application

Les listes (ou tableaux)

Les tableaux (*Array*) sont des listes indexées d'éléments qui partagent normalement une certaine relation sémantique pour appartenir à la liste.

Un exemple tout simple d'array est la liste des courses: on peut indexer cette liste en fonction de l'ordre des articles à acheter :

1. Lait
2. Farine
3. Pommes
4. Fromage

Attention: l'index d'un tableau commence à **0**. En prenant l'exemple précédent on dira que l'élément "Lait" est dans la première case du tableau qui est à l'index 0.

De la même façon l'élément "Fromage" est l'élément dans la 4ième case du tableau qui se trouve à l'index 3.

Exemple 3

- **Problème :**

Ecrire une fonction qui additionne tous les nombres d'une liste (passée en paramètres) et retourne le résultat

- **Solution :**

```
Fonction compte(Liste l): entier
  Variables
    i, result entier
  Début
    result <- 0
    Pour i <- 0 longueur(l)
      result <- result + l[i]
    i Suivant
    retourne result
  Fin
Fin
```

Les objets

On dit souvent en informatique qu'un langage est "**orienté objets**".

Un **objet** est un élément qui possède un nom, des caractéristiques (**propriétés ou attributs**) et qui peut normalement exécuter certaines fonctions (**méthodes**).

Exemple salade de fruits :

Tous les fruits de ma salade de fruit ont des caractéristiques et fonctions différentes, on peut donc les regrouper en un objet **Fruit**.

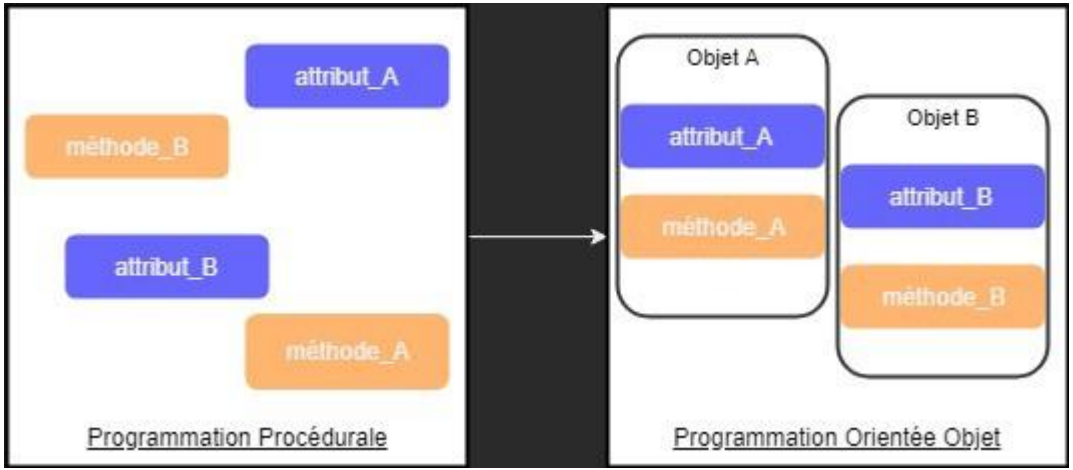
1. Pomme (Ronde, avec pépins, croquante)
2. Cerise (Ronde, avec noyau, juteuse)
3. Kiwi (Ovale, avec pépins, juteux)

Du coup on peut créer un objet Fruit avec les propriétés **Nom**, **Forme**, **PepinOuNoyau**, **Texture** et les méthodes **eplucher()**, **denoyauter()**, **manger()**

Introduction à la programmation

- **L'encapsulation**

C'est ce qu'on appelle l'**encapsulation**. Cela permet de protéger l'information contenue dans notre objet et de le rendre manipulable uniquement par ses actions ou propriétés



Introduction à la programmation

- **L'héritage**

Un objet dit « père » peut transmettre certaines de ses caractéristiques à un autre objet dit « fils ».

Pour cela, on pourra définir une relation d'héritage entre eux. S'il y a une relation d'héritage entre un objet père et un objet fils, alors **l'objet fils hérite de l'objet père**. On dit également que l'objet fils est une **spécialisation** de l'objet père ou qu'**il dérive de l'objet père**.