1. Introduction :

L'**algorithmique** et la **programmation** sont deux notions fondamentales en informatique. Elles sont liées, mais elles ne signifient pas exactement la même chose.

L'algorithmique est la base de toute programmation. Elle permet de développer une **pensée logique et structurée** indispensable en informatique.

La programmation, quant à elle, permet de transformer ces idées en applications concrètes : logiciels, sites web, jeux vidéo, applications mobiles, intelligence artificielle, etc.

A. L'algorithmique :

L'**algorithmique** est la science qui consiste à concevoir et décrire des **algorithmes**, c'est-à-dire des suites d'instructions claires et ordonnées permettant de résoudre un problème.

Un **algorithme** doit être :

- ✔ **Précis** : chaque étape doit être clairement définie
- ✔ **Fini** : il doit se terminer après un nombre limité d'étapes
- ✔ **Efficace** : il doit produire le résultat attendu.

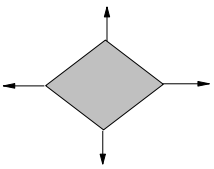
B. Représentation graphique ou organigramme d'un algorithme

La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l'inconvénient de consommer une place importante. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin. C'est ainsi qu'on a:

est utilisé pour les calculs

est utilisé pour la lecture et l'affichage (Parallélogramme)

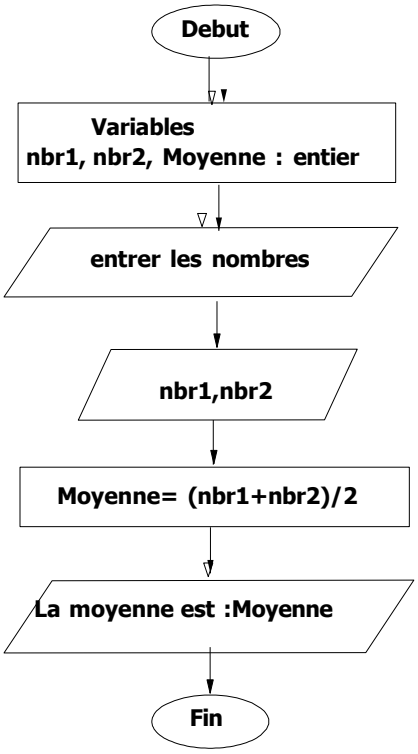
est utilisé pour le début et fin



utilisé pour représenter des testes

Exemple : Calculer la moyenne de deux nombres

1. Lire le premier nombre
2. Lire le deuxième nombre
3. Additionner les deux nombres
4. Diviser la somme par 2
5. Afficher le résultat



L'algorithmique ne dépend d'aucun langage informatique : on peut écrire un algorithme en français, en pseudo-code ou sous forme de schéma.

C. Forme pseudo-code (forme standard en algorithmique)

Plus structurée et proche d'un langage de programmation.

```
Algorithme Moyenne
   Variables : a, b, m : réel ;
Début
   Ecrire('Donner la valeur de a :' , a) ;
   Lire (a) ;
   Ecrire('Donner la valeur de b :' , b) ;
   Lire (b) ;
   m ← (a + b) / 2 ;
   écrire ('Afficher la moynne : ', m) ;
Fin
```

◆ **C'est la forme la plus utilisée en cours d'informatique.**

D. La programmation :

La **programmation** consiste à traduire un algorithme dans un **langage informatique** afin qu'un ordinateur puisse l'exécuter.

Un programme est un algorithme écrit dans un langage que l'ordinateur comprend. C'est à dire un ensemble d'instructions écrites dans un **langage de programmation** et destinées à être exécutées par un ordinateur afin de réaliser une tâche précise.

E. Les types de langages de programmation en informatique

Les langages de programmation peuvent être classés selon leur niveau d'abstraction, leur manière de fonctionner ou leur utilisation.

1 Langages de bas niveau

Les **langages de bas niveau** sont des langages de programmation très proches du fonctionnement interne de l'ordinateur. Ils permettent de communiquer presque directement avec le matériel (processeur, mémoire). Parmi ces langages :

a) Langage machine :

- C'est le langage **compréhensible directement par l'ordinateur**.
- Il est composé uniquement de **0 et de 1** (code binaire).
- Chaque processeur a son propre langage machine.
- **Inconvénient :**

  Très difficile à écrire et à comprendre pour un humain.

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

b) Langage assembleur :

- Plus lisible que le langage machine.
- Utilise des **mots symboliques** (ADD, MOV, SUB…).
- Doit être traduit en langage machine par un programme appelé **assembleur**.
- **Avantage :**

  Permet un contrôle précis du matériel et une grande performance.

```
01 .MODEL SMALL
02 .STACK 100H
03 .CODE
04
05 MOV AX, 0x3C
06 MOV BX, 0000000000001010B
07 ADD AX, BX
08 MOV BX, 14
09 SUB AX, BX
10
11 MOV AH, 04CH
12 INT 21H
```

2 Langages de haut niveau :

Les **langages de haut niveau** sont des langages de programmation proches du langage humain et plus faciles à comprendre, écrire et maintenir que les langages de bas niveau. Ils sont conçus pour simplifier la programmation et être **indépendants du matériel**.

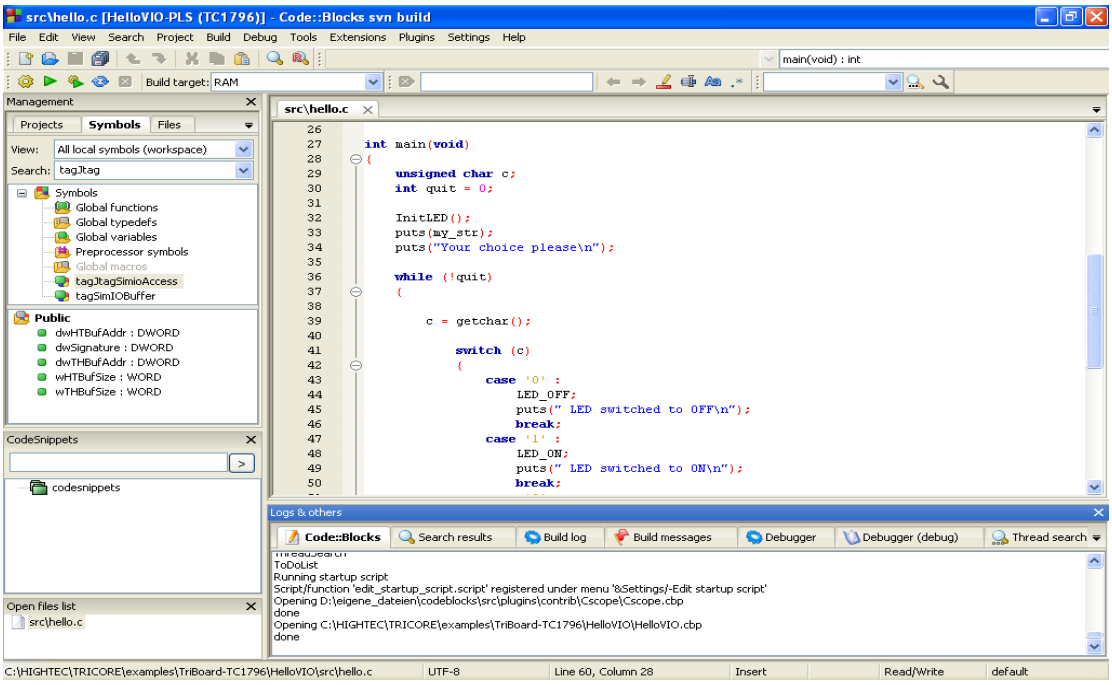
- **Caractéristiques principales**
  - ✔ Syntaxe simple et lisible
  - ✔ Portables (fonctionnent sur plusieurs systèmes)
  - ✔ Plus faciles à apprendre
  - ✔ Nécessitent un compilateur ou un interpréteur
- **Exemples de langages de haut niveau**

```
504             if not hasattr(self, '_headers_buffer'):
505                 self._headers_buffer = []
506             self._headers_buffer.append(("%s %d %s\r\n" %
507                     (self.protocol_version, code, message)).encode(
508                         'latin-1', 'strict'))
509
510     def send_header(self, keyword, value):
511         """Send a MIME header to the headers buffer."""
512         if self.request_version != 'HTTP/0.9':
513             if not hasattr(self, '_headers_buffer'):
514                 self._headers_buffer = []
515             self._headers_buffer.append(
516                 ("%s: %s\r\n" % (keyword, value)).encode('latin-1', 'strict'))
517
518         if keyword.lower() == 'connection':
519             if value.lower() == 'close':
520                 self.close_connection = True
521             elif value.lower() == 'keep-alive':
522                 self.close_connection = False
```

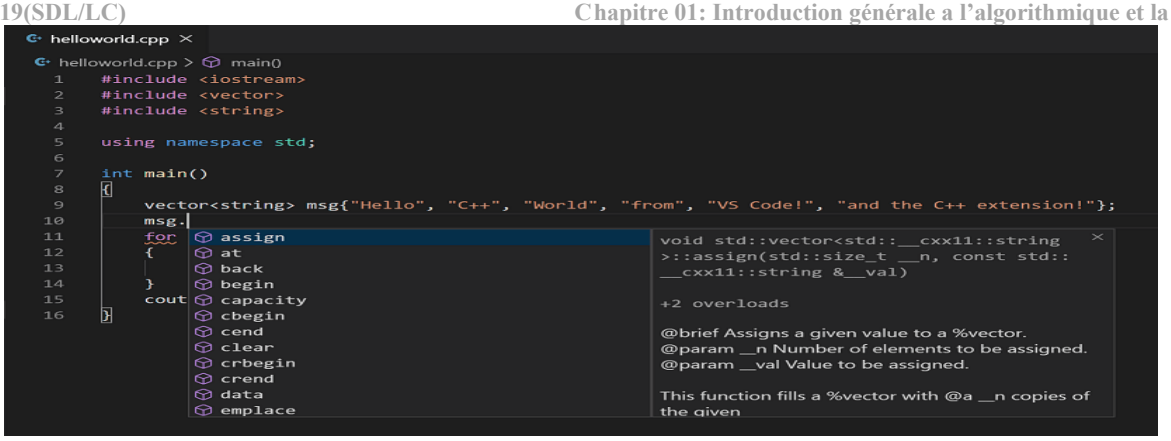
- **Langage de programmation: Python**

```java
import java.io.*;
import java.util.Date;

public class SaveDate {

  public static void main(String args[]) throws Exception {
    FileOutputStream fos = new FileOutputStream("date.ser");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    Date date = new Date();
    oos.writeObject(date);
    oos.flush();
    oos.close();
    fos.close();
  }
```

- **Langage de programmation : Java**



- **Langage de programmation :C++**



- **Langage de programmation : JavaScript**

➢ **Avantages**

- Développement plus rapide
- Moins d'erreurs
- Maintenance plus simple
- Utilisés pour créer des logiciels, sites web, applications mobiles, jeux, etc.

3 Types selon le paradigme de programmation

Un paradigme de programmation est une manière ou une approche pour écrire et organiser un programme. Selon le paradigme utilisé, la façon de penser et de structurer le code change.

3.1. Programmation procédurale :

La programmation procédurale est un paradigme de programmation basé sur l'organisation du programme en procédures ou fonctions. Un programme est structuré comme une suite d'instructions exécutées étape par étape pour résoudre un problème.

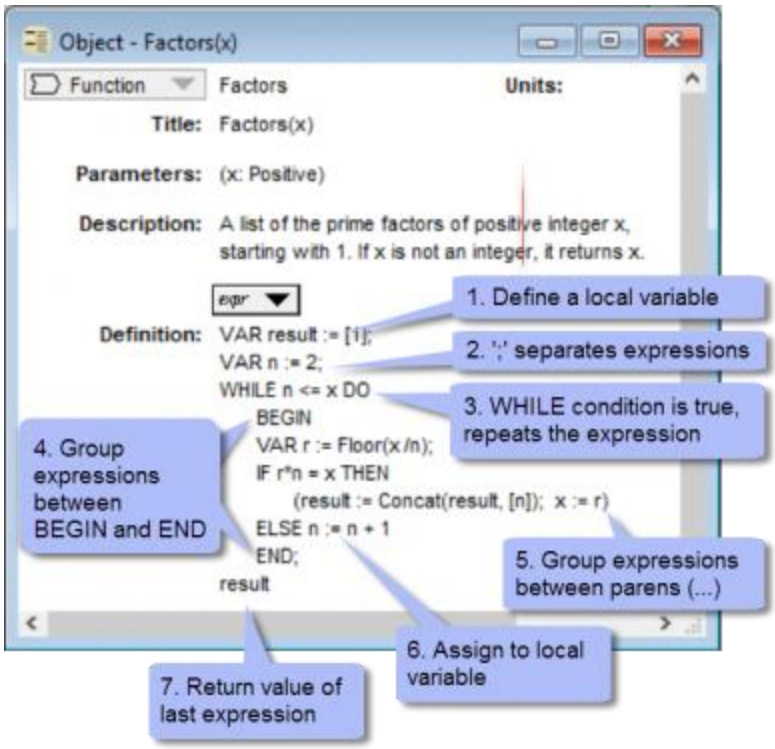
➢ **Caractéristiques**

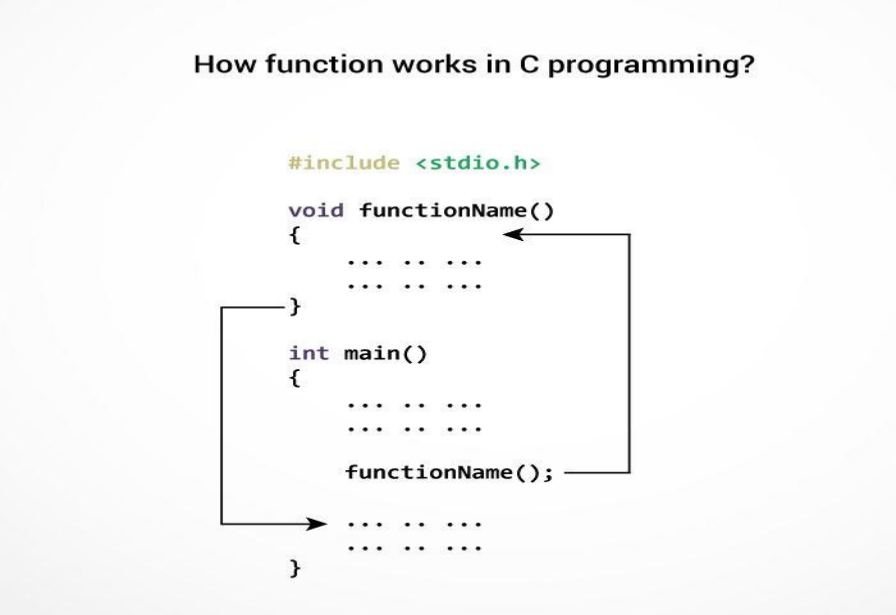
- Programme organisé en fonctions.
- Exécution séquentielle (ligne par ligne).
- Utilise des variables et des structures de contrôle (si, boucle…).
- Simple et adaptée aux petits programmes

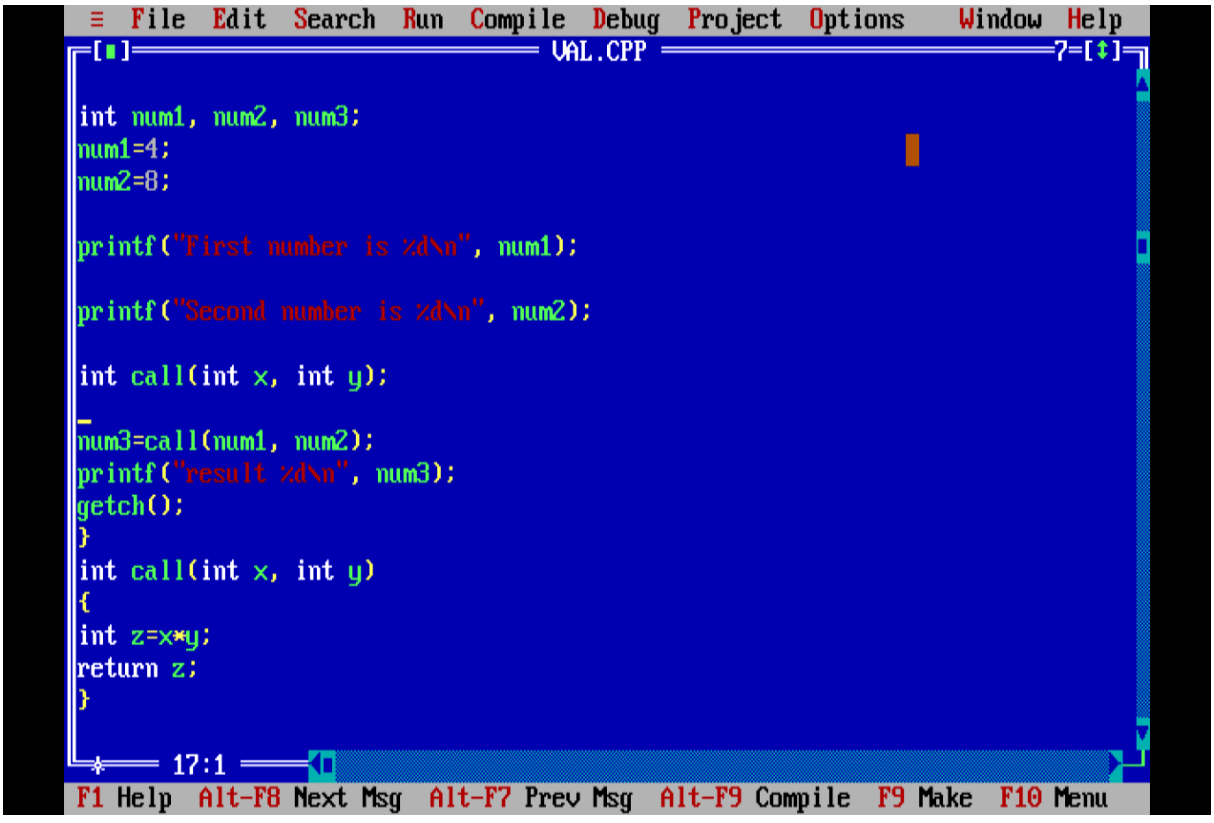
➢ **Exemple : C**

```c
#include <stdio.h>

int main() {
   int a, b, c;

    /* Input three numbers from user */
    printf("Enter three numbers: ");
    scanf("%d%d%d", &a, &b, &c);

   if ( a > b && a > c )
      printf("%d is the largest.", a);
   else if ( b > a && b > c )
      printf("%d is the largest.", b);
   else if ( c > a && c > b )
      printf("%d is the largest.", c);
   else
      printf("Values are not unique");

   return 0;
}
```

```
Enter three numbers: 10
20
15
20 is the largest.
```







2 Programmation orientée objet (POO)

La Programmation Orientée Objet (POO) est un paradigme de programmation basé sur la notion d'objets.

Un objet regroupe :

- des données (attributs)

- des fonctions (méthodes)

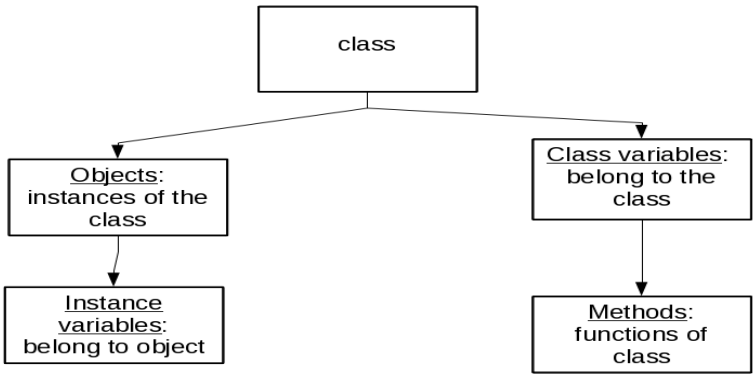
Définition simple

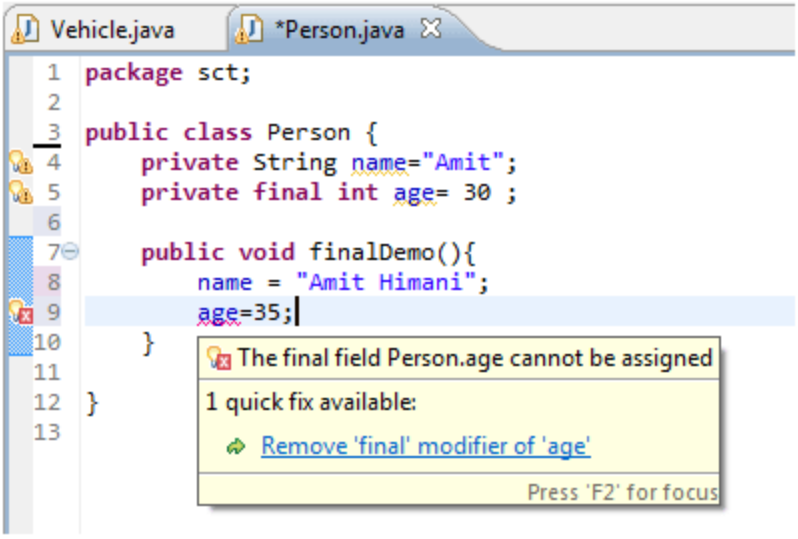
La programmation orientée objet est un style de programmation où le programme est organisé autour d'objets qui contiennent des données et des méthodes.

- **Concepts fondamentaux de la POO**
  1. **Classe :** modèle ou plan pour créer des objets
  2. **Objet** : instance d'une classe
  3. **Encapsulation** : protection des données
  4. **Héritage :** création d'une nouvelle classe à partir d'une autre
  5. **Polymorphisme :** capacité d'un objet à prendre plusieurs formes

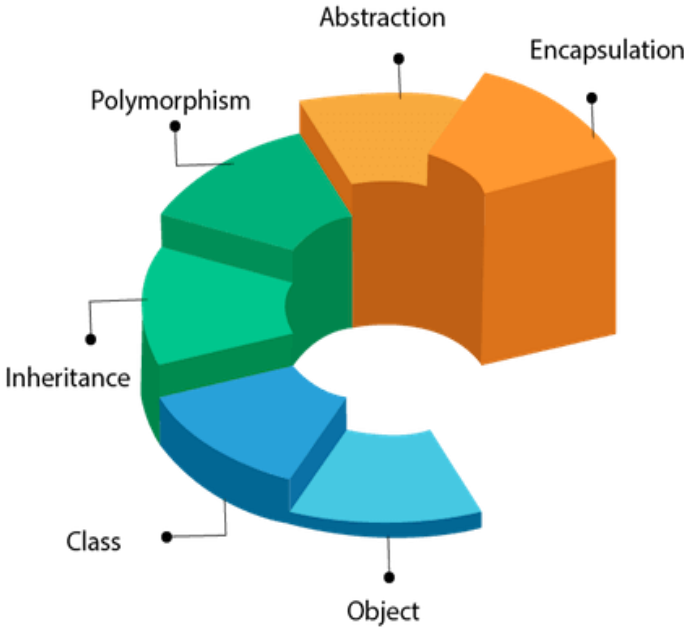
- **Exemples de langages utilisant la POO**
  - **Java**
  - **C++**
  - **Python**
- **Avantages**
  - Bonne organisation du code
  - Réutilisation facile (héritage)
  - Adaptée aux grands projets
- **Inconvénients**
  - Plus complexe que la programmation procédurale
  - Demande une bonne compréhension des concepts

Object Oriented Programming





OOPs (Object-Oriented Programming System)



3 Programmation fonctionnelle

La programmation fonctionnelle est un paradigme de programmation basé sur les fonctions mathématiques.

Elle considère le programme comme une évaluation d'expressions plutôt qu'une suite d'instructions.

a). Principe de base

En programmation fonctionnelle :

- ✅ On utilise des fonctions pures
- ✅ On évite de modifier les données (immutabilité)
- ✅ On évite les effets de bord
- ✅ Les fonctions peuvent être passées en paramètre

b) C'est quoi une fonction pure ?

Une fonction pure :

- Donne toujours le même résultat pour les mêmes paramètres
- Ne modifie aucune donnée extérieure

Exemple en Python :

```
def carre(x):
    return x * x
```

application : carre(4) donnera toujours 16.

---

c) .Caractéristiques principales

a) Immutabilité

Les données ne sont pas modifiées, on crée de nouvelles valeurs.

b) Fonctions d'ordre supérieur

Une fonction peut :

- Recevoir une fonction en paramètre
- Retourner une fonction

Exemple :

```
def appliquer(f, x):
    return f(x)
print(appliquer(carre, 5))
```

c) Récursion

On utilise souvent la récursion au lieu des boucles.

Exemple :

```
def factorielle(n):
    if n == 0:
        return 1
    return n * factorielle(n-1)
```

D) Langages fonctionnels connus

- Haskell
- Lisp
- Scala
- **JavaScript (supporte le style fonctionnel)**

🔷 5) Différence avec la programmation impérative

Programmation impérative	Programmation fonctionnelle
Modifie les variables	Évite la modification
Utilise beaucoup de boucles	Utilise la récursion
Basée sur les instructions	Basée sur les fonctions

➢ **Avantages**

✔ Code plus clair

✔ Moins d'erreurs

✔ Facile à tester

✔ Très adaptée au parallélisme

```
const formatABBRVName = (fullname) => {
  const splitted = splitBy(
    Maybe(fullname),
    " ",
  );

  return reduce(
    splitted,
    (acc, elm, index, arr) => {
      if (index === arr.length - 1) {
        return acc + `${toUpperCase(Maybe(elm)).value}`
      }

      return acc + `${toUpperCase(getFirstLetter(Maybe(elm))).value}. `
    }
    , ""
  )
}
```

```
def sub(num1, num2):
    print("type number 1:", num1)
    print("type number 2:", num2)
    subtraction = num1 - num2

    return subtraction
```

```
result = sub(10, 5)
print(result)
```

```
type number 1: 10
type number 2: 5
5
```

Lambda Syntax

- No arguments:
```
() -> System.out.println("Hello")
```
- One argument:
```
s -> System.out.println(s)
```
- Two arguments:
```
(x, y) -> x + y
```
- With explicit argument types:
```
(Integer x, Integer y) -> x + y
```
- Multiple statements:
```
(x, y) -> {
    System.out.println(x);
    System.out.println(y);
    return (x + y);
}
```

6

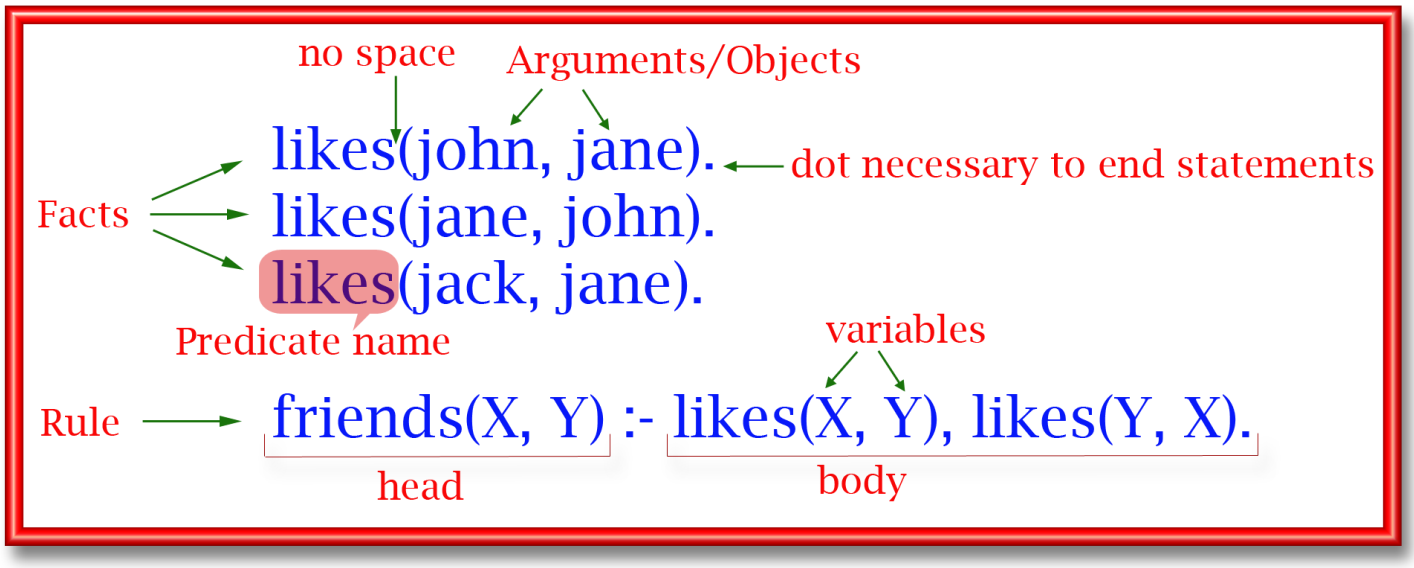
4 Programmation logique

Query Window

```
?- likes(john, jane).   ← dot necessary
true.                   ← answer from prolog interpreter

sign on
prolog query
prompt                      variables
                            ↙  ↘
?- friends(X, Y).
X = john,
Y = jane ;   ← type ; to get next solution
X = jane,
Y = john.
```

```
% Background knowledge:
    isFather("John", "Dan") % John is Dan's father
    isFather("Paul", "John") % Paul is John's father
    isWife("Alice", "Paul") % Alice is Paul's wife

    isGrandmother(X,Y):-isWife(X,Z),
                        isGrandfather(Z,Y) % X is Y's grandmother
                                           % if X is Z's wife and
                                           % Z is Y's grandfather

    isGrandfather(X,Y):-isFather(X,Z),
                        isFather(Z,Y)  % X is Y's grandfather
                                       % if X is Z's father and
                                       % Z is Y's father

% Logic programming query:
    ?-isGrandmother("Alice","Dan") % Is Alice Dan's grandmother?
```

Program Window



5. Write the following ten logic rules in Prolog notation:

(a) If Fred is a father of Mike, then Fred is an ancestor of Mike.
(b) An animal is a mammal if it is a human or its parents were mammals.
(c) You have attained the ultimate state if you are happy, healthy, and wise.
(d) Every dog likes all people.
(e) The Lakers will win games 2, 3, 5, and 7, but lose the other 3 games in the series with New Orleans.
(f) $(P \cap Q) \Rightarrow (R \cup S)$ (i.e., If P and Q, then R or S)
(g) $(P \Rightarrow Q) \Leftrightarrow (\sim P \cup Q)$ (i.e., P implies Q is equivalent to the disjunction of not P with Q)
(h) P *exclusive_or* Q is when P *inclusive_or* Q, but *not* (P *and* Q).
(i) Jack is disappointed when it rains and any student misses class.
(j) To be or not to be, that is the question.

🔷 **Basée sur la logique mathématique**

🔷 **Le programme décrit des faits et des règles**

🔷 **L'ordinateur déduit la solution**

📌 **Exemple : Prolog**

🎯 **Résumé**

Paradigme	**Principe**
Procédural	**Suite d'instructions**
Orienté objet	**Objets et classes**
Fonctionnel	**Fonctions mathématiques**
Logique	**Règles et déductions**

Un paradigme est une façon de programmer.

🔷 a) Programmation procédurale

- **Basée sur des procédures ou fonctions.**
- **Exemple : C**

🔷 b) Programmation orientée objet (POO)

- **Basée sur des objets (données + méthodes).**
- **Exemples : Java, C++**

🔷 **c) Programmation fonctionnelle**

- **Basée sur les fonctions mathématiques.**
- **Exemple : Haskell**

🔷 **d) Programmation logique**

- **Basée sur la logique mathématique.**
- **Exemple : Prolog**

4️⃣ **Types selon l'utilisation**

🌐 **Développement Web**

- **JavaScript**
- **PHP**

📱 **Applications mobiles**

- **Kotlin**
- **Swift**

🤖 **Intelligence artificielle**

- **Python**

🎯 **Conclusion**

Les langages de programmation sont nombreux et variés.

Le choix dépend de :

- **L'objectif du projet**
- **Le domaine d'application**
- **Les performances recherchées**

Si tu veux, je peux aussi te faire un résumé très court pour examen 📚 ou un tableau comparatif détaillé 😊

3. Relation entre algorithmique et programmation

On peut résumer ainsi :

🧠 **Algorithmique = réflexion et conception de la solution**

⌨️ **Programmation = mise en œuvre de la solution sur ordinateur**

Avant de programmer, il est essentiel de :

1. Comprendre le problème
2. Concevoir l'algorithme
3. Tester la logique
4. Traduire en code

---

4. Concepts fondamentaux

En algorithmique et en programmation, on retrouve des notions essentielles :

- 🔢 **Variables** : pour stocker des données
- 🔀 **Conditions** (si… alors… sinon)
-
- 🔁 **Boucles** (répéter des instructions)

- **Fonctions** : blocs d'instructions réutilisables
- **Structures de données** : tableaux, listes, etc.