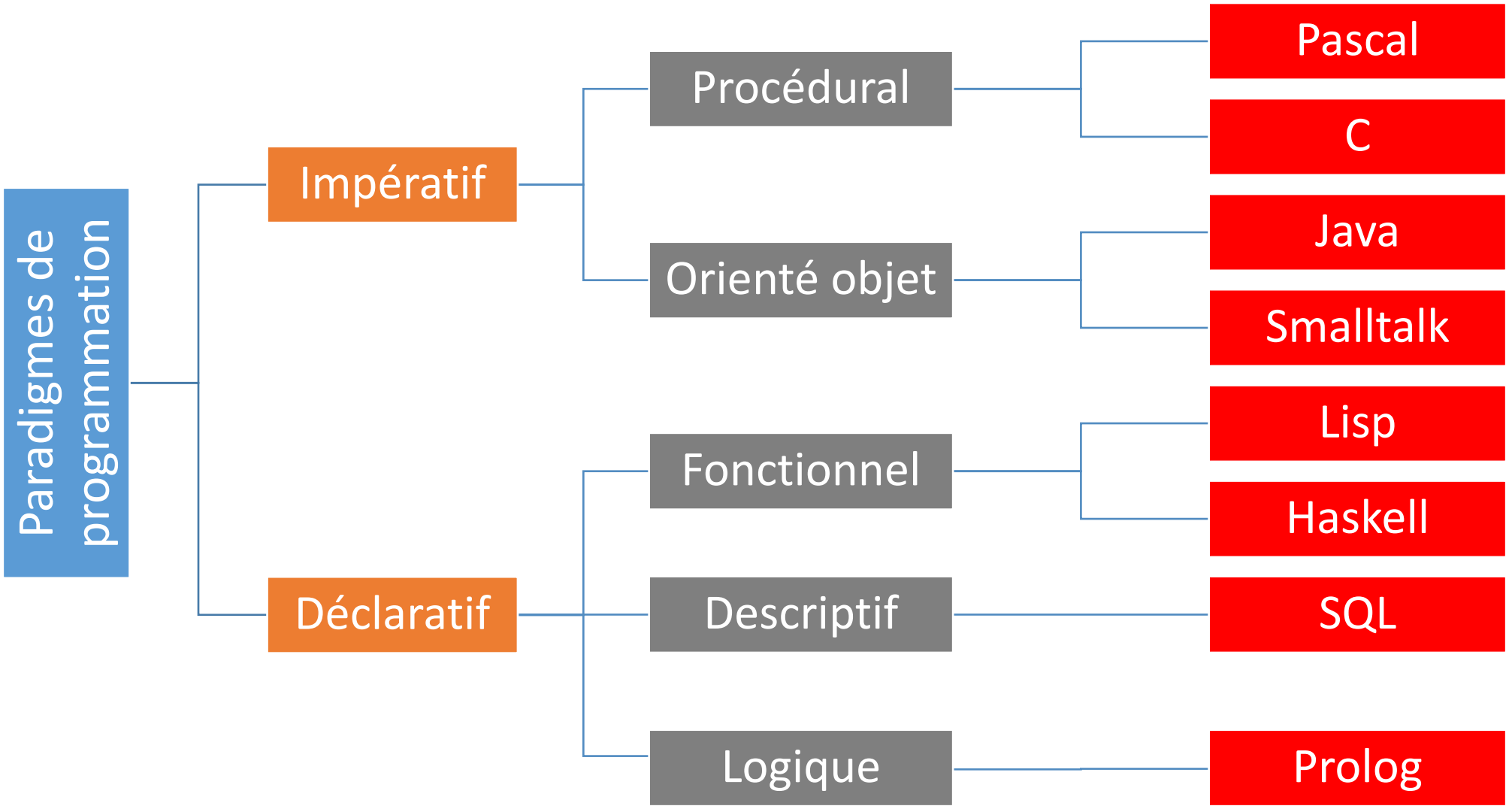
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Jijel
Faculté des Sciences exactes et de l'informatique
Département d'informatique

– Module –
Environnements et Programmation Dédiés

Master 1 : EPD

Enseignant du module : Dr. Hemza FICEL

Contact: hemza.ficel@univ-jijel.dz


Paradigmes de programmation
Impératif
Déclaratif
Procédural
Orienté objet
Fonctionnel
Descriptif
Logique
Pascal
C
Java
Smalltalk
Lisp
Haskell
SQL
Prolog

- **Le paradigme déclaratif** exprime la logique d'un calcul sans décrire son flux de contrôle.

```
-- Requête sql

SELECT first_name, last_name, age  FROM Customers WHERE age>25 ;
```

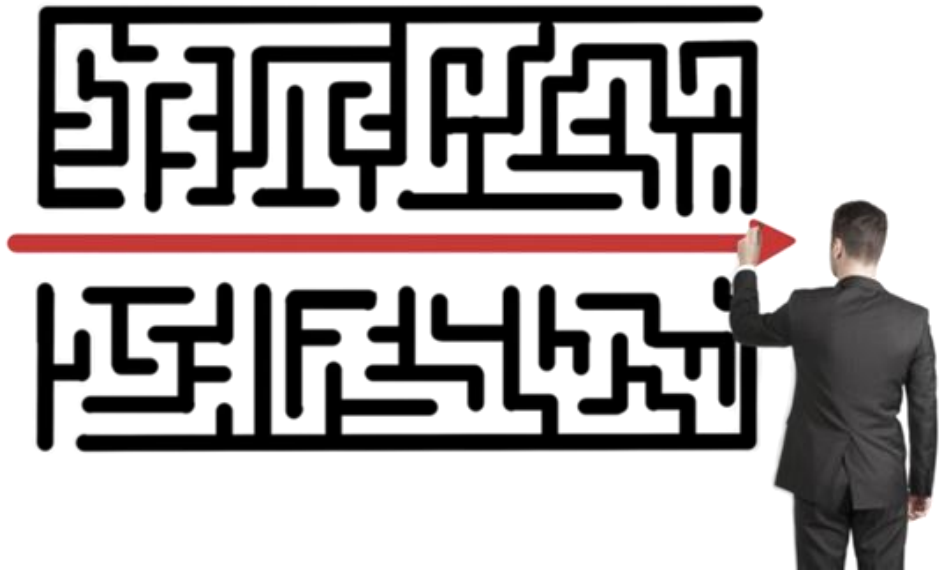
- Il se concentre sur ce que le programme devrait accomplir (**quoi ?**).

Chapitre 4
Programmation Fonctionnelle

Programmation Fonctionnelle

“ You will learn an entirely new way of thinking. Your brain is going to rebel a little. Just keep moving forward ! ”

La programmation fonctionnelle : Pourquoi faire compliqué quand on peut faire simple ?



Les nombres paires dans un tableau …

La programmation procédurale

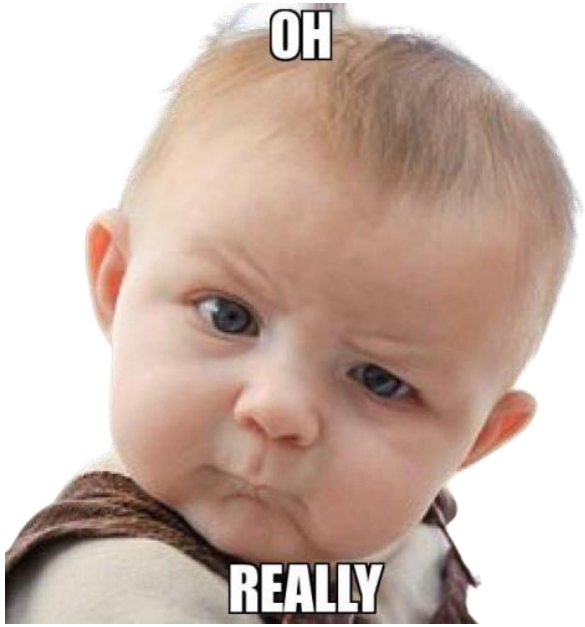
```
// Langage de programmation C
int isEven(int v) {
      return v%2 == 0;
}
int main(){
   int array [5] = {1,2,3,4,5};
   int evens[] = {};
   int pos = 0;
   for (int i = 0; i < 5; i++) {
      if (isEven(array[i])==1){
         evens[pos] = array[i];
         pos = pos+1;
      }
   }
 return 0;
}
```

La programmation fonctionnelle

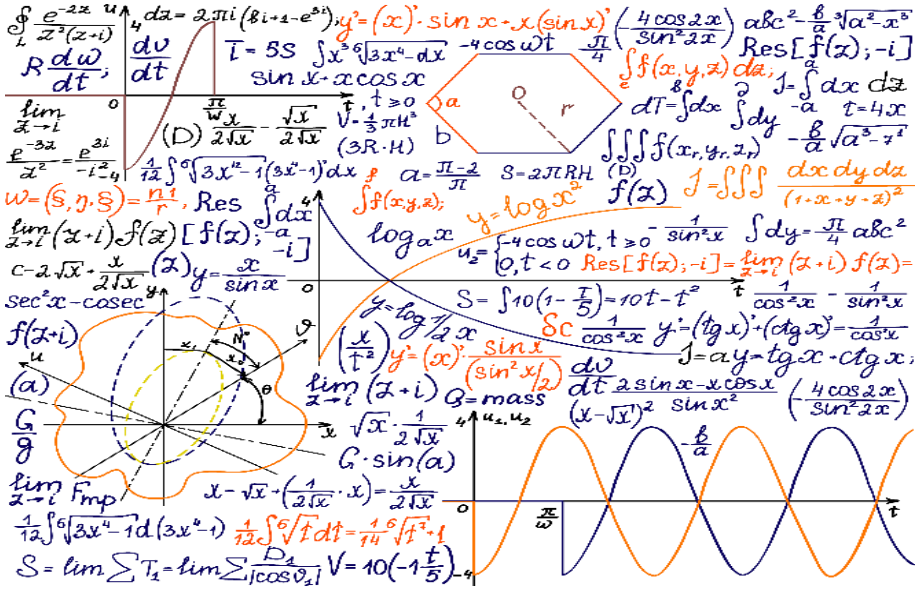
```
// Langage de programmation Javascript
let isEven = function(v) {
  return v % 2 == 0;
}
let array = [1,2,3,4,5];
let evens = array.filter(isEven);
```

Programmation Fonctionnelle

La source d'inspiration de la programmation fonctionnelle sont les mathématiques.



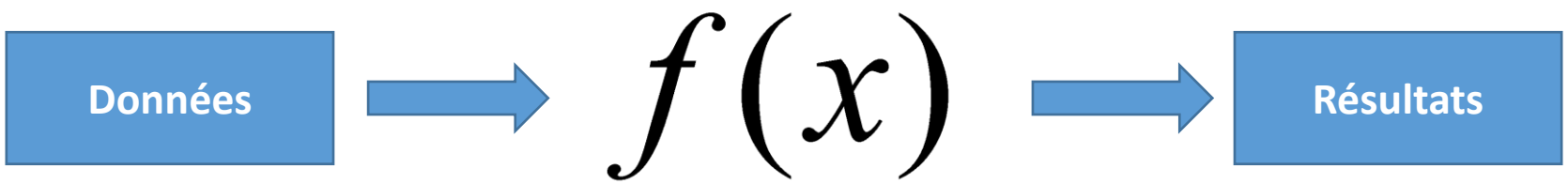
Simplicité = math ?



Principe de base

Fondamental

- **Le principe central de la programmation fonctionnelle** : **tout est fonction** !



- Une fonction prend des valeurs en entrée (**paramètres**), effectue des calcules (**traitements**) et génère un résultat (**valeur de retour**).

Principe de base

Fonction au sens mathématique

$$f(x) = x^2$$
$$f(1) = 1$$
$$f(2) = 4$$
$$f(3) = 6$$
$$f(4) = 16$$
$$f(5) = 25$$
$$\ldots\ldots\ldots$$

Fonction au sens informatique

```
int swap(int* a, int* b){
  int tmp = *a;
   *a = *b;
   *b = temp;
   printf("%d, %d",a,b);
   return 1;
}
```

La différence ????

Principe de base

La différence ????

- **Le principe mathématique de la fonction** : les fonctions ne vont pas **modifier des variables, écrire sur un écran, interroger une base de données, etc**.

Fondamental

- **Dans la programmation fonctionnelle**, **c'est la même chose** ! Les fonctions n'effectuent que **des traitements qui retournent un résultat**.

Principe de base

Concrètement

Pas de fichiers !!

Pas d'événements !!

Pas de bases de données !!

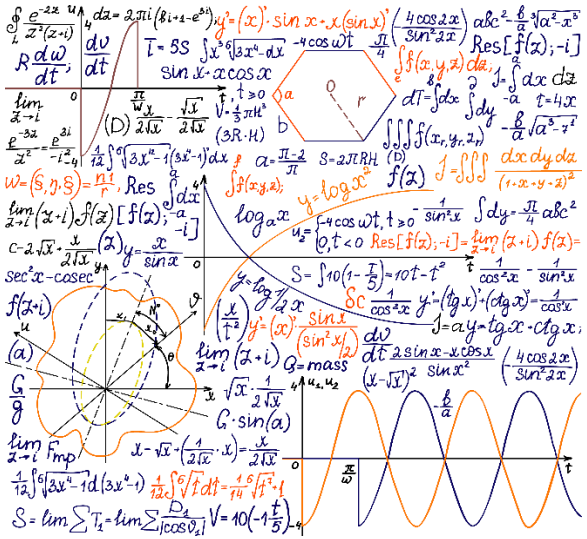
Pas de …. !!

Principe de base

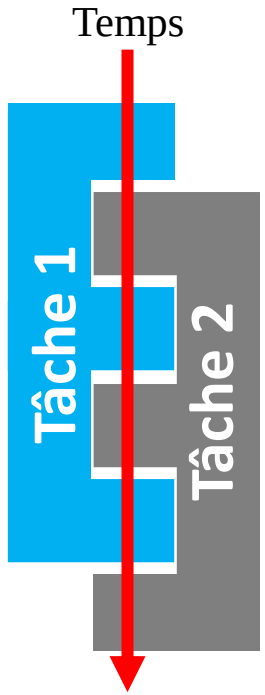
Quel est l'intérêt d'un programme qui ne fait rien que **retourner des valeurs ?**

Principe de base

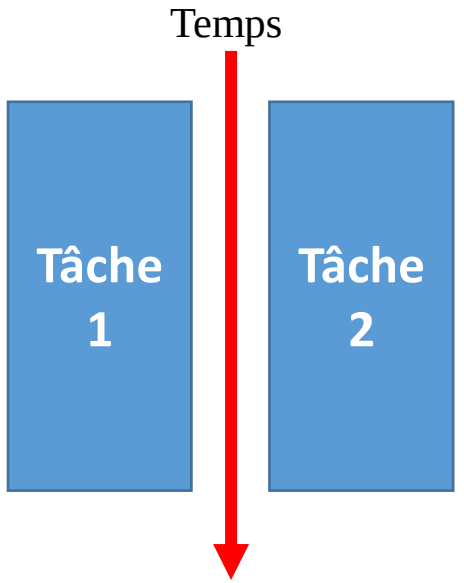
Des situations où la paradigme fonctionnel est plus adapté qu'un autre paradigme de programmation …



Le calcul des formules complexes



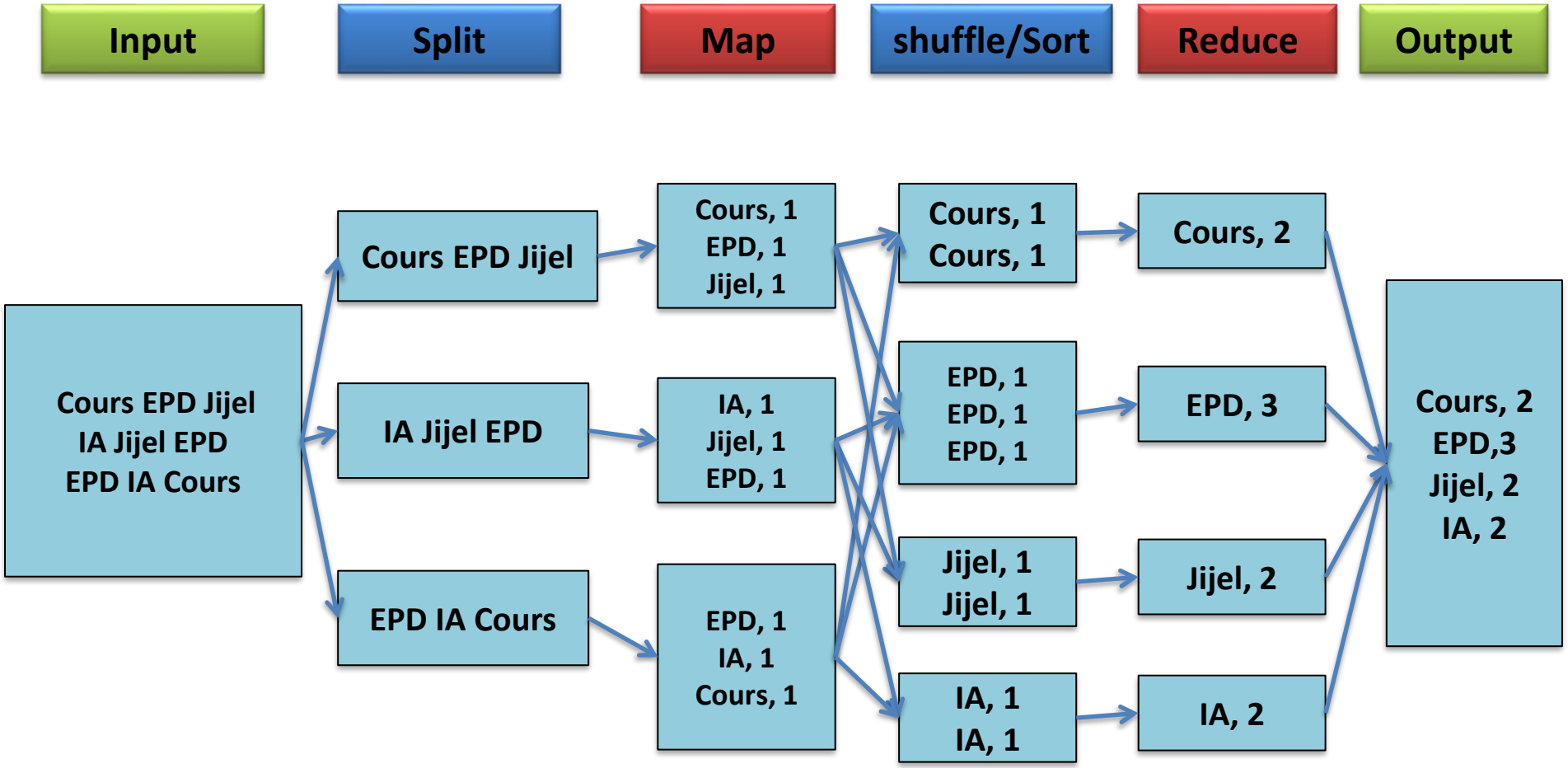
La gestion des threads (synchronisation)



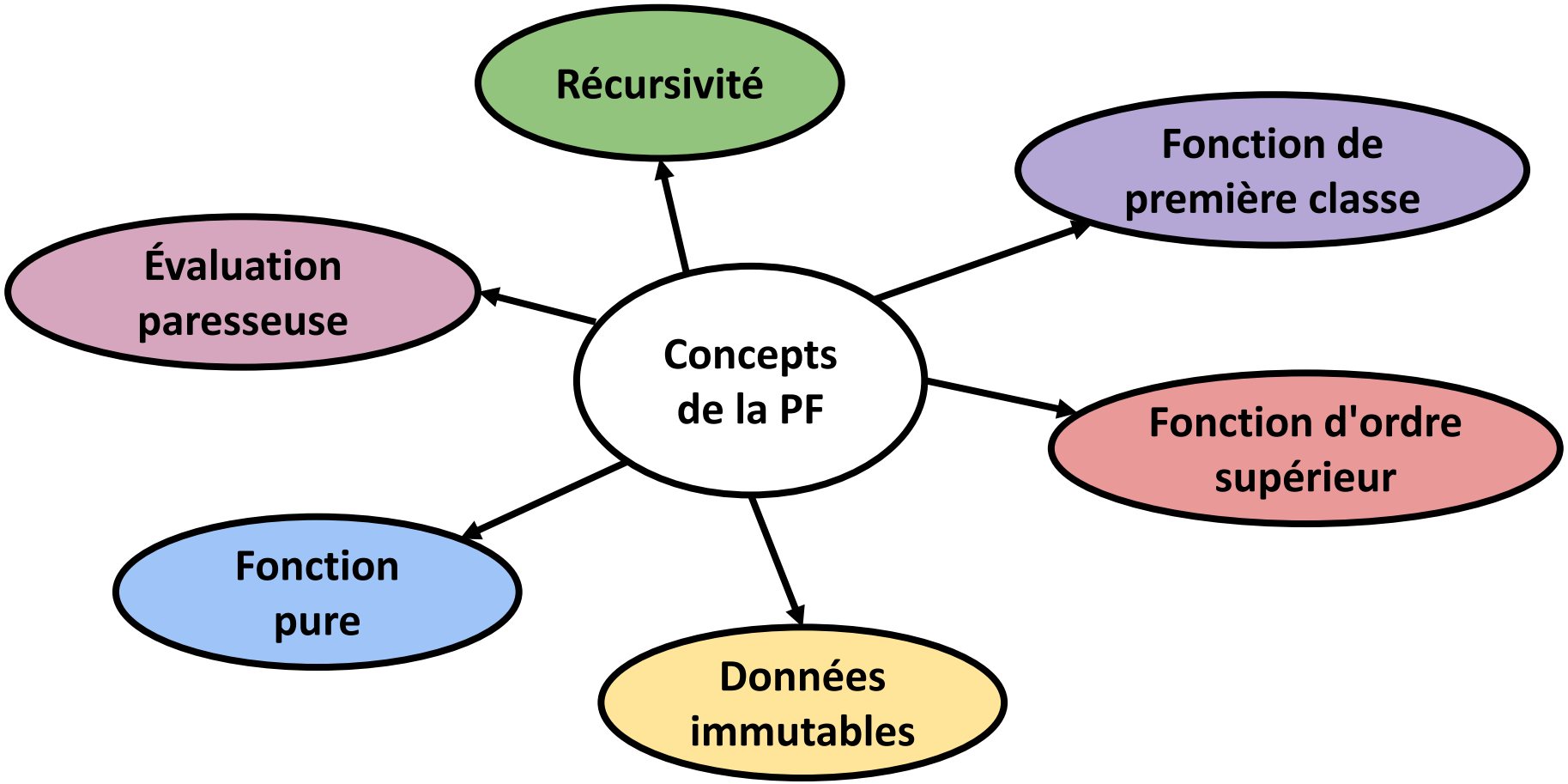
Le calcul distribué

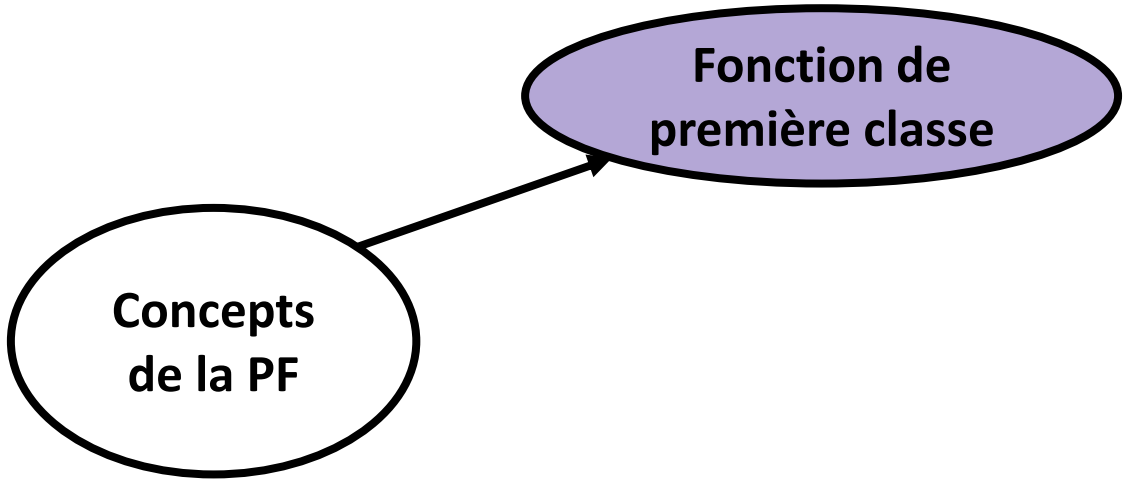
Principe de base

Exemple : traiter en parallèle une quantité importante de données dans un contexte « Big data » …



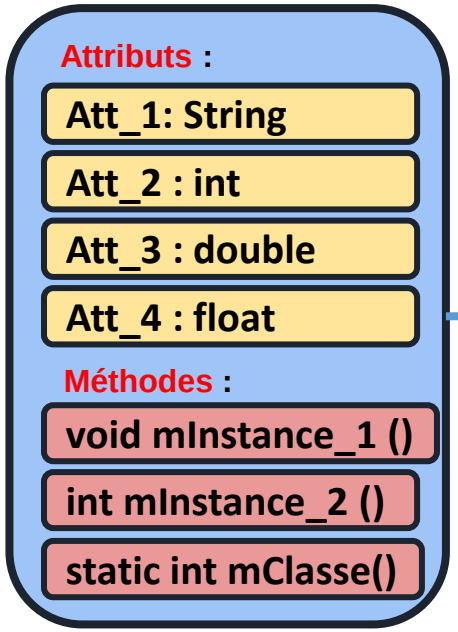
Concepts de la programmation fonctionnelle




Fonction de
première classe
Concepts
de la PF

Fonction de première classe

La classe CC



Le concept de méthode en POO

- **Méthode d'instance** : elle est **liée à un objet** (elle agit sur un objet).
- **Méthode de classe** : elle est indépendante des objets, mais elle reste néanmoins **liée à la classe**.

Le paradigme fonctionnel nous offre la possibilité de se libérer de ce lien (on parle de fonction et non de méthode).

Fonction de première classe

La fonction en programmation fonctionnelle est une valeur à part entière …

Fonction de première classe

La fonction en PF est une valeur à part entière

Fondamental

- Dans le paradigme fonctionnel, les fonctions sont des valeurs de **première classe** : elles sont considérées comme n'importe quel autre type de variable .

Liste de types : boolean, string, float, int , … , function

Fonction de première classe

La fonction en PF est une valeur à part entière

Fondamental

- Dans le paradigme fonctionnel, les fonctions sont des valeurs de **première classe** : elles sont considérées comme n'importe quel autre type de variable .

- Une **fonction** peut être **affectée en tant que valeur** à une variable, **transmise en tant que paramètre** à d'autres fonctions ou **retournée comme résultat** par une autre fonction.

Fonction de première classe

Exemple : JavaScript traite les fonctions comme des types de première classe.

Une **fonction** peut être **affectée en tant que valeur**

```
function f(x){
    return x*3;
}
console.log(f(2));
```

```
6
```

Variable

```
var tripleVar = function (x){
    return x*3;
}
console.log(tripleVar(2));
```

Une fonction anonyme

Fonction de première classe

Exemple : JavaScript traite les fonctions comme des types de première classe.

```
listOfElement = [1,2,3];
function printElement(element){
        console.log(element);
}
function actionToList(list,myFunction){
        list.forEach(myFunction);
}
actionToList(listOfElement,printElement);
```

Une fonction est transmise en tant que paramètre.

```
1
2
3
```

Fonction de première classe

Exemple : JavaScript traite les fonctions comme des types de première classe.

```
function random() {
    return Math.random();
}
function magic() {
 return random();
}
console.log(magic());
```

Une fonction est retournée comme résultat d'une autre fonction

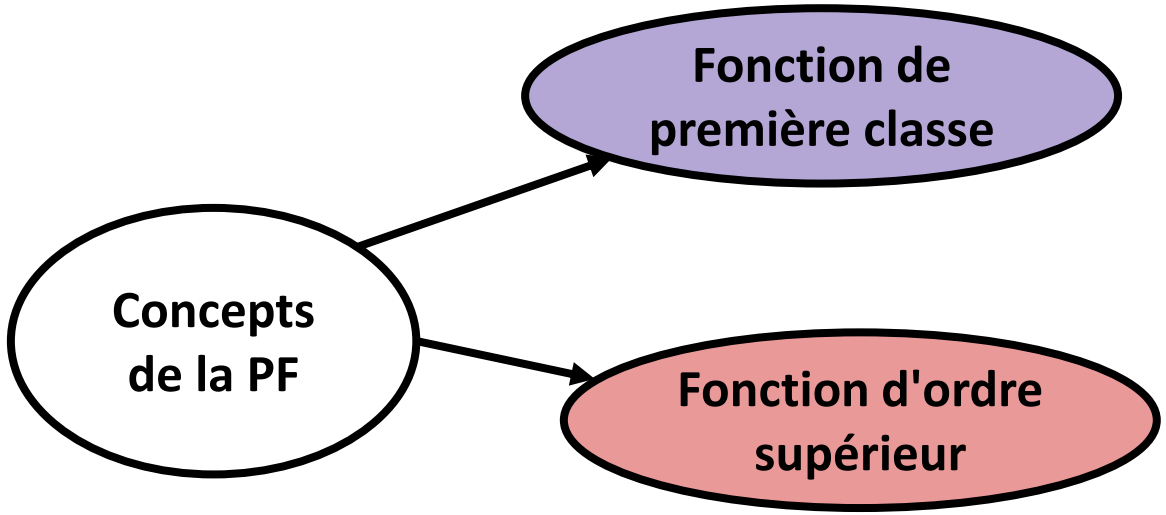
```
0.4487496844293595
```

Fonction de première classe

Quel est l'intérêt par rapport à une méthode ?

Fonction de première classe

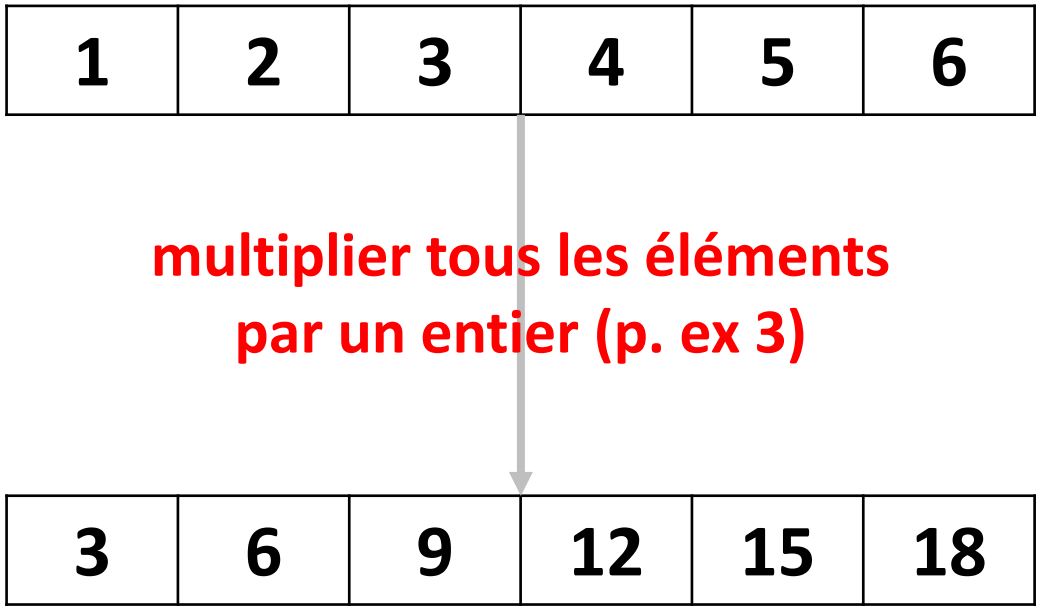
Une nouvelle étendue de possibilités : des fonctions d'ordre supérieur …


Concepts
de la PF
Fonction de
première classe
Fonction d'ordre
supérieur

Fonction d'ordre supérieur

Imaginez le scénario suivant …

Liste d'entiers

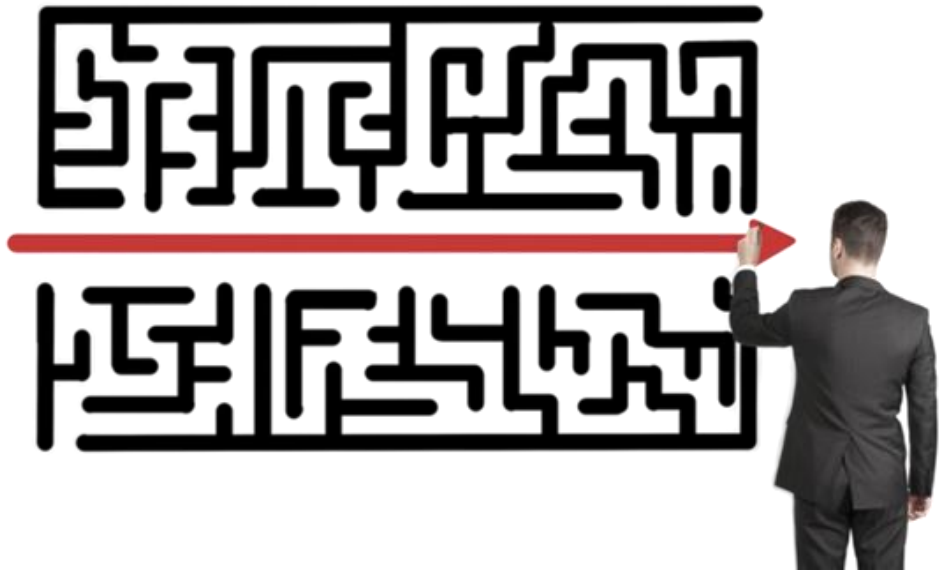



```
//code impératif
var tab = [1,2,3,4,5,6];
var tripleResult = [];
for(var i = 0; i  < tab.length; i++) {
      tripleResult [i] = tab[i] * 3 ;
}
```

Fonction d'ordre supérieur

Imaginez le scénario suivant …

Pourquoi faire compliqué quand on peut faire simple ?



Fonction d'ordre supérieur

Imaginez le scénario suivant …

Pourquoi ne pas simplement appliquer la fonction triple à ma liste ?

```
var tab = [1,2,3,4,5,6];
function triple(elem){
    return elem*3;
}
console.log(triple(2));
```

6

Quelque chose comme ça …

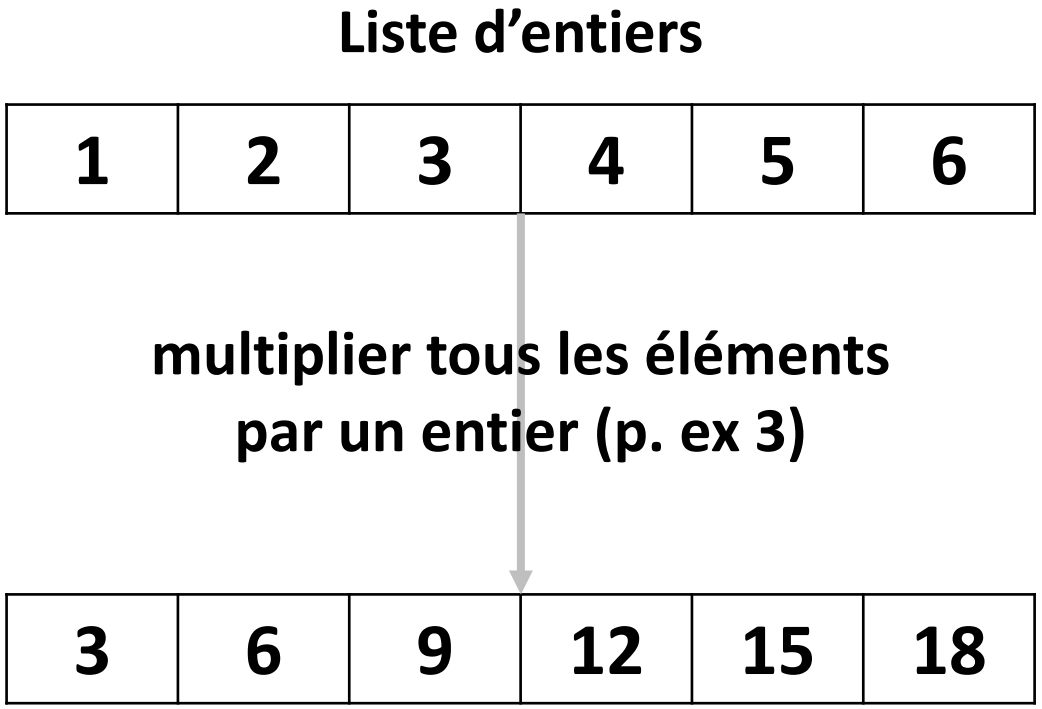
```
function applyToList(list,myFunction){
        list.forEach(myFunction);
}
applyToList(tab, triple);
```

Fonction d'ordre supérieur

Imaginez le scénario suivant …

Ils ont pensé à nous !!!!

Fonction d'ordre supérieur




```
//code impératif
var tab = [1,2,3,4,5,6];
var tripleResult = [];
for(var i = 0; i  < tab.length; i++) {
      tripleResult [i] = tab[i] * 3 ;
}
```

Donc pour faire ceci en programmation fonctionnelle …

Fonction d'ordre supérieur

La fonction d'ordre supérieur map.

map (*3) [1,2,3,4,5,6]

```
var tab = [1,2,3,4,5,6];
function triple(elem){
    return elem*3;
}
var tripleResult = tab.map(triple)
console.log(tripleResult);
```

map applique la fonction triple sur tous les éléments du tableau « tab ».

[3, 6, 9, 12, 15, 18]

Attention

- **Le langage de programmation utilisé est : Javascript.**

Fonction d'ordre supérieur

Des fonctions qui reçoivent ou produisent d'autres fonctions

Rappel

- Dans le paradigme fonctionnel, les fonctions peuvent :
  - (1) prendre des fonctions comme arguments.
  - (2) retourner des fonctions comme résultats.

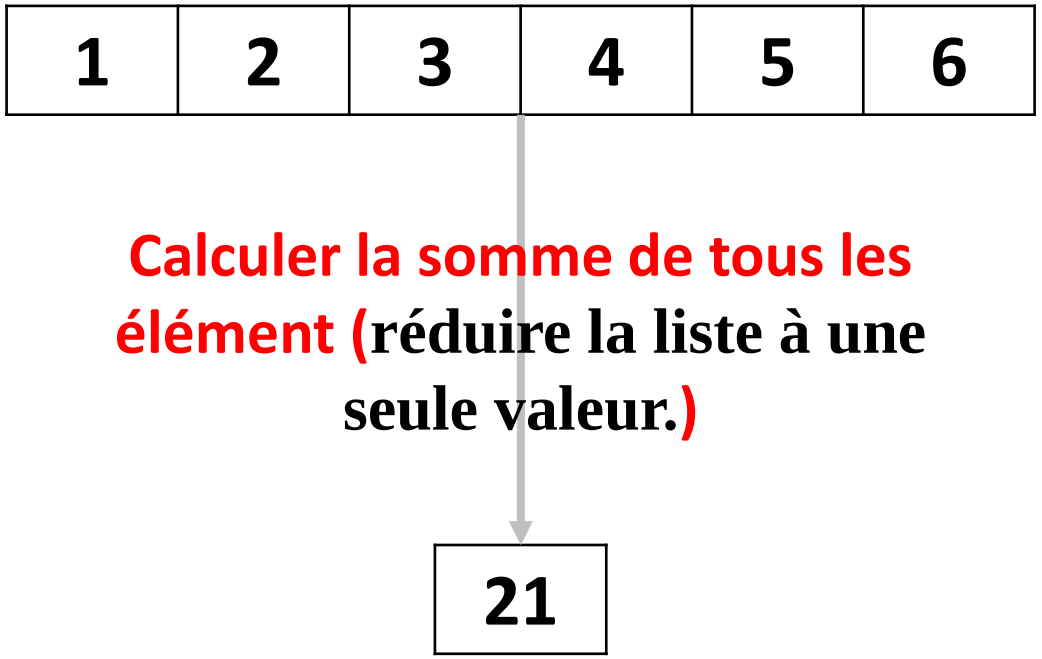
Fondamental

- Une **fonction d'ordre supérieur** est une fonction qui **a au moins une des deux propriétés précédentes**.

Fonction d'ordre supérieur

La fonction d'ordre supérieur reduce.

Liste d'entiers




```
//code impératif
var tab = [1,2,3,4,5,6];
result = 0;
for(var i = 0; i  < tab.length; i++) {
      result = result + tab[i];
}
```

Fonction d'ordre supérieur

La fonction d'ordre supérieur reduce.

reduce (+,0) [1,2,3,4,5,6]

```
var tab = [1,2,3,4,5,6];
function sum(result,elem){
    return result+elem;
}
var sumOfArray = tab.reduce(sum,0)
console.log(sumOfArray);
```

reduce applique la fonction sum sur tous les éléments du tableau « tab » afin de le réduire à une seule valeur.

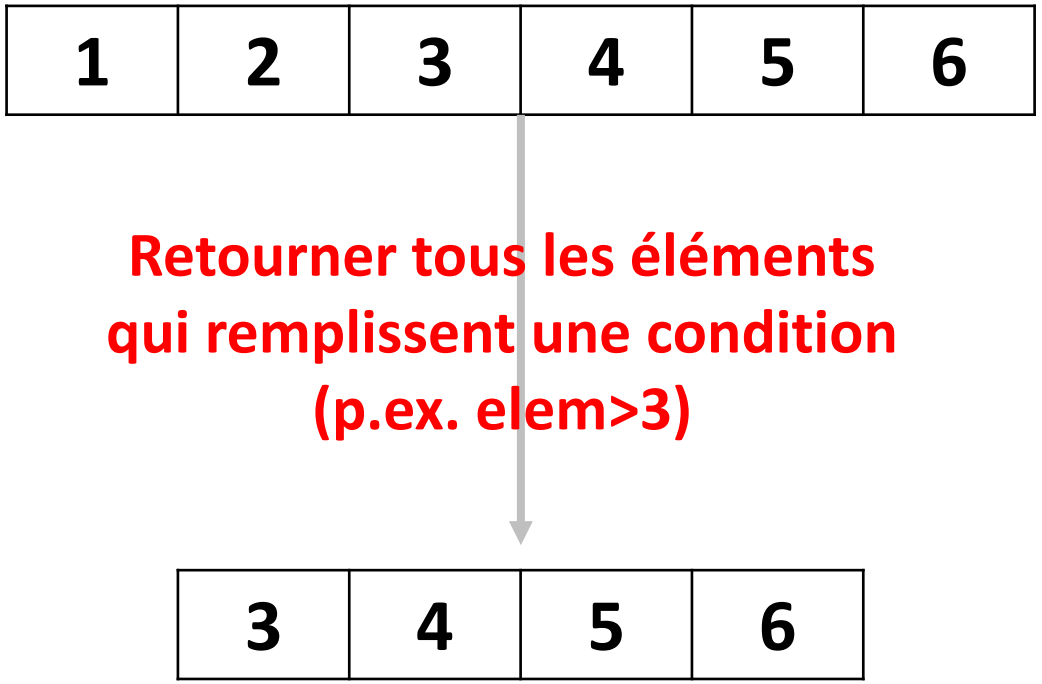
La valeur initiale de l'accumulateur **result**

```
21
```

Fonction d'ordre supérieur

La fonction d'ordre supérieur filter.

Liste d'entiers




```
//code impératif
pos = 0;
var filtredResult = [];
for(var i = 0; i  < tab.length; i++) {
  if (tab[i]>3){
    filtredResult[pos] = tab[i] ;
    pos = pos + 1;
  }
}
```

Fonction d'ordre supérieur

La fonction d'ordre supérieur filter.

filter (>3) [1,2,3,4,5,6]

```
var tab = [1,2,3,4,5,6];
function cond(elem){
    return elem>3;
}
var filtredResult = tab.filter(cond)
console.log(filtredResult);
```

filter applique la fonction cond sur tous les éléments du tableau « tab » pour retourner tous les éléments qui remplissent la condition déterminée par cette fonction

```
[ 4, 5, 6 ]
```

Exercice : Appliquer la notion

Fonction d'ordre supérieur

Exercice : Appliquer la notion

```
const users = [
  { name: 'Ali', age: 25 },
  { name: 'Salim', age: 30 },
  { name: 'Yassine', age: 35 },
];

function getName(user) {
  return user.name;
}
const names = users.map(getName);

console.log(names);
```

```
['Ali', 'Salim', 'Yassine']
```

```
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map(num => num * 2);

console.log(doubled);
```

Arrow functions /
lambda expressions

```
[2, 4, 6, 8, 10]
```

Fonction d'ordre supérieur

Exercice : Appliquer la notion

```
const numbers = [1, -2, 3, -4, 5, -6];

function isNegative(num) {
  return num < 0;
}

const negativeNbrs = numbers.filter(isNegative);

console.log(negativeNbrs);
```

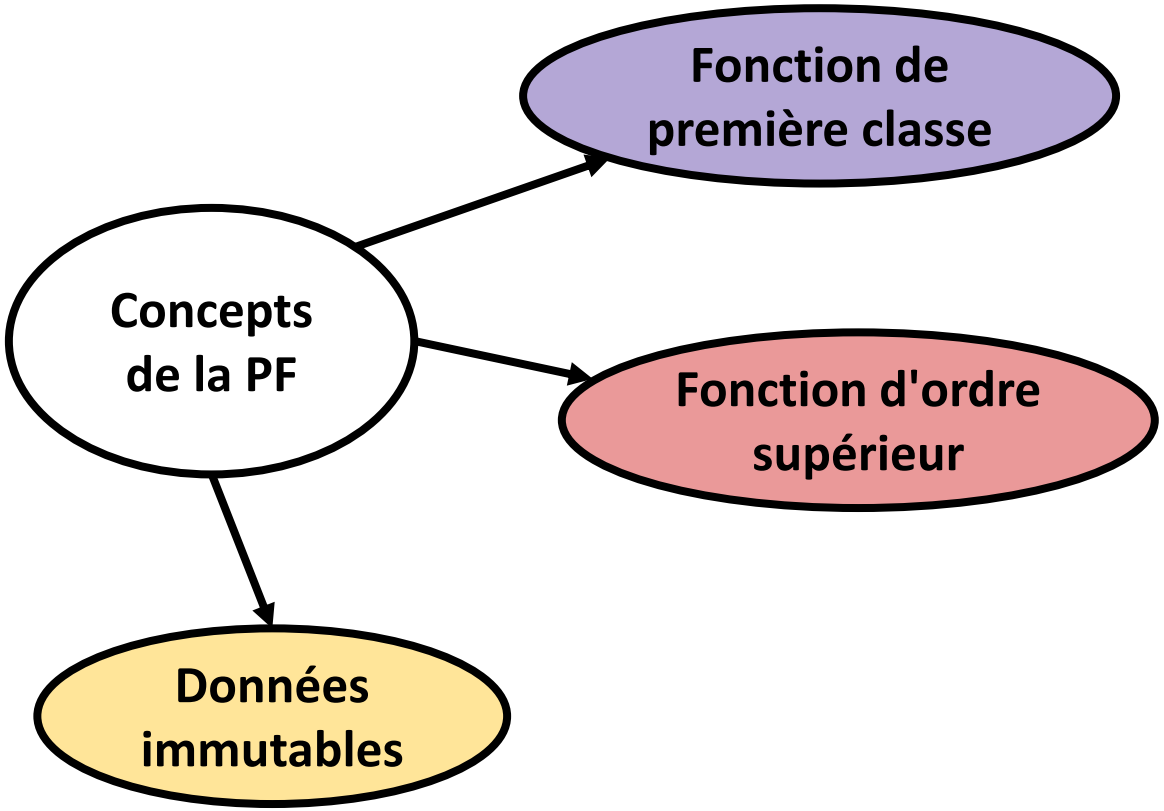
```
const users = [
  { name: 'Ali', age: 25 },
  { name: 'Salim', age: 30 },
  { name: 'Yassine', age: 35 },
];

const olderUsers = users.filter(user => user.age > 30);

console.log(olderUsers);
```

```
[-2, -4, -6]
```

```
[{ name: 'Yassine', age: 35 }]
```


Concepts
de la PF
Fonction de
première classe
Fonction d'ordre
supérieur
Données
immutables

Données immutables

Rappel

- Une variable **immutable** (**immuable**) (***cannot change***) est une variable qui ne peut pas être modifiée une fois la variable initialisée.

Attention

- **Immutability is a distinct notion than that of a constant :** "Constants are immutable in the sense that they cannot change. However, immutability refers to values, not to the assignment of values"**.**

```
String s = "x";
s=s+"y" ;
```

s est immutable

```
StringBuilder s = new StringBuilder "x";
s.append("y") ;
```

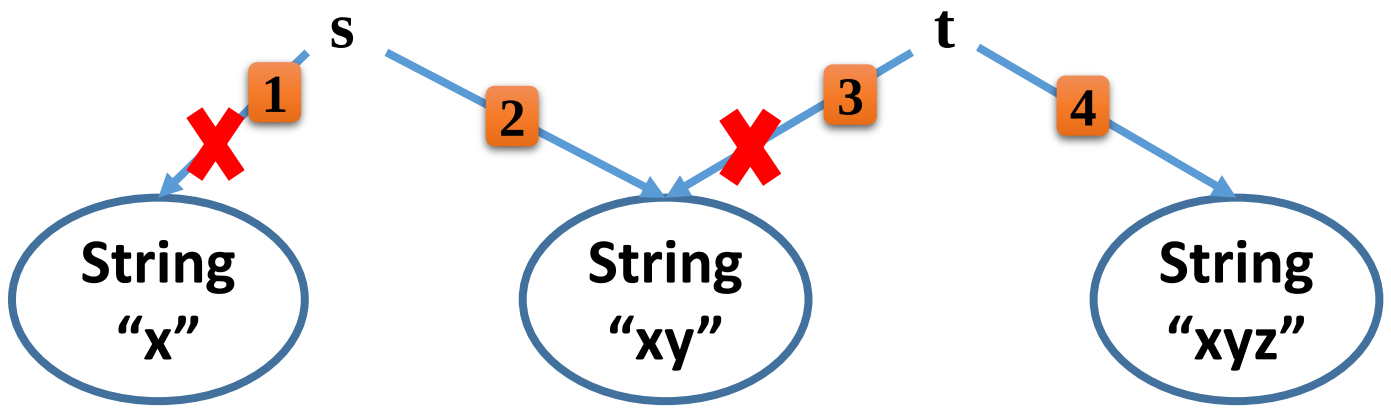
s est mutable

Données immutables

Le type « String » en Java est immutable !

- Les variables de type « **String** » en Java sont immutables, c'est-à-dire que vous ne pouvez pas les modifier. Ainsi, la modification d'une chaine de caractère de ce type (p. ex en utilisant l'opérateur de concaténation ou une méthode comme toUpperCase) génère une nouvelle chaîne de caractère, au lieu de modifier le contenu de la chaîne existante.

```
String s = "x"; //1
s=s+"y" ; //2
String t = s; //3
t=t+"z" ; //4
```

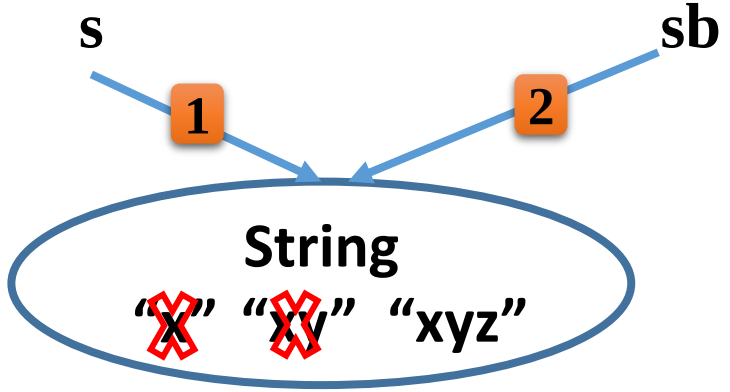


Données immutables

Le type « StringBuilder » en Java est mutable !

- Les variables de type « **StringBuilder** » en Java sont mutables, c'est-à-dire que vous pouvez les modifier sans aucun problème. Ainsi, la modification d'une chaine de caractère de ce type modifie le contenu de la chaîne existante.

```
StringBuilder s = new StringBuilder "x"; //1
s.append("y") ;
StringBuilder sb = s ; //2
sb.append("z") ;
```



Données immutables

Les types mutables semblent beaucoup plus puissants que les types immutables ...

Données immutables

Imaginez le scénario suivant …

```
String s = "";
for (int i = 0; i < n; ++i) {
    s = s + n;
}
```

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < n; ++i) {
  sb.append(String.valueOf(i));
}
```

Attention

- Gros défaut des données immutables : pour la modification des données immutables vous êtes obligé de créer d'autres copies de ces données = **Problème de performance (l'utilisation élevée de la mémoire)**.

Données immutables

Les types mutables semblent beaucoup plus puissants que les types immutables, mais les types immuables sont plus sûrs et plus faciles à comprendre.

Données immutables

Rappel

- Une chaîne de caractères littérale : est une séquence de caractères placée entre guillemets (" ") (p. ex. "Xyz").

- Pour des raisons d'optimisation en Java (gain d'espace et réduction de l'utilisation de la mémoire), **les références vers des chaînes littérales** identiques pointent vers un même String. P. ex. ici, "IA" et "IA" sont deux chaînes littérales identiques, s1 et s2 pointent donc vers un même String.

```
String s1 = "IA";
String s2 = "Java";
String s3 = "IA";
System.out.println(s1==s3);
```

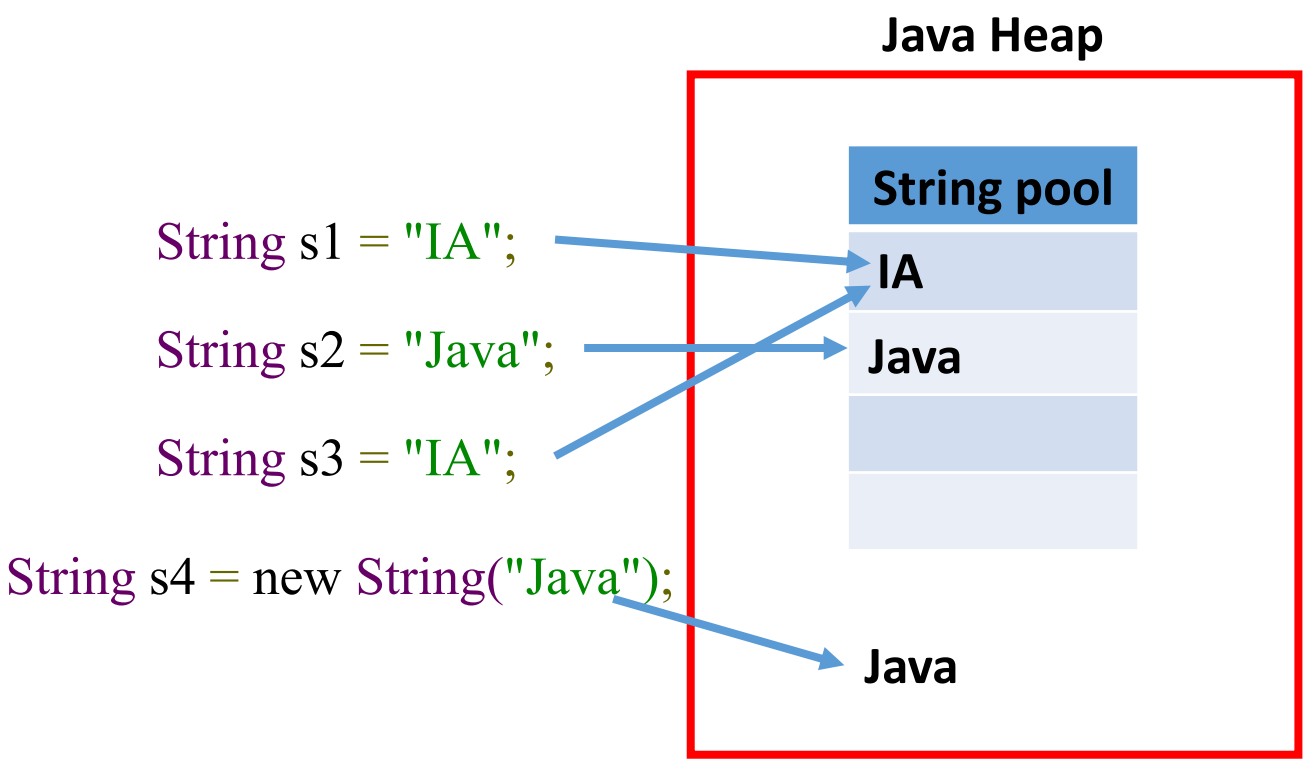
True

Données immutables

La classe String maintient un pool pour les chaînes de caractères littérales.

Rappel

- Pour la création d'une chaîne de caractères littérale, le système recherche une chaîne ayant la même valeur dans le pool.
  - s'il la trouve, il renvoie simplement la référence
  - sinon il crée une nouvelle chaîne dans le pool et renvoie sa référence.

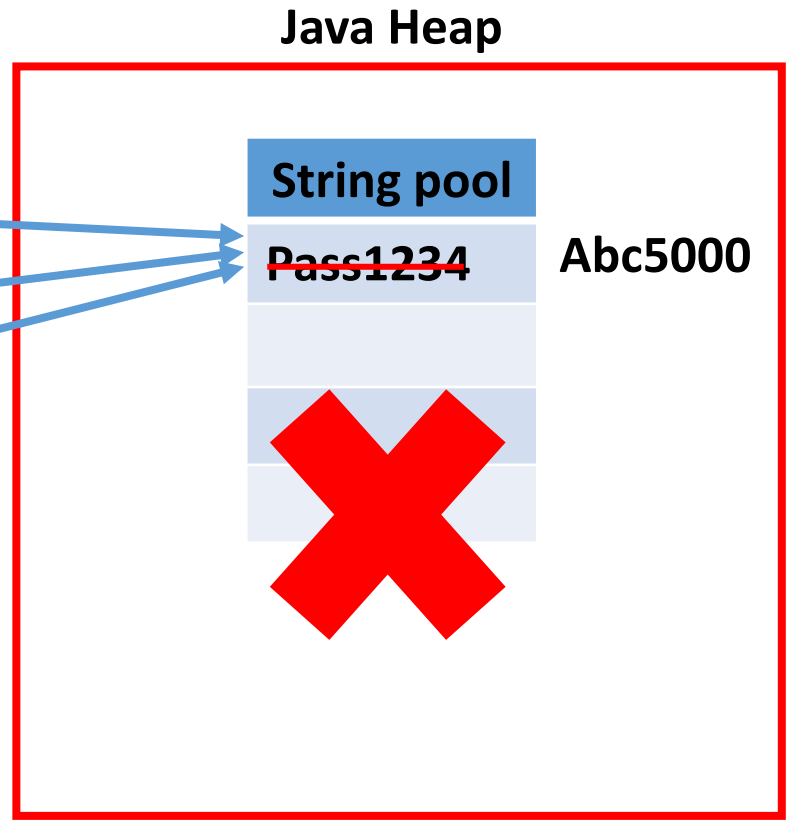


Données immutables

Imaginez le scénario suivant : la gestion des mots de passe …

```
String passUser1 = "Pass1234";
String passUser2 = "Pass1234";
String passUser3 = "Pass1234";

passUser1 = "Abc5000";
```



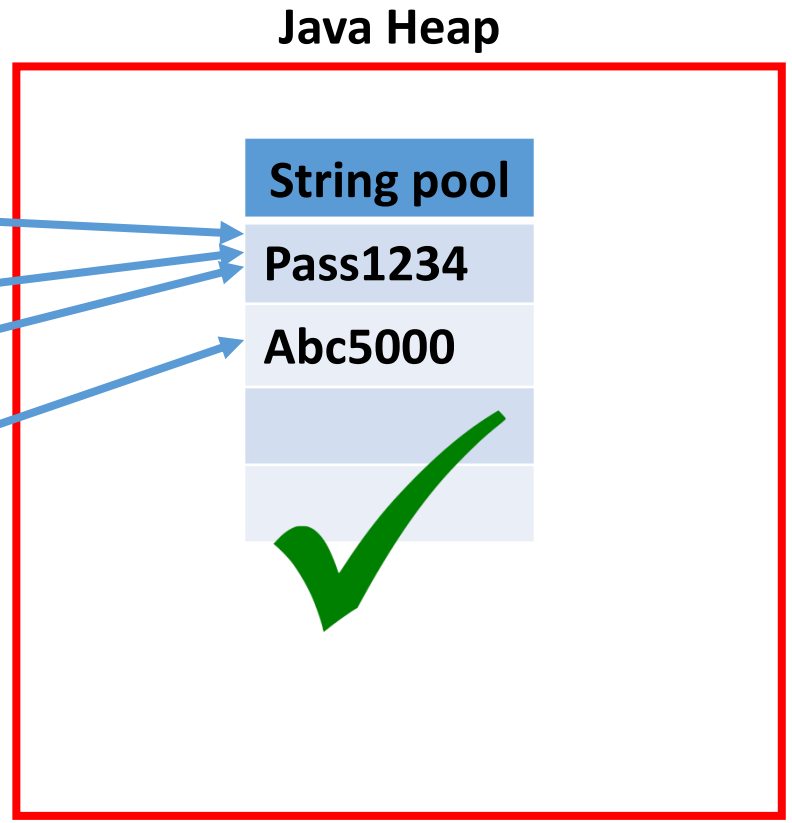
Si l'utilisateur « User 1 » veut changer son mot de passe, que se passe-t-il si « String » était un type mutable ?

Données immutables

Imaginez le scénario suivant : la gestion des mots de passe …

```
String passUser1 = "Pass1234";
String passUser2 = "Pass1234";
String passUser3 = "Pass1234";

passUser1 = "Abc5000";
```



Heureusement « String » est immutable !

Données immutables

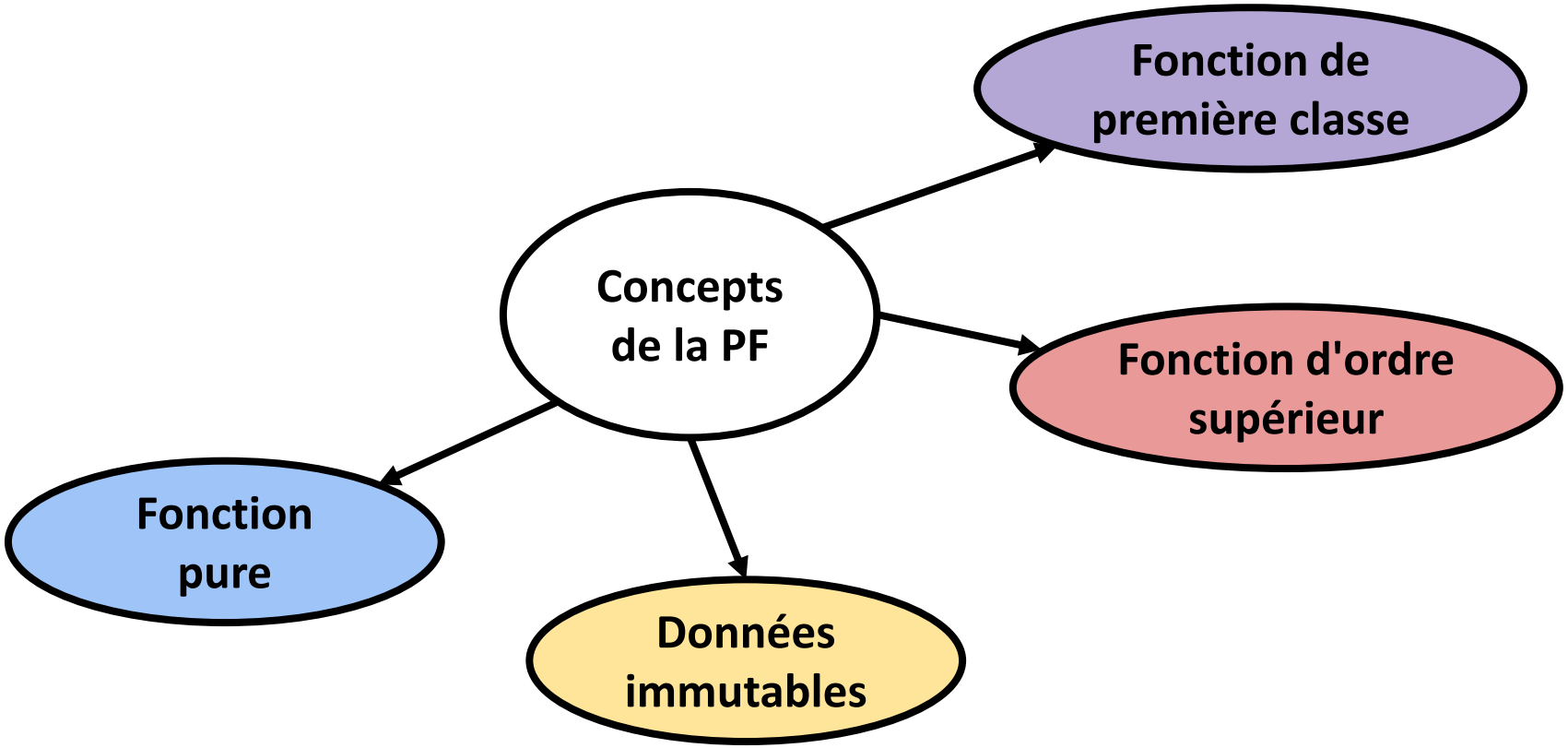
Immutable types are safer, and easier to understand

- Les variables de type « String » sont largement utilisées comme paramètres pour de nombreuses classes Java (par ex. le port et l'adresse de l'hôte pour ouvrir une connexion réseau, le chemin des fichiers à lire, l'identifiant et le mot de passe pour établir la connexion avec une base de données, ….)

Attention

- Simplifier la gestion d'un état partagé (des variables thread-safe) : un type « String » mutable pourrait bien causer de sérieuses **menaces de sécurité** à nos applications.

Programmation Fonctionnelle



Fonction pure

Vers la transparence référentielle : adieu les effets de bord !

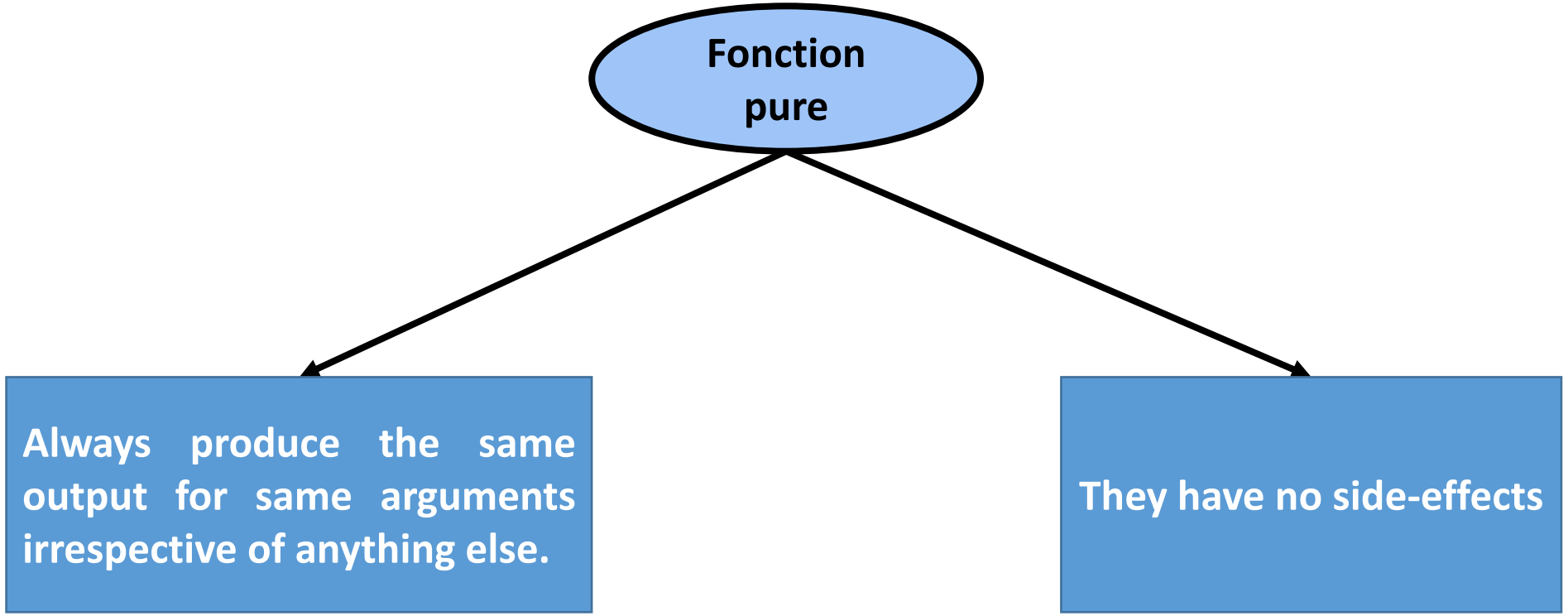
Fondamental

- Les **fonctions pures** sont inspirées des fonctions mathématiques. Ainsi, elles possèdent deux propriétés:
  - (1) **Aucune mutation** : elle retourne la même valeur pour les mêmes arguments (**transparence référentielle**).
  - (2) Aucune influence extérieure : elles n'engendre **pas d'effet de bord** (Side-effect free).

Rappel

- **L'effet de bord « side effect »** : on parle d'un effet de bord lorsque l'exécution d'un sous-programme cause la modification d'un état (variable) en dehors de son environnement local.

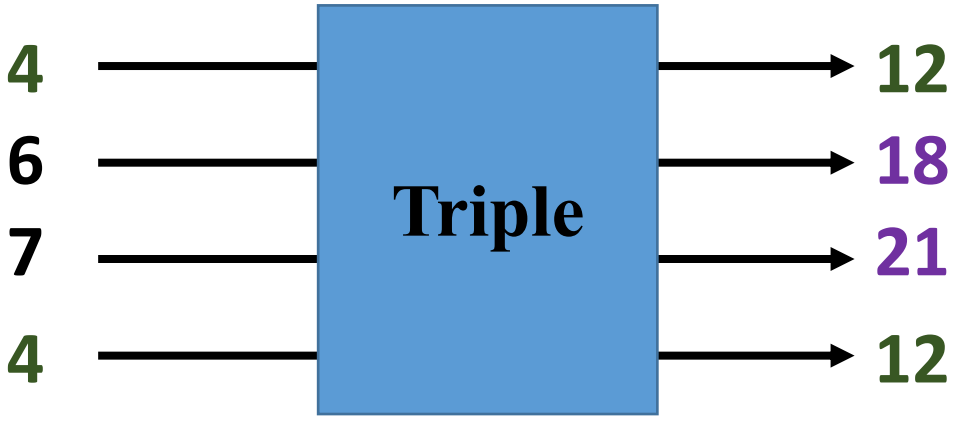
Fonction pure



Fonction pure

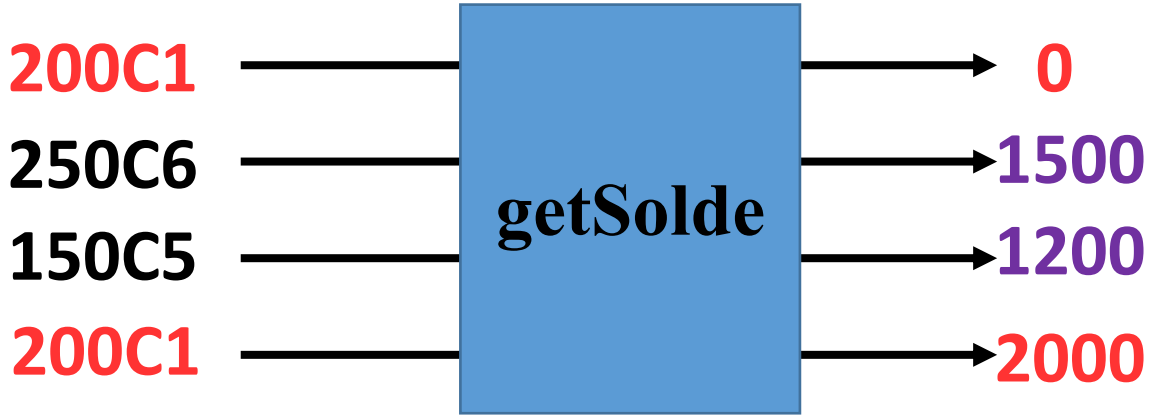
```
int triple(int elem){
    return elem*3;
}
```

Résultats prédictibles : ils seront toujours les mêmes quoi qu'il arrive.




```
int getSolde(String compte){
    //requête SQL
}
```

Résultats imprédictibles : ils viennent d'une base de données qui peut changer d'états.



Fonction pure

- Pure function are predictable, clean, transparent and safe :
  - Easier to read.
  - Easier to refactor.
  - Easier to debug.
  - Easier to test.
  - Do not depend on anything else, so you don't have to care about time or the order of execution.
  - Eencourage safe ways of programming (Thread-safe).

Fonction pure

“Keep It Simple and Stupid”
is an important design principle in computer science.

Pure functions are simple and stupid in the best possible way.

Wherever possible, you should use pure functions in your applications !

Exercice : Appliquer la notion

Fonction pure

Exercice : Appliquer la notion

```
public int factorial(int a){
int i,f;
f=1;
for(i=1;i<=a;i++)
 f=f*i;
return f;
}
```

Pure Impure

```
var init = 0;
function add(a,b){
  init =1;
  return a + b + init
}
```

Pure Impure

Fonction pure

Exercice : Appliquer la notion

Solution

Fonction pure

Exercice : Appliquer la notion

```
public int factorial(int a){
int i;
f=1;
for(i=1;i<=a;i++)
 f=f*i;
return f;
}
```

Pure

Impure

```
var init = 0;
function add(a,b){
  init =1;
  return a + b + init
}
```

Pure

Impure

Fonction pure

Exercice : Appliquer la notion

```
public void factorial(int a){
int i;
f=1;
for(i=1;i<=a;i++)
 f=f*i;
System.out.println(f);
}
```

Pure Impure

```
function magic() {
    return Math.random();
}
```

Pure Impure

Fonction pure

Exercice : Appliquer la notion

Solution

Fonction pure

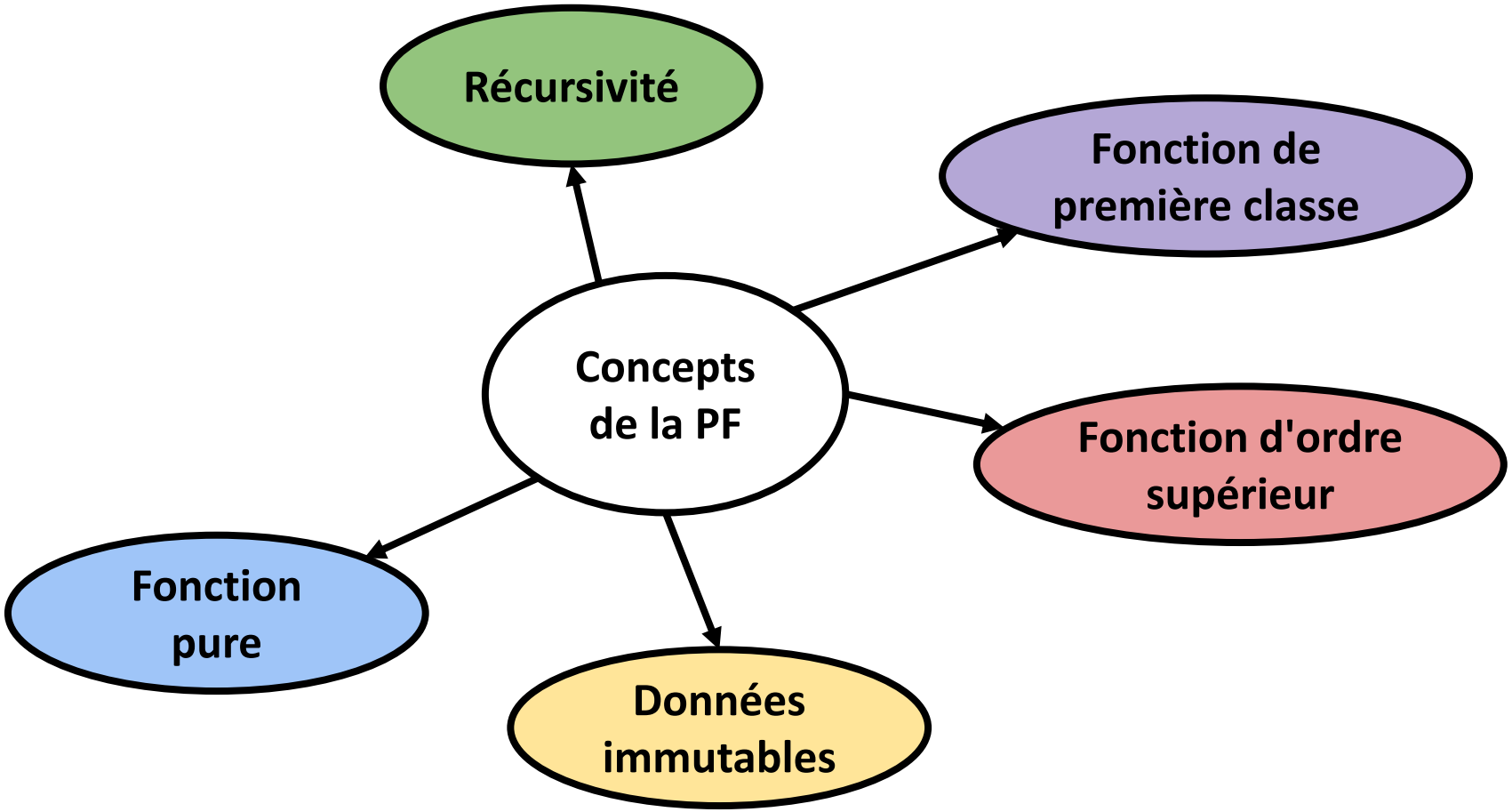
Exercice : Appliquer la notion

```
public void factorial(int a){
int i;
f=1;
for(i=1;i<=a;i++)
 f=f*i;
System.out.println(f);
}
```

Pure Impure ✓

```
function magic() {
    return Math.random();
}
```

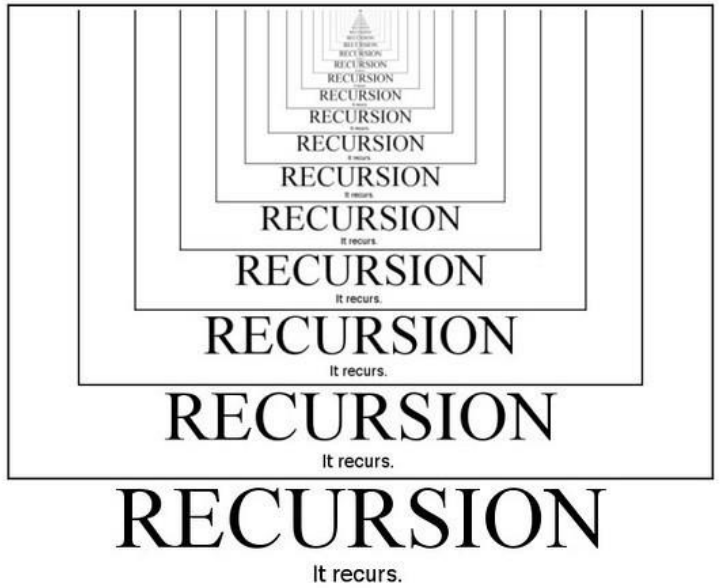
Pure Impure ✓


Récursivité
Fonction de première classe
Concepts de la PF
Fonction d'ordre supérieur
Fonction pure
Données immutables

Récursivité

Rappel

- Un programme est appelé **récursif** lorsque une entité de ce programme (p. ex une fonction) **s'appelle elle-même**.



- La programmation récursive permet d'implémenter des fonctions définies à partir de relations de récurrence (P. ex. la factorielle et les suites).

Suite de Fibonacci

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_n = U_{n-1} + U_{n-2} \text{ Si } n \geq 2 \end{cases}$$

Récursivité

Fondamental

La récursivité permet de résoudre un problème complexe en le ramenant à une succession de problèmes plus simples. Pour ce faire, il nous faut définir 2 choses :

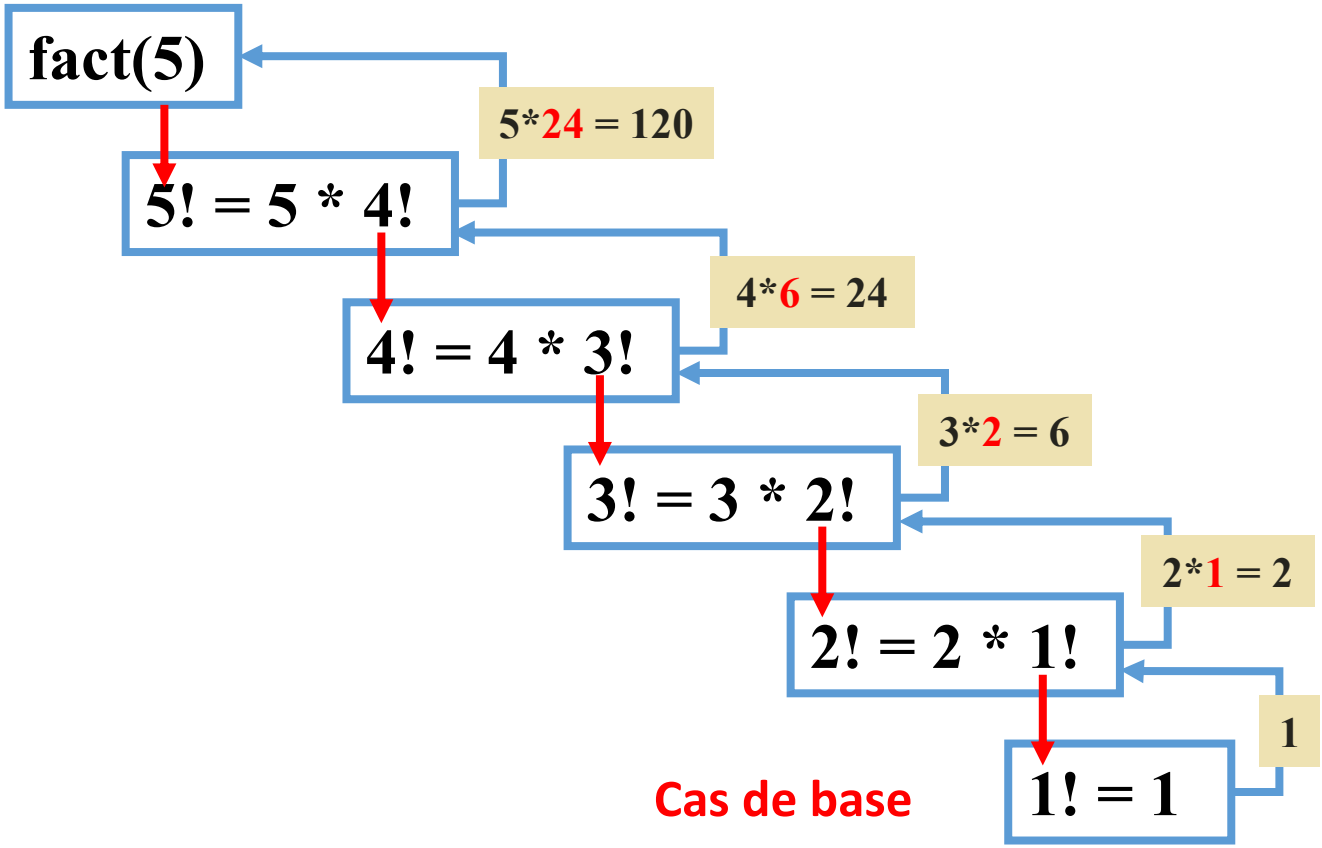
- (1) **une condition d'arrêt;**
- (2) **le comportement récursif.**

La condition d'arrêt

Le comportement récursif

```
function fact(n) {
    if (n === 1)
      return 1;
    else
      return n * fact(n-1);
}
```

Récursivité



Cas de base

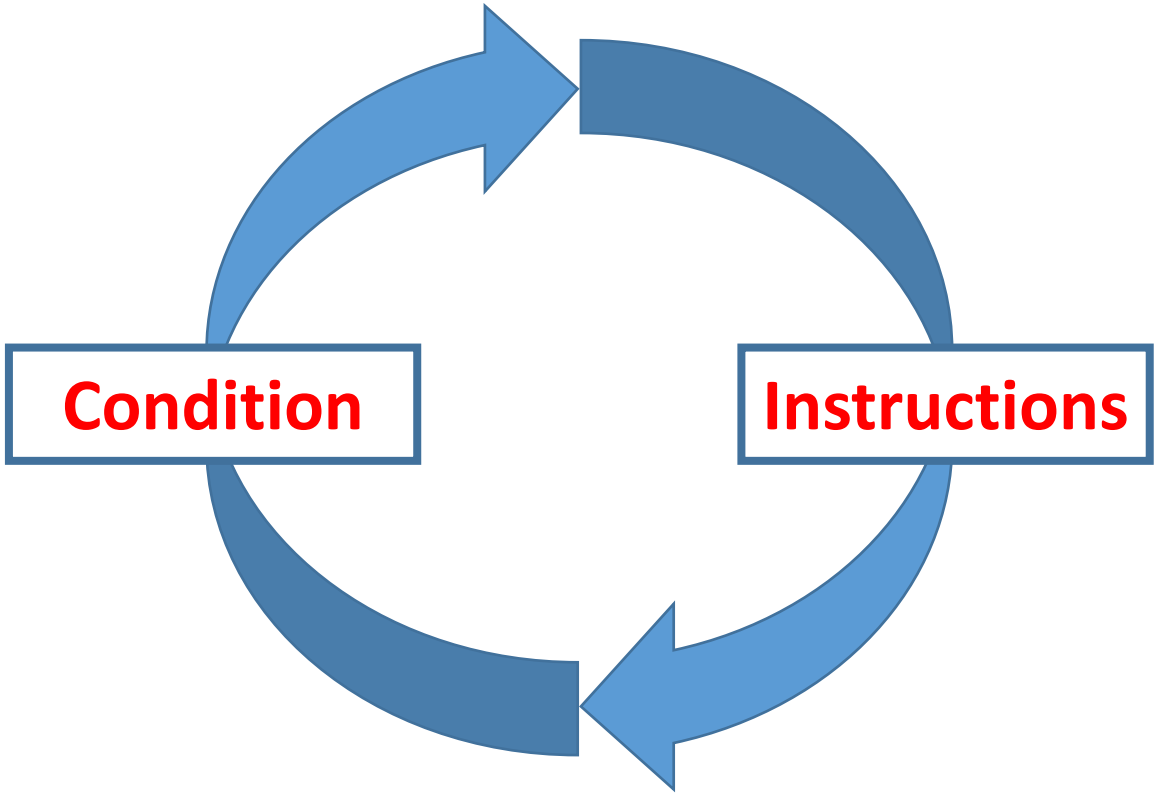
fact(5) = 5 * (fact(4))
fact(5) = 5 * (4 * fact(3))
fact(5) = 5 * (4 * (3 * fact(2)))
fact(5) = 5 * (4 * (3 * (2 * fact(1))))

Le calcul de la fonction factoriel débute lorsque la fonction rencontre son **cas de base.**

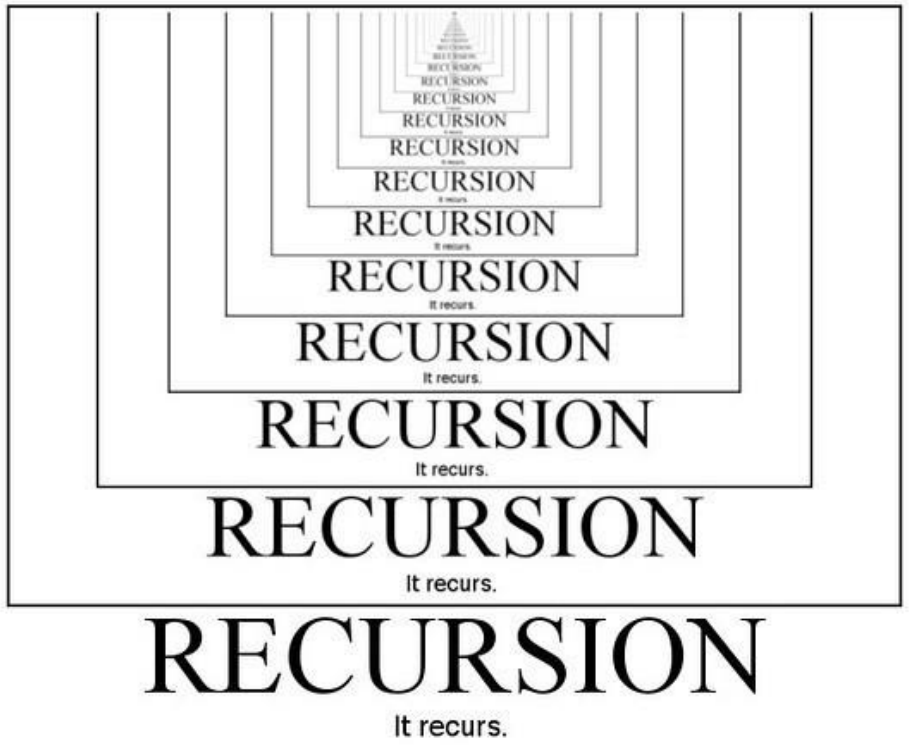
fact(5) = 5 * (4 * (3 * (2 * (1))))
fact(5) = 5 * (4 * (3 * (2)))
fact(5) = 5 * (4 * (6))
fact(5) = 5 * (24)
fact(5) = 120

Récursivité

Impératif vs itératif vs récursif vs fonctionnel



Boucle (itératif)



Récursif

Récursivité

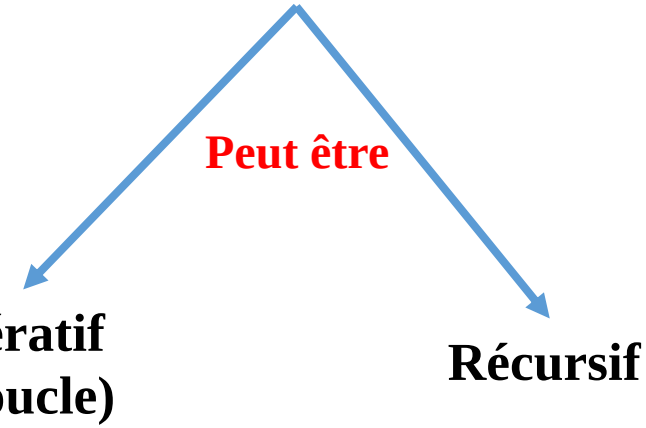
Impératif vs itératif vs récursif vs fonctionnel

Students often mistakenly associate iterative with imperative and recursive with functional.

Récursivité

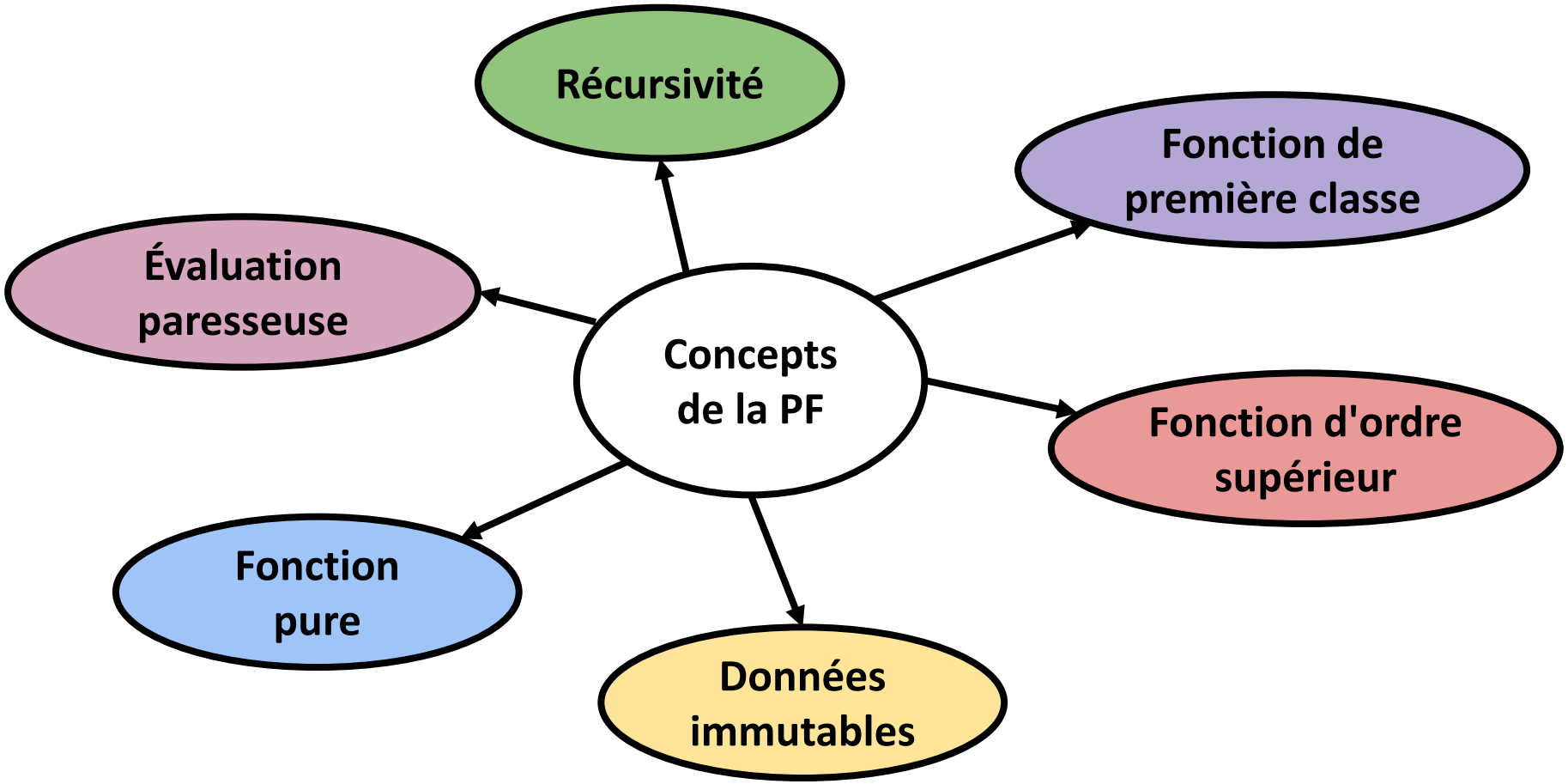
Impératif vs itératif vs récursif vs fonctionnel

Un programme impératif



Un programme purement fonctionnel

La plupart des langages de programmation purement fonctionnels ne comportent pas de boucles, car toutes les données sont immuables (la valeur de la condition d'une boucle ne changera jamais)


Récursivité
Fonction de première classe
Évaluation paresseuse
Concepts de la PF
Fonction d'ordre supérieur
Fonction pure
Données immutables

Évaluation paresseuse

Rappel

- Evaluation stricte (**immédiate**) : est une technique d'implémentation des langages de programmation qui consiste à exécuter les instructions d'un programme dans un ordre séquentiel jusqu'au point de sortie.

```
function fact(n) {
  if (n === 0 || n === 1)
    return 1;
  for (var i = n - 1; i >= 1; i--) {
    n *= i;
  }
  return n;
}
```

- Style impératif (itératif) : une séquence structurée d'instructions modifiant l'état du programme.
- Evaluation stricte : les instructions sont exécutées dans un ordre séquentiel jusqu'au point de sortie.

Évaluation paresseuse

Fondamental

- L'évaluation paresseuse **(évaluation retardée/lazy evaluation)** : est une technique d'implémentation des langages de programmation qui permet de n'exécuter une partie d'un code que lorsque le résultat de cette partie est devenue réellement nécessaire.

- Style fonctionnel : ensemble de fonctions que l'on peut imbriquer les unes dans les autres.
- Evaluation paresseuse : reporter ce que tu n'as pas besoin de faire maintenant à plus tard ou à jamais.

Évaluation paresseuse

```
function recommandation(day,temperature){
if (day == "Friday" && temperature > 20)
console.log("Aller se promener");
}
recommandation("Monday",26);
```

L'expression de droite n'est évaluée que si elle est nécessaire

Vu que le jour est différent de « Friday » le résultat du deuxième test (temperature > 20) importe peu (cette partie ne sera jamais exécutée).

Évaluation paresseuse

Lazy evaluation : la procrastination peut être une bonne idée

Fondamental

- L'évaluation paresseuse permet d'optimiser l'exécution en suivant une approche originale impossible à envisager avec l'évaluation strict. (p. ex. la définition des structures infinies : en adoptant une évaluation strict, le programme tenterait d'évaluer indéfiniment les termes de la suite et ne terminerait jamais).

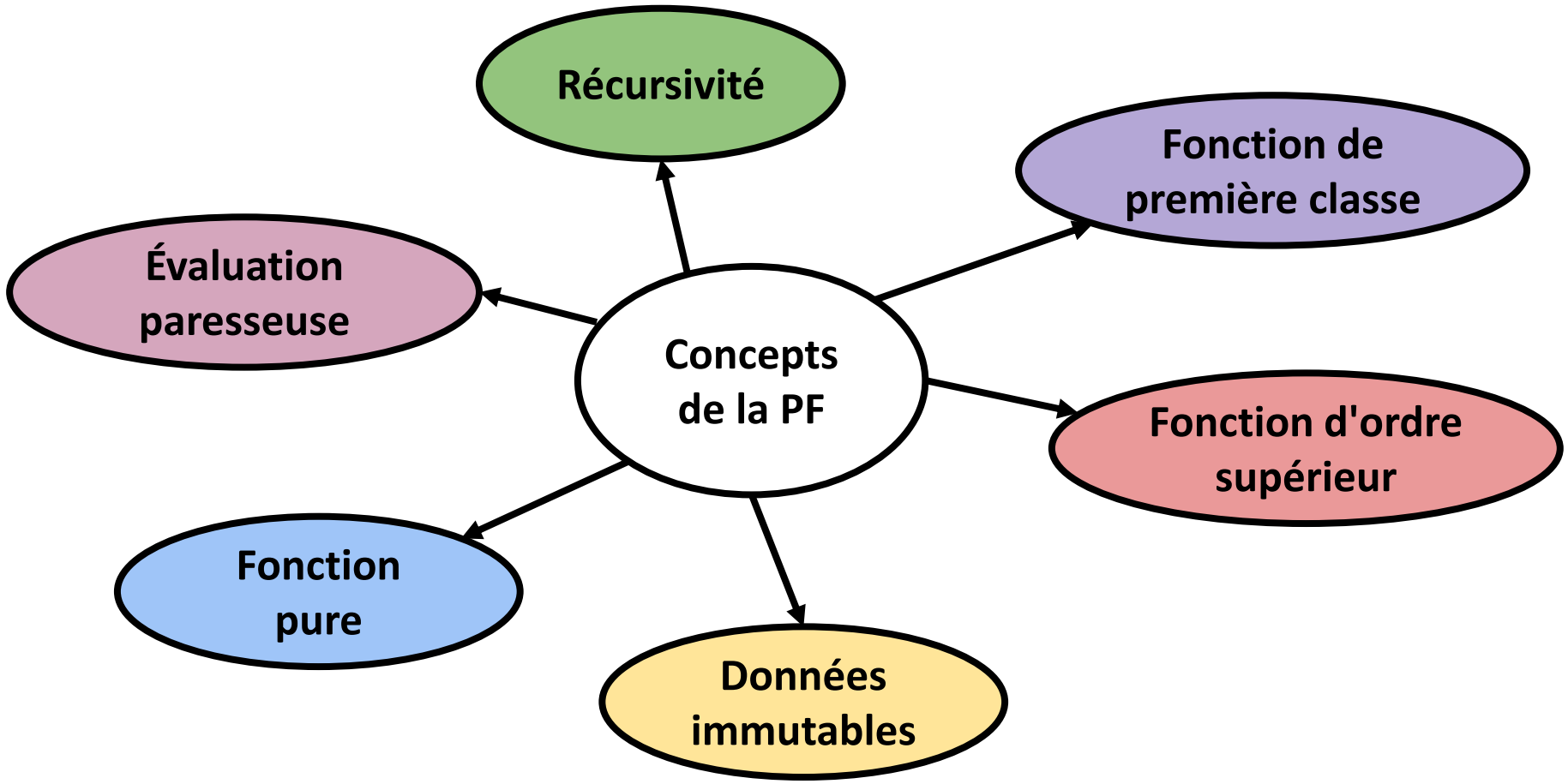
fact(5) = 120

fact(5) = 5 * (fact(4))
fact(5) = 5 * (4 * fact(3))
fact(5) = 5 * (4 * (3 * fact(2)))
fact(5) = 5 * (4 * (3 * (2 * fact(1))))

Cas de base

fact(5) = 5 * (4 * (3 * (2 * **(1)**)))
fact(5) = 5 * (4 * (3 * **(2)**))
fact(5) = 5 * (4 * **(6)**)
fact(5) = 5 * (**24**)

Seuls les termes nécessaires sont calculé.


Récursivité
Fonction de première classe
Évaluation paresseuse
Concepts de la PF
Fonction d'ordre supérieur
Fonction pure
Données immutables

Connexion au système mutable

- **Même dans un programme fonctionnel la mutabilité peut être utile dans certains cas** : la programmation fonctionnelle est très intéressante, mais à un moment ou à un autre, un programme aura besoin d'afficher des résultats sur un écran ou de persister un état dans une base de données.

- **Une fois arrivé à ce stade, le travail du paradigme fonctionnel est terminé : toute la partie mutable se fait en dehors de l'architecture fonctionnelle, dans des fonctions impures sur un paradigme procédural ou orienté objets, …**

Avantages et Inconvénients

Avantages

- Généricité, Réutilisation, modularité.
- Meilleure testabilité et fiabilité.
- Adapter à la programmation concurrente.
- Les fonctions pures sont plus faciles à comprendre et à tester car elles ne changent aucun état.

Inconvénients

- Une façon de pensé différente
- Déclaratif (pas de contrôle du comportement)
- Mois lisible.
- L'écriture de fonctions pures est facile, mais leur combinaison avec le reste de l'application (impures) est difficile.

Question fréquente

La programmation fonctionnelle va tuer la programmation orientée objet ?

- Non, car la PF n'est pas meilleure que la POO, c'est juste une manière différente de programmer, avec ses avantages et ses inconvénients.

- Le choix entre POO et fonctionnelle se fait selon le contexte du projet (**Analogie Théorème de CAP dans les BD**).

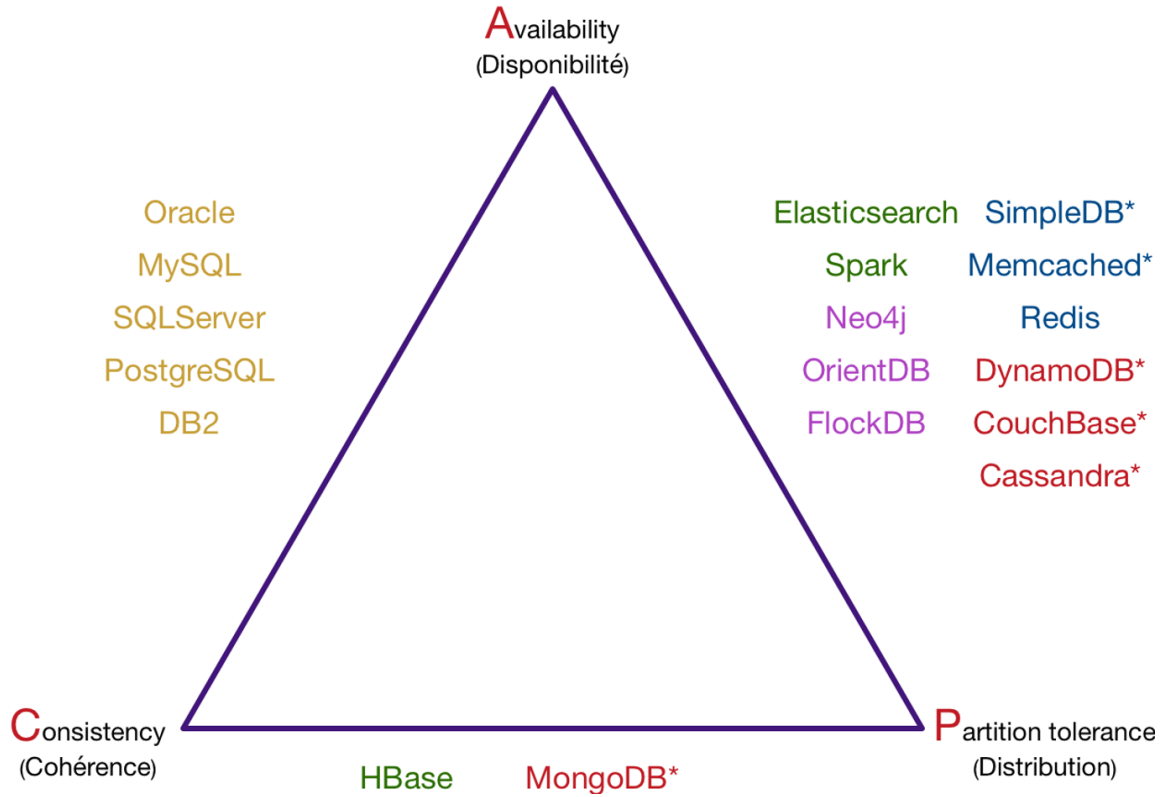
Rappel

- Théorème de CAP (Brewer) : il est impossible dans une base de données de respecter en même temps les trois contraintes suivantes : la cohérence (**C**onsistency), la disponibilité (Availability) et la distribution (Partition).

Modèle

Relationnel
Orienté Clé-Valeur
Orienté Colonne
Orienté Document
Orienté Graphe

* Possibilité de changer le mode de cohérence
Cohérence <-> Disponibilité



Exercice : Appliquer la notion

Exercice : Appliquer la notion

Soit le programme javascript suivant :

```
function sum(x,y){
    return  x + y;
};
const oddNumbers = [1,3,5,7,9];
var s = 5;
function f(x,init){
    return x.reduce(sum,init)
};
console.log(f(oddNumbers,s));
```

- ☐ f est une **fonction de première** classe, car elle ne prend pas une fonction comme paramètre.
- ☐ f est une **fonction impure**, car elle n'engendre pas un effet de bord et elle respecte le principe de la transparence référentielle.
- ☐ f est une **fonction d'ordre supérieur**, car elle est considérée comme n'importe quel autre type de variable.
- ☐ le résultat d'affichage de ce programme est : 25.
- ☐ le résultat d'affichage de ce programme est : [30].
- ☐ Aucune réponse n'est juste.