

INTRODUCTION A LA PROGRAMMATION FONCTIONNELLE POUR LES DÉVELOPPEURS ET DÉVELOPPEUSES OBJET

Effective Labs

- Rendre la conception et le développement logiciel plus qualitatifs et inspirants.
- La mission est d'apporter des solutions et d'assurer la montée en compétences des personnes en travaillant sur 3 thèmes :
 - ▣ Conception & Architecture Logiciel
 - ▣ Culture Lean & Agile
 - ▣ Software Craftsmanship

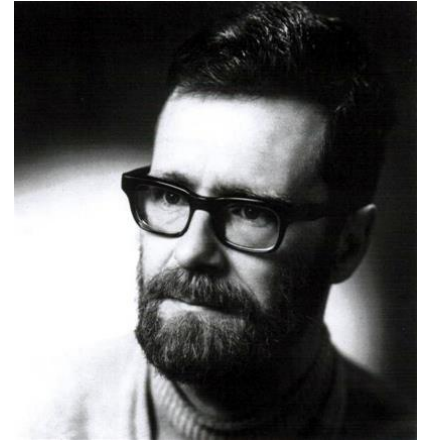
Pourquoi le sujet ?

- **1968** [NR69] – Le plus **gros problème** dans le développement et la maintenance des logiciels est la complexité – les systèmes sont difficiles à comprendre.
 - C'est difficile pour les humains, pas pour le CPU évidemment 😊

- Après 30 ans d'hégémonie de la POO le problème reste d'actualité
 - Ne pas comprendre que la POO ne sert à rien et que l'on perd notre temps depuis 30 ans.

Dans les années 1970

- The art of **programming** is the art of **organizing complexity**.
- **Simplicity** is the prerequisite of **reliability**.
- **Programming** is the most difficult branch of **applied mathematics**.



Edsger W. Dijkstra,
Prix Turing en 1972

Les sources de la complexité

- ❑ Gestion d'état
- ❑ Complexité cyclomatique (flow of control)
- ❑ Volume de code

Out of the Tar Pit, Ben Mosley, Peter Marks – 2006

No Silver Bullet — Essence and Accident in Software Engineering,
Frederick P. Brooks, Jr - 1986

Gestion des états

- ❑ Qui à déjà eu ~~des bugs~~, des comportements inattendus qui disparaissent quand :
 - ▣ On réexécute la même fonctionnalité
 - ▣ On redémarre l'application, et/ou l'ordinateur
 - ▣ On réinstalle l'application

- ❑ Ce sont les symptômes des bugs liés à la gestion des états.

Gestion des états

```
var myList = Enumerable.Range(-10000, 20001).ToList();  
var action1 = new Action(() => Console.WriteLine(myList.Sum()));  
var action2 = new Action(() => { myList.Sort(); Console.WriteLine(myList.Sum()); });  
  
action1();  
action2();
```

Gestion des états

```
var myList = Enumerable.Range(-10000, 20001).ToList();  
var action1 = new Action(() => Console.WriteLine(myList.Sum()));  
var action2 = new Action(() => { myList.Sort(); Console.WriteLine(myList.Sum()); });  
  
action1();  
action2();
```

```
Parallel.Invoke(action1, action2);|
```


Gestion des états

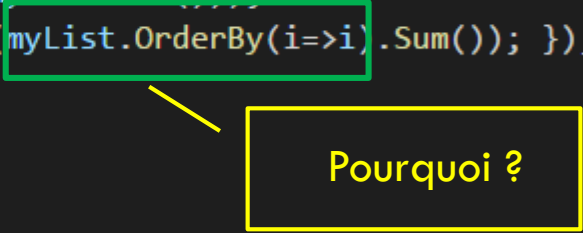
```
var myList = Enumerable.Range(-10000, 20001).ToList();  
var action1 = new Action(() => Console.WriteLine(myList.Sum()));  
var action2 = new Action(() => { myList.Sort(); Console.WriteLine(myList.Sum()); });  
  
action1();  
action2();
```

```
Parallel.Invoke(action1, action2);|
```

Comportement non-déterministe

Gestion des états

```
var myList = Enumerable.Range(-10000, 20001).ToList();  
var action1 = new Action(() => Console.WriteLine(myList.Sum()));  
var action2 = new Action(() => {Console.WriteLine(myList.OrderBy(i=>i).Sum()); });  
  
action1();  
action2();  
  
Parallel.Invoke(action1, action2);
```



A yellow rectangular box with the text "Pourquoi ?" is positioned to the right of the code. A yellow arrow points from the box to the expression `myList.OrderBy(i=>i).Sum()` in the code, which is enclosed in a green rectangular box. This highlights the stateful variable `myList` used in the second action.

Comportement déterministe

Gestion des états

```
var myList = Enumerable.Range(-10000, 20001).ToList();  
Console.WriteLine($"Avg : {AnotherComponent.CalculateAverage(myList)}");  
Console.WriteLine($"Sum : {myList.Sum()}");
```

```
var myList = Enumerable.Range(-10000, 20001).ToList();  
Console.WriteLine($"Avg : {AnotherComponent.CalculateAverage(new List<int>(myList))}");  
Console.WriteLine($"Sum : {myList.Sum()}");
```

Complexité cyclomatique (flow of control)

```
0 references
public IEnumerable<string> GetNBugsFromFilesImperative(
    string[] logFilePath,
    int nbMax)
{
    List<string> result = new List<string>();
    foreach (var filePath in logFilePath)
    {
        if (File.Exists(filePath))
        {
            var lines = File.ReadAllLines(filePath);
            foreach (var line in lines)
            {
                if (line.StartsWith("BUG"))
                {
                    result.Add(line);
                }
                if (result.Count == nbMax)
                {
                    return result;
                }
            }
        }
    }
    return result;
}
```

- ❑ **Programmation impérative == Dire comment on fait.**
- ❑ **Familier != Lisible**
- ❑ **La tentation d'ajouter des commentaires**

Les Commentaires ?



"Now! ... That should clear up
a few things around here!"



Les Commentaires ?



"Now! ... That should clear up
a few things around here!"



**ON PEUT FAIRE
UN MEILLEUR CODE !!!**

Complexité cyclomatique (flow of control)

```
public IEnumerable<string> GetNBugsFromFilesImperative(
    string[] logFilePath,
    int nbMax)
{
    List<string> result = new List<string>();
    foreach (var filePath in logFilePath)
    {
        if (File.Exists(filePath))
        {
            var lines = File.ReadAllLines(filePath);
            foreach (var line in lines)
            {
                if (line.StartsWith("BUG"))
                {
                    result.Add(line);
                }
                if (result.Count == nbMax)
                {
                    return result;
                }
            }
        }
    }
    return result;
}
```

Filtre

Transformation

Filtre

Limite

Complexité cyclomatique (flow of control)

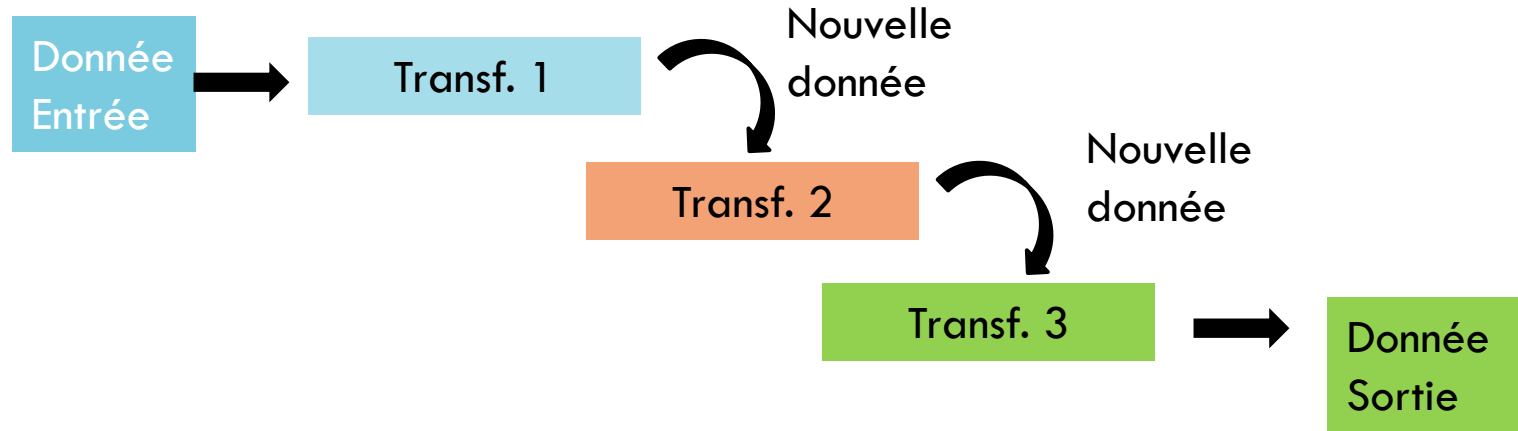
```
public IEnumerable<string> GetNBugsFromFilesDeclarative(
    string[] logFilePath,
    int nbMax){
    return logFilePath
        .Where(filePath=>File.Exists(filePath))
        .SelectMany(filePath=>File.ReadLines(filePath))
        .Where(line=>line.StartsWith("BUG"))
        .Take(nbMax);
}
```

□ **Programmation
déclarative == Dire
ce que l'on fait.**

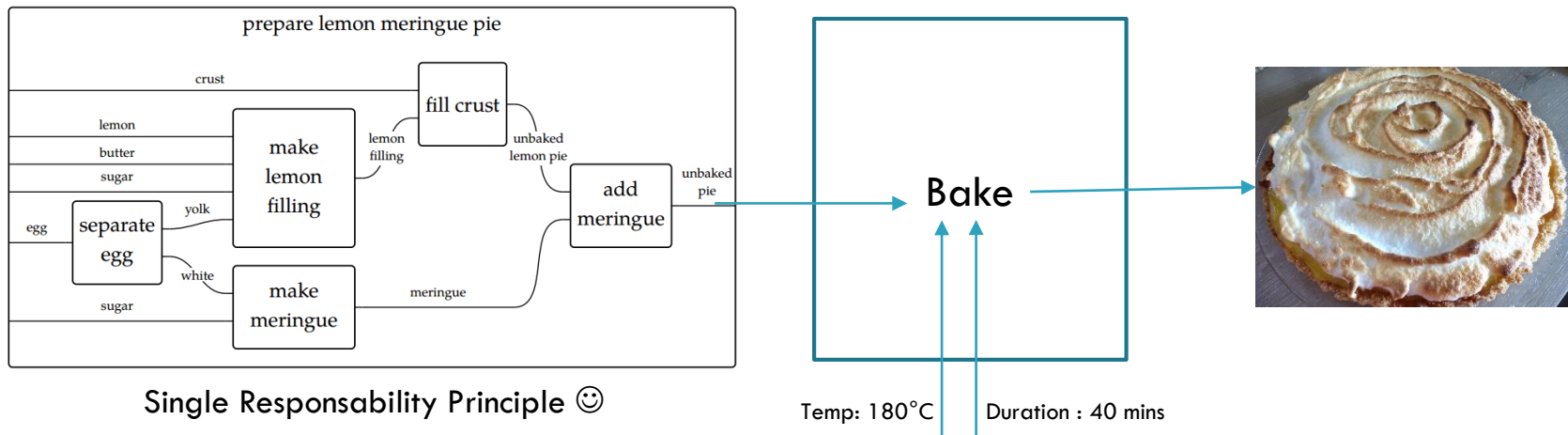
La vision de la PF

- ❑ Résoudre un problème complexe :
 - ❑ en l'**analysant finement**
 - ❑ en le **divisant** en problèmes plus simples
 - ❑ en trouvant des **solutions simples** pour les problèmes plus simples
 - ❑ en **combinant** les solutions simples, pour obtenir la solution du problème complexe
- ❑ Vision mathématique

La vision de la PF \Rightarrow Flux de transformations



Si j'avais à l'expliquer différemment



Définition PF 1

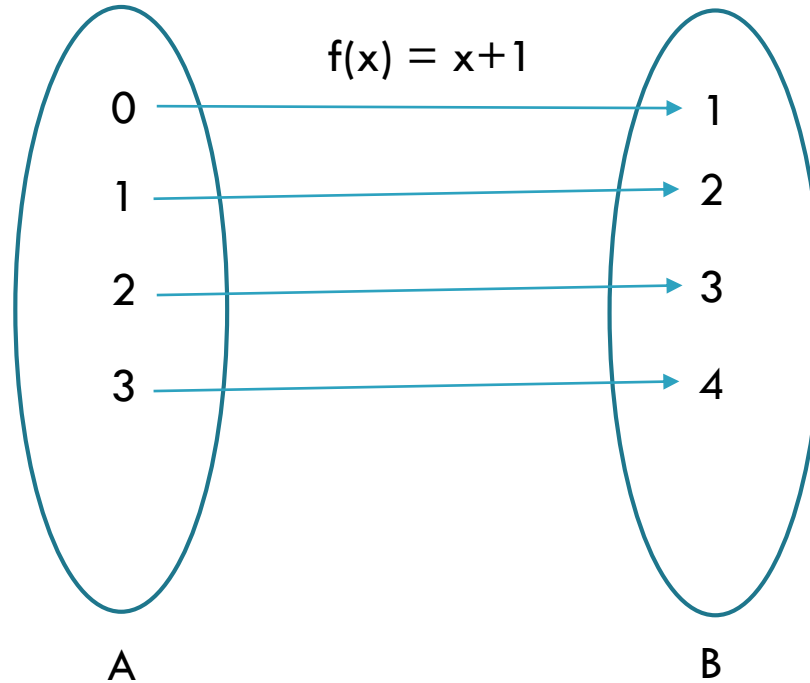
- La Programmation Fonctionnelle
 - ~~▣ Programmer avec des fonctions~~
 - ▣ Programmer avec des fonctions mathématiques

Fonction mathématique

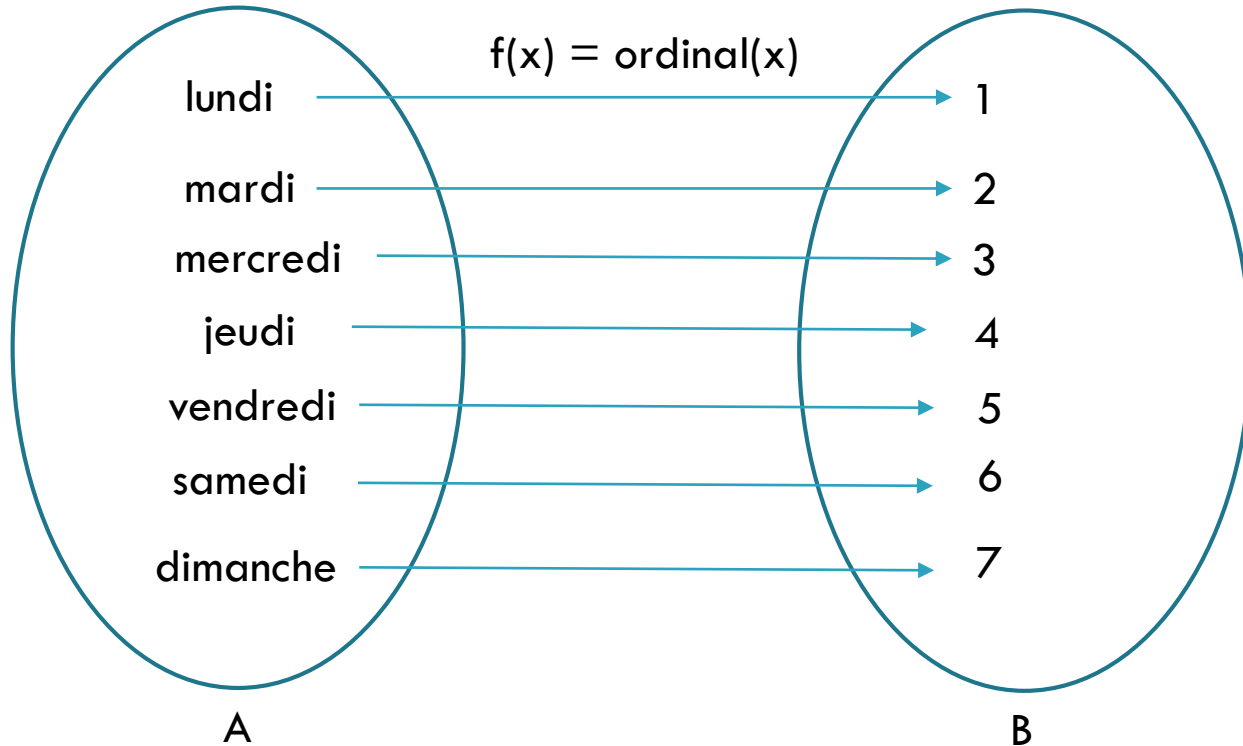
- Représente une association unique entre certains éléments d'un domaine (l'ensemble d'entrée) et certains éléments d'un Codomaine (l'ensemble de sortie)

Etant donné $f: A \rightarrow B$, $\forall x, y \in A$
Si $f(x) \neq f(y)$ alors $x \neq y$

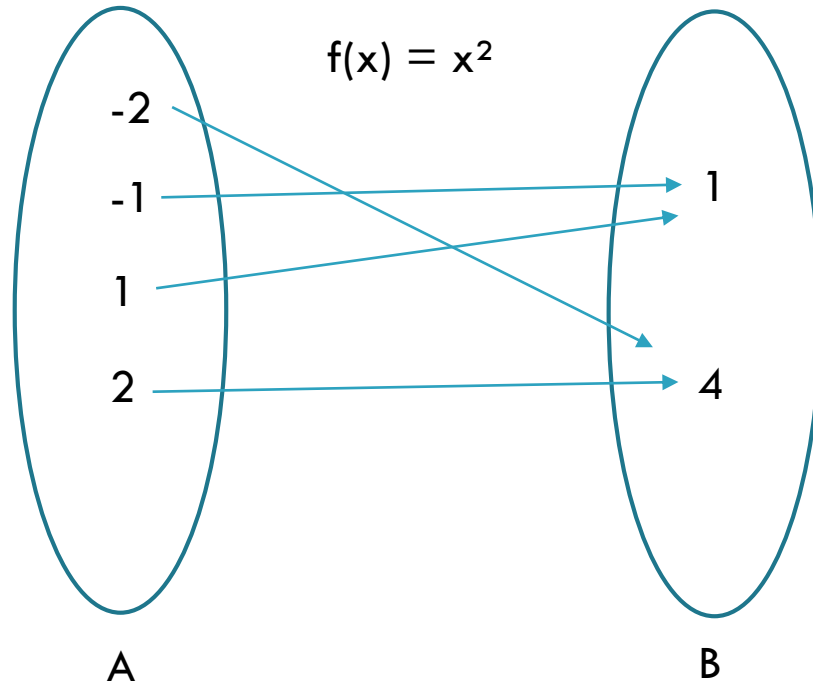
Exemple



Exemple



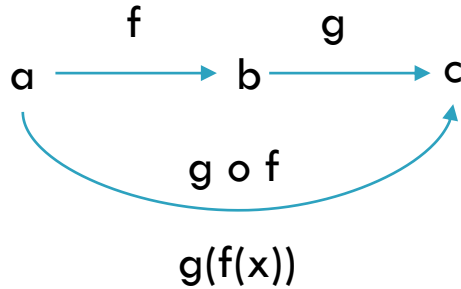
Exemple



Quelque propriétés intéressantes

- Pureté, Isolation
- Déterminisme
- Transparence référentielle (cad. $25 == \text{carré}(5)$ je peux remplacer tous les $\text{carré}(5)$ par 25 sans créer de changement) \Rightarrow les fonctions sont des données
- Toujours une entrée et une sortie

Composition



□ Associativité

$$\begin{aligned} \blacksquare (f \circ g) \circ h &= f \circ (g \circ h) \\ &= f \circ g \circ h \end{aligned}$$

□ Identité

$$\blacksquare \text{Id}(x) \rightarrow x$$

Les promesses de la PF

- ❑ Code plus :
 - ▣ Concis, maintenable, expressif, robuste, testable, apte à la parallélisation
- ❑ Il ne faut pas comprendre que c'est la solution miracle à tous ces problèmes

Différence avec la POO

□ Focus

- ▣ Fonctions
- ▣ Isolation
- ▣ Transformation de données

□ Focus

- ▣ Objets
- ▣ Encapsulation
- ▣ Modification d'état

Définition PF 2 (simplifiée et réductrice)

- ❑ La Programmation Fonctionnelle est une série de pratiques et patterns basée sur 2 principes fondamentaux :
 - ▣ Immutabilité
 - ▣ Les fonctions sont des données
- ❑ La POO et la PF sont complémentaires et non pas mutuellement exclusives
- ❑ La FPOO est possible

FPOO en action

Attention : le code est donné pour illustrer des concepts il n'est pas forcément prêt pour la prod 😊

Un rappel

- Méthodes d'extension en C#
- Func



Pipeline de transformations

Chaines de fonctions

Code initial : programmation impérative

```
static void Main(string[] args)
{
    Console.ForegroundColor = ConsoleColor.Green;
    byte[] data;
    using (var stream = DataStreams.GetXMenStream())
    {
        data = new byte[stream.Length];
        stream.Read(data, 0, data.Length);
    }

    var xMen = Encoding.UTF8.GetString(data)
        .Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries)
        .Select(line =>
        {
            var values = line.Split(";");
            return Tuple.Create(values[0], values[1]);
        });
    System.Console.WriteLine(AsHtmlComboBox("comboXMen", xMen));

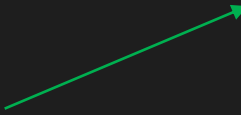
    Console.ForegroundColor = ConsoleColor.White;
    Console.ReadLine();
}
```

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");

    return htmlBuilder.ToString();
}
```

Amélioration

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");
    return htmlBuilder.ToString();
}
```



```
htmlBuilder
    .AppendLine($"<select id=\"{controlId}\">")
    .When(() => isOptional, (sb) => sb.AppendLine("<option>None</option>"));
```

```
10 references
public static T When<T>(this T @this, Func<bool> predicate, Func<T,T> fn){
    return predicate() ? fn(@this) : @this;
}
```

Amélioration

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");
    return htmlBuilder.ToString();
}
```

```
htmlBuilder
    .AppendLine($"<select id=\"{controlId}\">")
    .When(()=>isOptional, (sb)=>sb.AppendLine("<option>None</option>"));
```

10 references

```
public static T When<T>(this T @this, Func<bool> predicate, Func<T,T> fn){
    return predicate() ? fn(@this) : @this;
}
```

Amélioration

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");
    return htmlBuilder.ToString();
}
```

```
htmlBuilder
    .AppendLine($"<select id=\"{controlId}\">")
    .When(() => isOptional, (sb) => sb.AppendLine("<option>None</option>"));
```

10 references

```
public static T When<T>(this T @this, Func<bool> predicate, Func<T,T> fn){
    return predicate() ? fn(@this) : @this;
}
```

Amélioration

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");
    return htmlBuilder.ToString();
}
```

```
htmlBuilder
    .AppendLine($"<select id=\"{controlId}\">")
    .When(() => isOptional, (sb) => sb.AppendLine("<option>None</option>"));
```

10 references

```
public static T When<T>(this T @this, Func<bool> predicate, Func<T,T> fn){
    return predicate() ? fn(@this) : @this;
}
```

C'était une première composition

```
10 references  
public static T When<T>(this T @this, Func<bool> predicate, Func<T,T> fn){  
    return predicate() ? fn(@this) : @this;  
}
```

- ❑ Fonctions comme arguments
- ❑ Code simple et testable

Amélioration

```
public static string AsHtmlComboBox_Step1ControlFlow(
    string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();

    htmlBuilder
        .AppendLine($"<select id=\"{controlId}\">")
        .When(() => isOptional, (sb) => sb.AppendLine("<option>None</option>"));

    foreach (var data in dataSource) //let's try to remove this
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");

    return htmlBuilder.ToString();
}
```

La prochaine fois que j'aurais,
à écrire un code similaire,
ce serait bien de ne pas
avoir à dupliquer le code.

```
.AppendSequence(dataSource, [sb, data] => sb.AppendLine($"<option id={data.Item1}>{data.Item2}</option>"))
```

Impératif vs Déclaratif

```
public static string AsHtmlComboBox(string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    var htmlBuilder = new StringBuilder();
    htmlBuilder.AppendLine($"<select id=\"{controlId}\">");
    if (isOptional)
    {
        htmlBuilder.AppendLine("<option>None</option>");
    }
    foreach (var data in dataSource)
    {
        htmlBuilder.AppendLine($"<option id={data.Item1}>{data.Item2}</option>");
    }
    htmlBuilder.AppendLine("</select>");

    return htmlBuilder.ToString();
}
```

```
public static string AsHtmlComboBox_Step2ControlFlow(
    string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    return new StringBuilder()
        .AppendLine($"<select id=\"{controlId}\">")
        .When(() => isOptional, (sb) => sb.AppendLine("<option>None</option>")) //we can do better
        .AppendSequence(dataSource, (sb, data) => sb.AppendLine($"<option id={data.Item1}>{data.Item2}</option>"))
        .AppendLine("</select>")
        .ToString();
}
```

```
public static string AsHtmlComboBox(
    string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    return new StringBuilder()
        .AppendLine($"<select id=\"{controlId}\">")
        .When(Is.True(isOptional), (sb) => sb.AppendLine("<option>None</option>"))
        .AppendSequence(dataSource, (sb, data) => sb.AppendLine($"<option id={data.Item1}>{data.Item2}</option>"))
        .AppendLine("</select>")
        .ToString();
}
```


Déclaratif

```
public static string AsHtmlComboBox(
    string controlId,
    IEnumerable<Tuple<string, string>> dataSource,
    bool isOptional = false)
{
    return new StringBuilder()
        .AppendLine($"<select id=\"{controlId}\">")
        .When(Is.True(isOptional), (sb) => sb.AppendLine("<option>None</option>"))
        .AppendSequence(dataSource, (sb, data) => sb.AppendLine($"<option id={data.Item1}>{data.Item2}></option>"))
        .AppendLine("</select>")
        .ToString();
}
```

- ✓ Concis
- ✓ Simple
- ✓ Expressif
- ✓ Testable
- ✓ Pur (sans partage d'état)

Code initial : programmation impérative

```
static void Main(string[] args)
{
    Console.ForegroundColor = ConsoleColor.Green;
    byte[] data;
    using (var stream = DataStreams.GetXMenStream())
    {
        data = new byte[stream.Length];
        stream.Read(data, 0, data.Length);
    }

    var xMen = Encoding.UTF8.GetString(data)
        .Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries)
        .Select(line =>
        {
            var values = line.Split(";");
            return Tuple.Create(values[0], values[1]);
        });
    System.Console.WriteLine(AsHtmlComboBox("comboXMen", xMen));

    Console.ForegroundColor = ConsoleColor.White;
    Console.ReadLine();
}
```

Amélioration

```
byte[] data;
using (var stream = DataStreams.GetXMenStream())
{
    data = new byte[stream.Length];
    stream.Read(data, 0, data.Length);
}
var xMen = Encoding.UTF8.GetString(data)
```

```
byte[] data = Disposable.Using(
    () => DataStreams.GetXMenStream(),
    (stream) =>
    {
        var bytes = new byte[stream.Length];
        stream.Read(bytes, 0, bytes.Length);
        return bytes;
    });
var xMen = Encoding.UTF8.GetString(data) //ho
```

5 references

```
public static TResult Using<TDisposable, TResult>(
    Func<TDisposable> factory,
    Func<TDisposable, TResult> fn)
    where TDisposable:IDisposable
{
    using(var ressource = factory()){
        return fn(ressource);
    }
}
```

Amélioration

```
private static void ExecuteImproveUsingStatement()
{
    byte[] data = Disposable.Using(
        () => DataStreams.GetXMenStream(),
        (stream) =>
        {
            var bytes = new byte[stream.Length];
            stream.Read(bytes, 0, bytes.Length);
            return bytes;
        });

    var xMen = Encoding.UTF8.GetString(data) //how can we let the flow of transformations continue
        .Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries)
        .Select(line =>
        {
            var values = line.Split(";");
            return Tuple.Create(values[0], values[1]);
        });

    System.Console.WriteLine(AsHtmlComboBox("comboXMen", xMen, true));
}
```

Transformation

Transformations

Action

Amélioration

9 references

```
public static TResult Map<TSource,TResult>(this TSource @this,Func<TSource,TResult> transformFn){  
    return transformFn(@this);  
}
```

8 references

```
public static T Act<T>(this T @this, Action<T> actFn){  
    actFn(@this);  
    return @this;  
}
```

Amélioration

```
private static void ExecuteImproveTranformationsFlow2()
{
    Disposable
        .Using(
            () => DataStreams.GetXMenStream(),
            (stream) =>{
                var bytes = new byte[stream.Length]; //improve this transformation
                stream.Read(bytes, 0, bytes.Length);
                return bytes;
            })
        .Map((data) => Encoding.UTF8.GetString(data))
        .Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries)
        .Select(line =>
            {
                var values = line.Split(";"); //improve this transformation
                return Tuple.Create(values[0], values[1]);
            })
        .Map((tuples) => AsHtmlComboBox("comboXMen", tuples, true))
        .Act((html) => Console.WriteLine(html));
}
```

Amélioration

```
private static void ExecuteImproveTranformationsFlow4()
{
    Disposable
        .Using(() => DataStreams.GetXMenStream(), (stream) => ReadStream(stream))
        .Map((bytes) => Encoding.UTF8.GetString(bytes))
        .Split(Environment.NewLine, StringSplitOptions.RemoveEmptyEntries)
        .Select(line => line.Split(";").Map(values => Tuple.Create(values[0], values[1])))
        .Map((tuples) => AsHtmlComboBox("comboXMen", tuples, true))
        .Act((html) => Console.WriteLine(html));
}
```



Fonctions honnêtes

Je dis ce que fais, je fais ce que je dis !

Implémentation initiale

```
public string GetHistoricalCategory(int yearOfPublication){
    if(yearOfPublication<1963 || yearOfPublication>2006){
        throw new ArgumentOutOfRangeException("Only publications from 1963 to 2006 are handled");
    }

    if(yearOfPublication>=1963 && yearOfPublication<1970){
        return "Original";
    }

    if(yearOfPublication>=1970 && yearOfPublication<2000){
        return "Revival";
    }

    if(yearOfPublication>=2000 && yearOfPublication<=2006){
        return "Revolution";
    }
    return "Unknown";
}
```

❑ Faits

- Uniquement les publications entre 1963 et 2006 sont gérés par l'application
- Les périodes sont {Original, Revival, Revolution}

❑ Problèmes

- La fonction est malhonnête. Pourquoi ?

Amélioration

```
14 references
public struct YearOfPublication
{
    2 references
    private int _year;

    1 reference
    public YearOfPublication(int year)
    {
        if(year<1963 || year>2006){
            throw new ArgumentOutOfRangeException("Only
        }
        _year = year;
    }

    6 references
    public int Value => _year;
```

```
public string GetHistoricalCategory(YearOfPublication year){
    if(year>=1963 && year<1970){
        return "Original";
    }

    if(year>=1970 && year<2000){
        return "Revival";
    }

    if(year>=2000 && year<=2006){
        return "Revolution";
    }
    return "unknown";
}
```

Amélioration

```
public string GetHistoricalCategory2(YearOfPublication year){  
    var rules = new List<Tuple<string,Func<YearOfPublication,bool>>>(){  
        Tuple.Create<string,Func<YearOfPublication,bool>>("Original",(y)=>y>=1963 && y<1970),  
        Tuple.Create<string,Func<YearOfPublication,bool>>("Revival",(y)=>y>=1970 && y<2000),  
        Tuple.Create<string,Func<YearOfPublication,bool>>("Revolution",(y)=>y>=2000 && y<=2006),  
    };  
    return rules  
        .Where(r=>r.Item2(year))  
        .Select(r=>r.Item1)  
        .FirstOrDefault();  
}
```

C'est mieux mais ce n'est pas à 100% honnête !



Gérer le NULL

Savoir dire peut-être !

Code initial

```
static void Main(string[] args)
{
    Console.ForegroundColor = ConsoleColor.Green;
    PrintPower("Professor X");
    Console.ForegroundColor = ConsoleColor.White;
}
```

```
1 reference
private static void PrintPower(string characterName)
{
    var repository = new XMenCharactersRepository();
    var character = repository.GetXMen(characterName);
    Console.WriteLine($"{characterName} is {character.Powers.FirstOrDefault()}");
}
```

```
public ImmutableCharacter GetXMen(string characterName)
{
    return _data.FirstOrDefault(x => x.Name.Equals(characterName, StringComparison.OrdinalIgnoreCase));
}
```

Amélioration

0 references

```
private static void PrintPower2(string characterName){  
    new XMENCharactersRepository()  
        .GetXMen2(characterName)  
        .Match(  
            whenNothing: () => $"Can't find {characterName}",  
            whenJust: (xmen) => $" {characterName} is {xmen.Powers.FirstOrDefault()}"  
        )  
        .Act(System.Console.WriteLine);  
}
```

3 references

```
public Maybe<ImmutableCharacter> GetXMEN2(string characterName)  
{  
    return _data.FirstOrDefault(x => x.Name.Equals(characterName, StringComparison.OrdinalIgnoreCase)).AsMaybe();  
}
```

Amélioration

```
public class Maybe<TValue>
{
    3 references
    private TValue _value;
    3 references
    private bool _isJust;
    1 reference
    private Maybe(){}
    1 reference
    private Maybe(TValue value)
    {
        if (value == null) throw new ArgumentNullException(nameof(value));
        _value = value;
        _isJust = true;
    }
    3 references
    public static Maybe<TValue> Nothing() => new Maybe<TValue>();
    2 references
    public static Maybe<TValue> Just(TValue value) => new Maybe<TValue>(value);

    3 references
    public TResult Match<TResult>(Func<TResult> whenNothing, Func<TValue, TResult> whenJust)
    {
        return _isJust ? whenJust(_value) : whenNothing();
    }
    2 references
    public TValue GetValueOrDefault(TValue defaultValue) => _isJust ? _value : defaultValue;
}
```

Beaucoup de choses peuvent être développées autour de Maybe.

Objets Immuables

Eviter les états partagés => problèmes d'accès concurrent


```
public class ImmutableCharacter
{
    3 references
    public string Name { get; }

    3 references
    public int YearOfPublication { get; }

    5 references
    public IReadOnlyList<string> Powers { get; }

    0 references
    private ImmutableCharacter() { }

    2 references
    public ImmutableCharacter(string name, int yearOfPublication, List<string> powers)
    { ...
    }

    1 reference
    public ImmutableCharacter AddPower(string power)
    {
        return new ImmutableCharacter(
            Name,
            YearOfPublication,
            new List<string>(Powers) { power });
    }
}
```

Conclusion

- ❑ Des patterns de PF peuvent être implémentés dans tous les langages qui acceptent des pointeurs de fonctions
- ❑ C# et Java évoluent afin de faciliter la PF
 - ▣ Mais le debug reste compliqué
- ❑ Linq c'est de la PF
- ❑ POO et PF sont compatibles à 100%



**Your code should
better be SOLID!**



Uncle Bob Martin

@unclebobmartin

It is perfectly possible to write a program that is both object oriented and functional. Not only is it possible, it is desireable.

[Translate Tweet](#)