

F#



Histoire de F#

F# est un langage de programmation fonctionnel développé par **Don Syme** et son équipe chez **Microsoft Research** au début des années 2000.

Il a été conçu pour s'intégrer avec la plateforme .NET tout en exploitant les avantages de la programmation fonctionnelle.

En 2005, F# a été dévoilé publiquement, et en 2007, il est devenu un produit officiel de Microsoft, intégré dans **Visual Studio**.

En 2010, F# a gagné en popularité avec une adoption croissante dans des domaines comme la finance et les sciences.

En 2017, Microsoft a rendu F# **open source**, permettant à la communauté d'y contribuer activement.

Aujourd'hui, F# est un langage moderne et puissant utilisé dans l'écosystème .NET, apprécié pour ses caractéristiques comme l'immuabilité par défaut, le pattern matching et son interopérabilité avec C#.

Pourquoi s'intéresser à F# ?



Code concis

Le code F# est très concis. On peut souvent faire en 1 ou 2 lignes ce qui demande 5 ou 10 lignes en C#. Moins de code = moins de bugs.



Immuabilité par défaut

L'immuabilité par défaut rend le code plus fiable, surtout quand plusieurs parties du programme s'exécutent en même temps (multi-thread).

On évite ainsi les bugs liés à des données modifiées par d'autres morceaux du code.



Typage puissant

Le typage est puissant, avec notamment l'inférence de type. Vous n'avez quasiment jamais besoin d'écrire vous-même les types, le compilateur les déduit automatiquement.



Gestion de la concurrence

La gestion de la concurrence (plusieurs tâches en même temps) est particulièrement bien gérée en F# grâce aux workflows asynchrones.

Et surtout : **F# est interopérable** avec les autres langages .NET. Vous pouvez appeler du code C# depuis F#, et inversement.

Ce langage est utilisé dans des domaines exigeants comme :

- La finance (trading algorithmique, calcul quantitatif)
- La science des données
- Les APIs légères
- Ou encore le machine learning

Comment ça se code, F# ?

Une fonction simple en C#

```
public int Add(int x, int y) {  
    return x + y;  
}
```

C'est la syntaxe classique que vous connaissez en C#.

La même chose en F#

```
let add x y = x + y
```

C'est une syntaxe **extrêmement simple** et lisible. Ici, let est utilisé pour définir une fonction. Il n'y a pas besoin de types explicites : le compilateur les devine.

Définir un type de données en F#

Définition d'un record (structure de données)

```
type Person = { Name: string; Age: int }
```

Cela définit un **record immuable**. C'est l'équivalent d'une classe avec des propriétés en C#, mais de manière beaucoup plus concise.

Création d'une instance

```
let person1 = { Name = "Alice"; Age = 30 }
```

La création d'instances est également très simple et lisible. Notez l'absence du mot-clé "new".

Les records sont des structures de données immuables qui permettent de regrouper des valeurs connexes, similaires aux classes en programmation orientée objet, mais avec une approche fonctionnelle.



Le pattern matching en F#



Définition du pattern matching

Une technique puissante pour décomposer des données de manière structurée et élégante, en fonction de leur forme ou de leur structure.



Syntaxe élégante

Plus lisible qu'un switch en C#



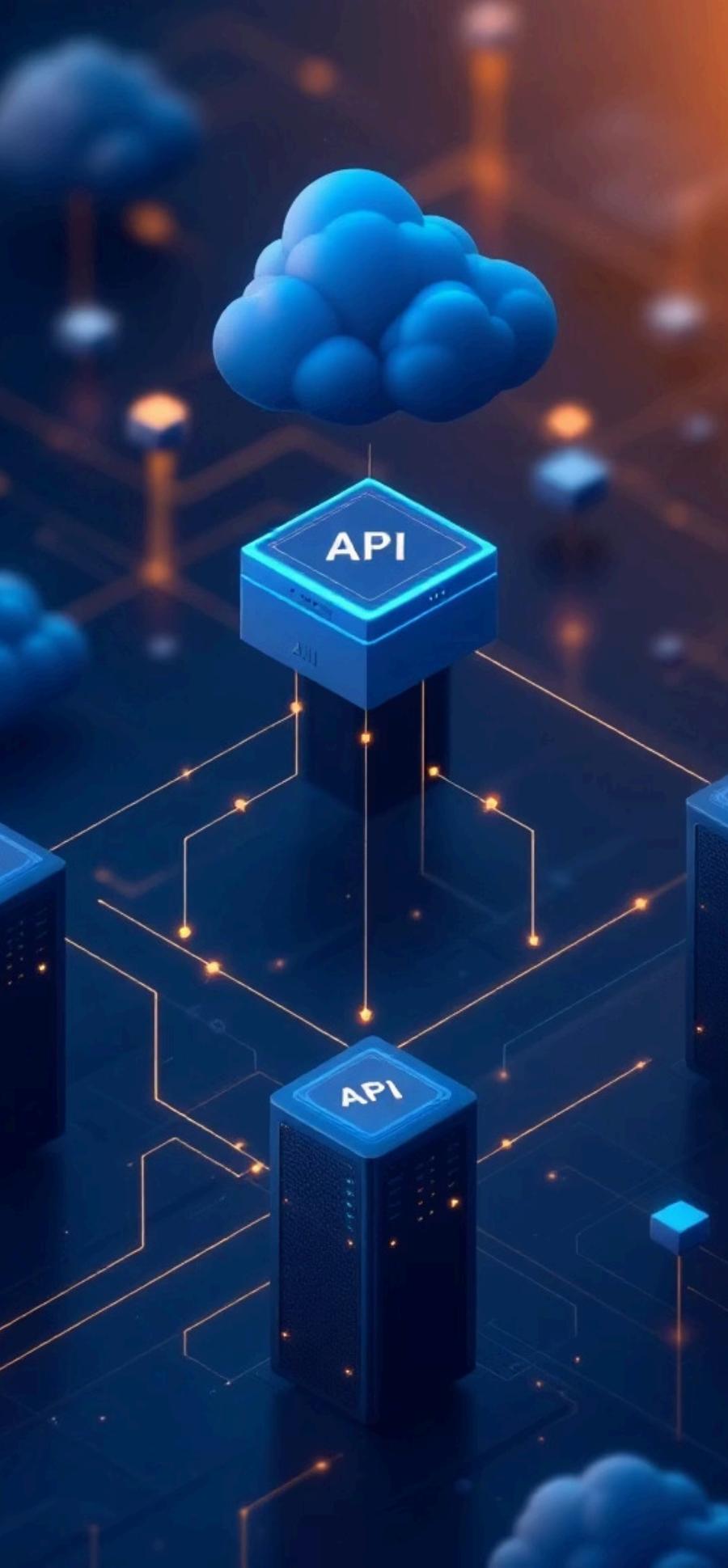
Sécurité accrue

Le compilateur vérifie l'exhaustivité des cas (si un cas n'est pas couvert => Erreur)

Une des fonctionnalités les plus puissantes de F# :

```
let describe n = match n with
| 0 -> "Zero"
| 1 -> "One"
| _ -> "Another number"
```

C'est beaucoup plus lisible et sécurisé qu'un switch en C#. Vous pouvez matcher non seulement des valeurs, mais aussi des structures de données.



Programmation asynchrone en F#



Définition

Création d'un workflow asynchrone

Attente

Utilisation de let! (await) pour attendre un résultat

Résultat

Retour de la valeur obtenue

Et l'asynchrone ?

```
let fetchData() = async {
    let! result = asyncHttpGet "http://api.exemple.com"
    return result
}
```

Avec le mot-clé `let!`, vous attendez un résultat de manière asynchrone, comme avec `await` en C#. F# utilise des **workflows** pour rendre le code asynchrone facile à lire.

Quand utiliser F# ?

Calculs intensifs

Si vous avez besoin de **calculs intensifs**, comme dans les applis financières ou scientifiques.

Programmation fonctionnelle

Si vous voulez explorer la **programmation fonctionnelle moderne** sans quitter l'écosystème .NET.



APIs légères

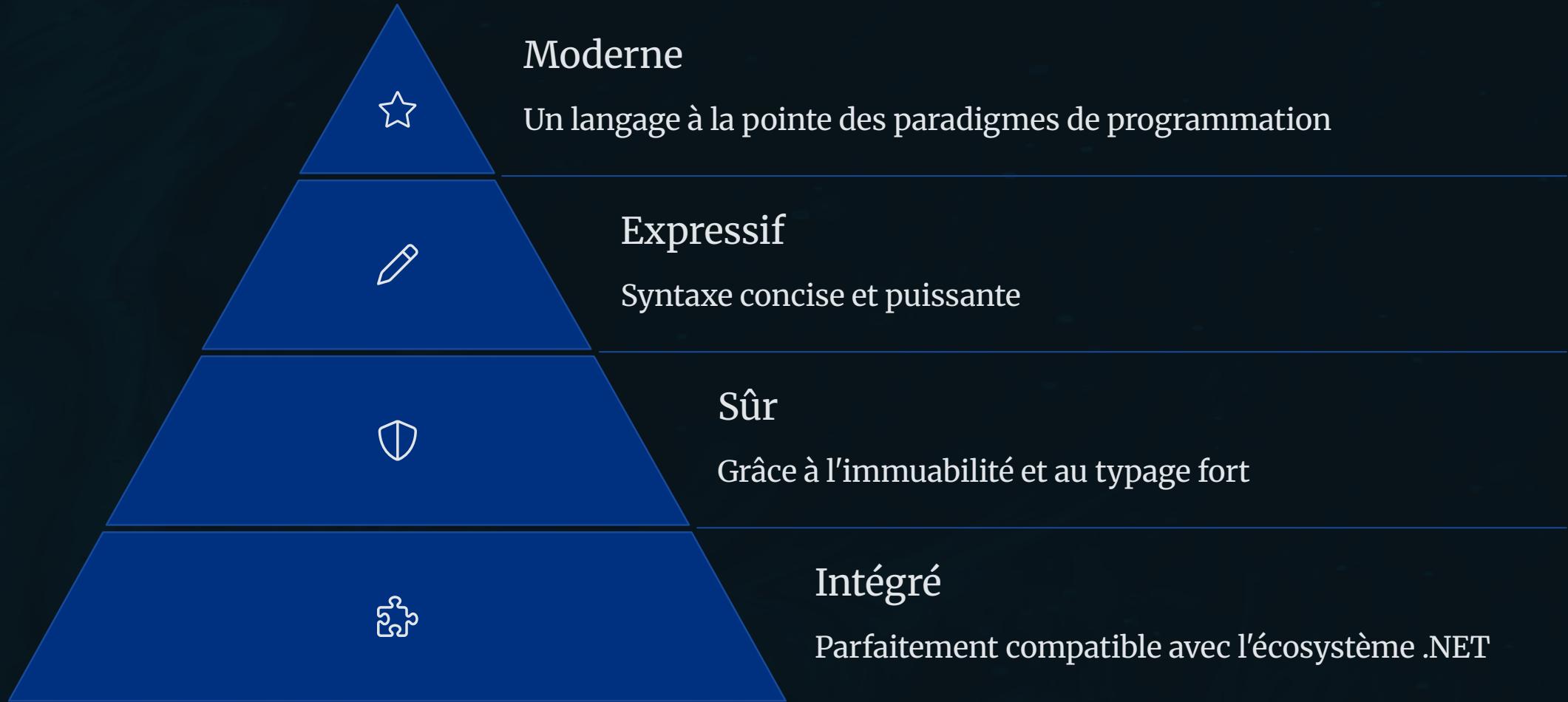
Pour les **APIs légères**, comme des microservices ou fonctions cloud (Azure Functions).

Prototypage

Si vous cherchez un langage rapide pour **prototyper** des idées, grâce à sa concision.

F# n'est pas destiné à remplacer C#, mais à **compléter** votre boîte à outils.

Conclusion



Pour conclure, F# est un langage :

- Moderne
- Expressif
- Très sûr grâce à l'immuabilité et au typage fort
- Et parfaitement intégré dans l'écosystème .NET