

Diversité et unité des langages de programmation

Dernière mise à jour le : 23/01/2024

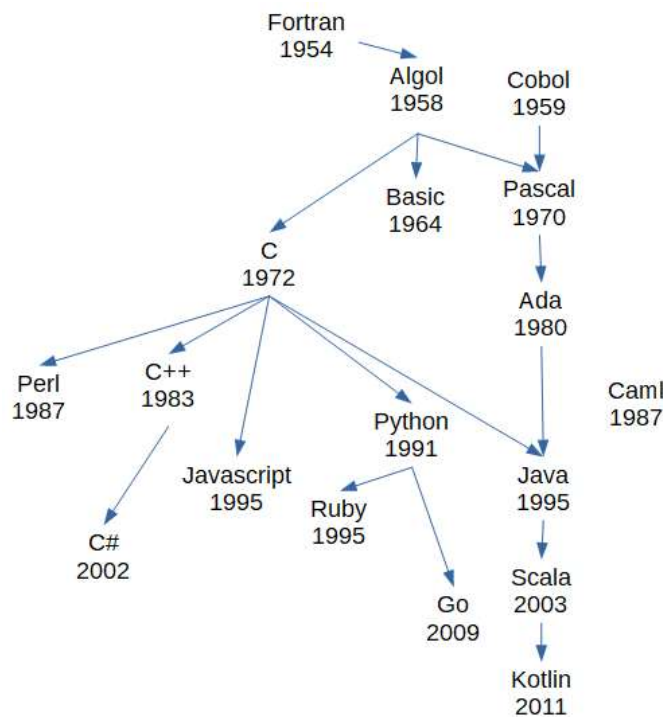
■ Introduction

Il n'existe pas de langage de programmation *universel*. De nombreux langages existent, chacun ayant ses spécificités, ses domaines d'application, ses points forts et ses points faibles.

i Par exemple, le langage Python est réputé pour être facile à prendre en main et permet d'écrire et d'exécuter rapidement des programmes. En revanche, il est réputé comme étant un langage "lent". Le langage C, par exemple, a des performances plus élevées car il est plus proche du langage machine.

L'évolution de techniques concernant différents aspects des langages de programmation (comme la programmation orientée objets, la gestion de la mémoire, des exceptions, des adresses mémoires) a entraîné l'apparition de nouveaux langages, ou l'évolution d'autres, pour inclure de nouvelles techniques.

C'est pourquoi, depuis la création de l'informatique, de nouveaux langages ont régulièrement vu le jour, souvent inspirés de langages déjà existants et incorporant ou développant de nouveaux aspects.



Généalogie (non exhaustive) des langages de programmation

Source : document Eduscol sur la [Diversité et unité des langages de programmation](#)

On constate en particulier la plupart des langages récents sont inspirés du langage C, qui a eu une place centrale parmi les langages de programmation. Par exemple :

- le noyau Linux est écrit en langage C,
- le langage Python est écrit en langage C (pour son implémentation classique CPython)

Selon l'index de TIOBE, le top 10 des langages informatiques pour le mois de mars 2023, est le suivant :

1. Python
2. C
3. Java
4. C++
5. C#
6. Visual Basic
7. JavaScript
8. SQL (pas un langage de programmation)
9. PHP
10. Go

Ces langages présentent des différences plus ou moins importantes, mais également beaucoup de points communs car tous les langages sont basés sur un **corpus de constructions élémentaires** : des variables, des affectations, des instructions conditionnelles, des boucles, des fonctions, etc.

On se propose dans la suite de comparer quelques langages, en s'aidant de ChatGPT pour nous proposer des traductions d'un langage à l'autre, chose pour laquelle l'agent conversationnel est particulièrement fort.



L'éditeur replit.com permet d'écrire dans différents langages directement dans la navigateur. N'hésitez pas à l'utiliser pour tester les différents langages.

■ Comparaisons de différents langages

On va demander à ChatGPT (dans sa version gratuite, version du 23 mars 2023) de traduire dans différents langages la fonction Python de recherche du maximum dans un tableau :

```
def maximum(T):  
    maxi = T[0]  
    for e in T:  
        if e > maxi:  
            maxi = e  
    return maxi
```



Les codes proposés par ChatGPT n'ont pas été vérifiés, donc il y a peut-être de petites erreurs. Il est intéressant de remarquer que ChatGPT commente lui-même le code qu'il a produit, sans qu'on le lui ait demandé. De façon générale, tout ce qu'il raconte demande vérification.

La conversation est retranscrite ci-dessous, si vous souhaitez voir la conversation d'origine, suivez ce lien : <https://shareg.pt/q1WflmQ>.



Travail à faire

L'objectif pour vous est de repérer les similitudes et les différences entre les différents langages de programmation proposés. On vous demande un résumé de quelques lignes, qui pourra être réalisé à l'écrit ou à l'oral.

Pour vous guider un peu, votre cheminement peut être le suivant :

- Lire les différents programmes et repérer la boucle permettant de parcourir les éléments du tableau, avec à l'intérieur une instruction conditionnelle permettant de savoir s'il faut mettre à jour le maximum ou non.
- Une fois cette structure repérée, se pencher sur les variations et les similitudes en répondant aux questions suivantes :
 - Comment les blocs d'instructions à l'intérieur d'une boucle ou d'un test sont-ils délimités ?
 - Comment sont structurées les boucles for ? les tests ?
 - Les types de données sont-ils spécifiés explicitement ? Doit-on indiquer que telle variable correspond à un entier ou un tableau ?

Voilà la conversation !

GE

Bonjour, peux-tu me traduire la fonction suivante, écrite en Python, dans le langage C ?

```
def maximum(T):
    maxi = T[0]
    for e in T:
        if e > maxi:
            maxi = e
    return maxi
```



Bien sûr, voici la traduction de la fonction Python en langage C :

```
int maximum(int T[], int n) {
    int maxi = T[0];
    for (int i = 1; i < n; i++) {
        if (T[i] > maxi) {
            maxi = T[i];
        }
    }
    return maxi;
}
```

Notez que cette version prend deux arguments : un tableau `T` et sa taille `n`. En effet, contrairement à Python, en C il n'est pas possible de déduire la taille d'un tableau à partir de son nom. Il faut donc la passer explicitement en argument.

De plus, le type de retour est ici `int`, puisque la fonction renvoie la valeur maximale du tableau, qui est un entier.

GE

Merci, et en langage Java ? Peux-tu me donner une version avec un parcours par valeur et une autre avec un parcours par indice ?



Oui, voici une traduction de la fonction Python en Java avec un parcours par valeur :

```
public static int maximum(int[] T) {
    int maxi = T[0];
    for (int e : T) {
        if (e > maxi) {
            maxi = e;
        }
    }
    return maxi;
}
```

Et voici une version avec un parcours par indice :

```
public static int maximum(int[] T) {
    int maxi = T[0];
    for (int i = 1; i < T.length; i++) {
        if (T[i] > maxi) {
            maxi = T[i];
        }
    }
}
```

```
}
return maxi;
}
```

Notez que dans les deux versions, le type de retour est `int`, puisque la fonction renvoie la valeur maximale du tableau, qui est un entier. De plus, la version avec un parcours par indice utilise la propriété `length` du tableau pour déterminer sa taille, ce qui est possible en Java.

GE Merci ! Et en C++ ?



Voici la traduction de la fonction Python en C++ avec un parcours par valeur :

```
int maximum(const std::vector<int>& T) {
    int maxi = T[0];
    for (int e : T) {
        if (e > maxi) {
            maxi = e;
        }
    }
    return maxi;
}
```

Et voici une version avec un parcours par indice :

```
int maximum(const std::vector<int>& T) {
    int maxi = T[0];
    for (int i = 1; i < T.size(); i++) {
        if (T[i] > maxi) {
            maxi = T[i];
        }
    }
    return maxi;
}
```

Notez que la version avec un parcours par valeur utilise une référence constante (`const std::vector&`) pour le vecteur d'entrée, ce qui évite une copie inutile du vecteur. La version avec un parcours par indice utilise la méthode `size()` pour déterminer la taille du vecteur.

GE Et cela donne quoi en PHP ?



Voici la traduction de la fonction Python en PHP avec un parcours par valeur :

```
function maximum($T) {
    $maxi = $T[0];
    foreach ($T as $e) {
        if ($e > $maxi) {
            $maxi = $e;
        }
    }
}
```

```
    return $maxi;
}
```

Et voici une version avec un parcours par indice :

```
function maximum($T) {
    $maxi = $T[0];
    for ($i = 1; $i < count($T); $i++) {
        if ($T[$i] > $maxi) {
            $maxi = $T[$i];
        }
    }
    return $maxi;
}
```

Notez que les tableaux en PHP sont en fait des tableaux associatifs, mais la fonction `maximum` telle que définie ci-dessus suppose que le tableau en entrée est indexé par des entiers consécutifs à partir de zéro, de sorte qu'elle fonctionne correctement dans ce cas.

GE

Cela veut dire que les tableaux PHP correspondent aux dictionnaires de Python ?



En PHP, un tableau est une structure de données qui peut être utilisée pour stocker des valeurs indexées par des clés. Cependant, contrairement aux tableaux en C++, Java ou Python, les tableaux en PHP peuvent être associatifs, c'est-à-dire que les clés ne sont pas limitées à des entiers, mais peuvent être des chaînes de caractères, des objets, etc.

Ainsi, en PHP, un tableau peut être utilisé pour stocker des valeurs indexées par des clés arbitraires, ce qui est plus similaire à un dictionnaire en Python qu'à un tableau en C++ ou Java.

Cependant, en PHP, il est également possible d'utiliser des tableaux indexés par des entiers, qui peuvent être utilisés comme des tableaux classiques. C'est ce type de tableau qui est utilisé dans la fonction `maximum` que j'ai traduite en PHP dans ma réponse précédente.

GE

Comment s'écrit la fonction en JavaScript ?



Voici la traduction de la fonction Python en JavaScript avec un parcours par valeur :

```
function maximum(T) {
    let maxi = T[0];
    for (let e of T) {
        if (e > maxi) {
            maxi = e;
        }
    }
    return maxi;
}
```

Et voici une version avec un parcours par indice :

```
function maximum(T) {
  let maxi = T[0];
  for (let i = 1; i < T.length; i++) {
    if (T[i] > maxi) {
      maxi = T[i];
    }
  }
  return maxi;
}
```

Notez que JavaScript possède une boucle `for...of` qui permet de parcourir les éléments d'un tableau sans avoir besoin d'utiliser des indices. La version avec un parcours par indice utilise la propriété `length` du tableau pour déterminer sa taille.



Et en Visual Basic ?



Voici la traduction de la fonction Python en Visual Basic avec un parcours par valeur :

```
Function Maximum(T() As Integer) As Integer
  Dim maxi As Integer = T(0)
  For Each e As Integer In T
    If e > maxi Then
      maxi = e
    End If
  Next
  Return maxi
End Function
```

Et voici une version avec un parcours par indice :

```
Function Maximum(T() As Integer) As Integer
  Dim maxi As Integer = T(0)
  For i As Integer = 1 To T.Length - 1
    If T(i) > maxi Then
      maxi = T(i)
    End If
  Next
  Return maxi
End Function
```

Notez que en Visual Basic, on utilise `As Integer` pour spécifier le type des éléments du tableau, et `()` pour indiquer que T est un tableau. La version avec un parcours par indice utilise la propriété `Length` du tableau pour déterminer sa taille.



Et en Basic ?



En Basic, la syntaxe dépend du dialecte de Basic que vous utilisez, car il existe de nombreuses variantes de Basic. Voici toutefois une traduction possible de la fonction Python en Basic :


```

FUNCTION Maximum(T())
  maxi = T(1)
  FOR i = 2 TO UBOUND(T)
    IF T(i) > maxi THEN
      maxi = T(i)
    END IF
  NEXT i
  Maximum = maxi
END FUNCTION

```

Cette version utilise un parcours par indice, et utilise la fonction `UBOUND` pour déterminer l'indice maximum du tableau T. Notez que la fonction Python utilise `T[0]` comme première valeur de départ pour la recherche du maximum, tandis que la version en Basic utilise `T(1)` car les tableaux en Basic commencent à l'indice 1 par défaut.

Il est important de noter que le langage Basic est souvent utilisé dans des environnements particuliers (par exemple, pour écrire des macros dans Microsoft Excel), et les fonctionnalités disponibles peuvent varier en fonction de l'environnement spécifique.

 Enfin, cela donne quoi dans le langage Go ?



Voici la traduction de la fonction Python en Go avec un parcours par valeur :

```

func maximum(T []int) int {
  maxi := T[0]
  for _, e := range T {
    if e > maxi {
      maxi = e
    }
  }
  return maxi
}

```

Et voici une version avec un parcours par indice :

```

func maximum(T []int) int {
  maxi := T[0]
  for i := 1; i < len(T); i++ {
    if T[i] > maxi {
      maxi = T[i]
    }
  }
  return maxi
}

```

Notez que Go utilise des slices pour représenter les tableaux dynamiques. Les slices sont des structures de données similaires aux tableaux, mais avec des fonctionnalités supplémentaires, telles que la capacité de redimensionner dynamiquement. Les deux versions de la fonction `maximum` utilisent des slices en entrée, et la version avec un parcours par indice utilise la fonction `len` pour déterminer la taille du slice.

 Est-ce possible de l'écrire en OCaml ?



Oui, voici la traduction de la fonction Python en OCaml avec un parcours par valeur :

```
let maximum t =
  let rec loop t' m =
    match t' with
    | [] -> m
    | hd :: tl -> if hd > m then loop tl hd else loop tl m
  in
  loop (List.tl t) (List.hd t)
```

Et voici une version avec un parcours par indice :

```
let maximum t =
  let rec loop t' m i =
    match t' with
    | [] -> m
    | hd :: tl -> if hd > m then loop tl hd (i+1) else loop tl m (i+1)
  in
  loop (List.tl t) (List.hd t) 1
```

Notez que le type de la fonction `maximum` est `int list -> int`, car en OCaml les tableaux sont représentés par des listes. La version avec un parcours par valeur utilise une fonction récursive `loop` qui prend en entrée la queue de la liste `t'` et la valeur maximale `m` trouvée jusqu'à présent. La fonction `List.hd` renvoie la première valeur de la liste, tandis que `List.tl` renvoie la queue de la liste.

La version avec un parcours par indice utilise également une fonction récursive `loop` qui prend en entrée la queue de la liste `t'`, la valeur maximale `m` trouvée jusqu'à présent, et l'indice `i` de l'élément en cours de traitement. Cette version utilise également les fonctions `List.hd` et `List.tl` pour accéder aux éléments de la liste.

■ Langages compilés vs interprétés

Les programmes écrits dans un langage de programmation doivent être **traduits** en langage machine pour être exécutés par le processeur. Il existe deux types de traducteurs : des *interpréteurs* et des *compilateurs*.

Ainsi, selon le type de traduction effectuée, on distingue deux catégories de langages : les **langages compilés** et les **langages interprétés**. Notez par exemple que Python est un langage *interprété*, tandis que C est un langage *compilé*.

Regardez la vidéo suivante qui explique la différence entre les langages interprétés et compilés :

Source : https://youtu.be/4lXp_89c3RU

On a aussi demandé à ChatGPT d'expliquer cette différence. Voici sa réponse :



Peux-tu expliquer la différence entre un langage interprété et un langage compilé ? L'explication doit être compréhensible pour des débutants en informatique.



Bien sûr, je serai heureux de vous expliquer la différence entre un langage interprété et un langage compilé.

Les langages informatiques sont des moyens de communiquer avec l'ordinateur pour lui donner des instructions. Les deux types de langages les plus courants sont les langages interprétés et les langages compilés.

Un langage compilé est un langage informatique où le code source est converti en code machine (binaire) avant l'exécution. Cela signifie que le code source est analysé en entier par un programme appelé un compilateur, qui le

traduit en code binaire qui peut être exécuté directement par l'ordinateur. Le code binaire produit par le compilateur peut ensuite être exécuté plusieurs fois sans avoir besoin de recompiler le code source à chaque fois. Les exemples de langages compilés sont le C, le C++, le Java et le Swift.

En revanche, un langage interprété est un langage informatique où le code source est exécuté ligne par ligne à mesure que l'on tape les commandes dans un interpréteur. L'interpréteur est un programme qui traduit le code source en code machine, ligne par ligne, chaque fois que le code est exécuté. Cela signifie que le code source est analysé et traduit à chaque fois qu'il est exécuté, ce qui ralentit l'exécution du code. Les exemples de langages interprétés sont Python, Ruby et JavaScript.

En somme, les langages compilés sont généralement plus rapides que les langages interprétés, mais nécessitent une étape de compilation avant l'exécution. Les langages interprétés sont plus faciles à apprendre et à déboguer, mais sont souvent plus lents que les langages compilés en raison de la nécessité de traduire le code source à chaque exécution.



Lien vers la conversation d'origine : <https://shareg.pt/ZENzHL4>.

■ Bilan

- Il existe énormément de langages de programmation, chacun ayant ses propres caractéristiques, syntaxes et approches qui le distinguent des autres.
- Néanmoins, tous les langages s'appuient sur un **corpus de constructions élémentaires** sur lequel on s'appuie pour écrire des programmes : séquences, affectation, conditionnelles, boucles bornées, boucles non bornées, appels de fonction...
- Les constructions élémentaires sont donc les mêmes mais la syntaxe varie d'un langage à l'autre.
- Il existe deux catégories principales de langages : les langages **interprétés** et les langages **compilés** : cela dépend de la façon dont la traduction se fait en langage machine. Un *interpréteur* va traduire un programme ligne par ligne et on pourra le modifier directement mais il doit être traduit à chaque exécution, tandis qu'un *compilateur* va traduire le programme dans son entièreté (= le compiler) avant de le transmettre au processeur mais il n'est plus nécessaire de le traduire à chaque exécution ultérieure sauf s'il faut le modifier.
- Nous parlerons en Terminale de différents paradigmes de programmation (= manière de voir, de programmer) : les paradigmes *impératif* (celui que vous connaissez), *objet* et *fonctionnel*.

Références :

- Document ressource Eduscol : [Diversité et unité des langages de programmation](#)

