Les langages de programmation

Pourquoi ne développerait-on pas un langage de programmation universel ?

Mémoire 2022-2023

MPUTU Yoann – 2eB

Directeur de mémoire : OBERTIN Pierre

Table of Contents

Introduction

Les langages de programmation sont un élément fondamental de l'informatique et de la technologie, servant de principal moyen de communication entre les humains et les ordinateurs. Un langage de programmation est un ensemble d'instructions qui peuvent être utilisées pour créer des programmes informatiques, des applications et des logiciels. Il fournit aux programmeurs un moyen d'exprimer leurs idées et leurs algorithmes d'une manière qu'un ordinateur peut comprendre et exécuter.

L'importance des langages de programmation ne peut être surestimée. Ils nous permettent de développer des logiciels et des systèmes qui pilotent le monde moderne, des systèmes d'exploitation qui s'exécutent sur nos ordinateurs personnels et nos smartphones aux algorithmes qui alimentent les moteurs de recherche et les plateformes de médias sociaux. Sans langages de programmation, il serait impossible de créer la vaste gamme d'applications, logicielles et de services que nous utilisons tous les jours.

Les langages de programmation ont parcouru un long chemin depuis que les premiers programmes informatiques ont été écrits dans les années 1940. Au fil des ans, ils ont évolué et mûri, devenant plus sophistiqués et plus puissants au fil des décennies. Aujourd'hui, des centaines de langages de programmation sont utilisés, chacun avec ses propres forces, faiblesses et cas d'utilisation.

Malgré la vaste gamme de langages de programmation disponibles, ils partagent tous certains éléments communs. Chaque langage a sa propre syntaxe et sa propre sémantique, qui déterminent son écriture et son exécution.

L'un des facteurs clés qui distinguent les langages de programmation est leur niveau d'abstraction. Les langages de bas niveau, tels que le langage d'assemblage, fournissent un accès direct aux ressources de l'hardware et requiert des programmeurs d'avoir une compréhension approfondie de l'architecture informatique. Les langages de haut niveau, tels que Python et Java, fournissent des paradigmes de programmation plus abstraits et expressifs qui permettent aux programmeurs d'écrire plus facilement des applications complexes.

Les langages de script, tels que JavaScript et Ruby, sont conçus pour le prototypage rapide et le développement interactif, tandis que les langages spécifiques à un domaine (DSLs) sont adaptés à des domaines spécifiques, tels que le calcul scientifique ou la gestion de bases de données.

Une autre distinction importante est le paradigme ou le style de programmation pris en charge par le langage. Les langages impératifs, tels que C et Java, se concentrent sur la description de la séquence d'opérations qu'un programme doit effectuer. Les langages fonctionnels, tels que Haskell et Lisp, se concentrent sur l'utilisation de fonctions pour effectuer des calculs. Les langages orientés objet (OOP), tels que Python et Ruby, se concentrent sur l'utilisation d'objets et de classes pour organiser le code.

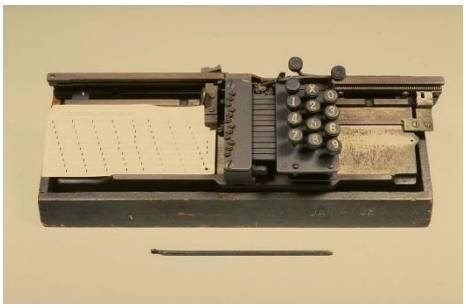
Finalement, il existe une grande diversité de langages de programmation. Pourquoi ne pourrait-on pas crée un langage de programmation universelle, de la même façon que l'anglais est une langue presque universelle ? Ce mémoire examinera l'histoire et l'évolution des langages de programmation, les différents types de langages, la conception et la mise en œuvre d'un langage, ainsi que la popularité des langages. Enfin, une brève perspective sur l'avenir des langages sera présentée, pour conclure avec une réponse à cette question.

Histoire et évolution des langages de programmation

Les langages de programmation ont parcouru un long chemin depuis que les premiers programmes informatiques ont été écrits dans les années 40. Au fil des ans, ils ont évolué et mûri, devenant plus sophistiqué et puissant à chaque décennie qui passe. L'évolution des langages de programmation remonte à plusieurs étapes clés de l'histoire de l'informatique.

Différentes générations de langages de programmation

La première génération de langages de programmation (1GL), connue sous le nom de langage machine, consistait en des instructions de bas niveau qui ont été écrites en code binaire, qui était le seul langage que les ordinateurs pouvaient comprendre. La programmation dans le langage de la machine était un processus fastidieux et sujet aux erreurs, obligeant les programmeurs à entrer manuellement chaque instruction à l'aide de commutateurs ou de cartes de punch.



1 Appareil de perforation de cartes.

La deuxième génération de langages de programmation (2GL), connue sous le nom de langage d'assemblage, a été développée pour rendre la programmation plus accessible et efficace. Le langage d'assemblage a utilisé des codes mnémoniques (ensemble de quelques lettres, souvent trois) pour représenter les instructions du langage de la machine, ce qui facilite l'écriture du code des programmeurs. Cependant, le langage d'assemblage était encore de bas niveau et nécessitait une compréhension approfondie de l'architecture informatique.

```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS    #STACK

                        ****************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

C003 86 13      INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR

                        ****************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH     LDA A  ACIA      GET STATUS
C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC    INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A  ACIA+1    GET CHAR
C019 84 7F              AND A  #$7F      MASK PARITY
C01B 7E C0 79           JMP    OUTCH     ECHO & RTS
```

La troisième génération de langages de programmation (3GL), qui a émergé dans les années 50 et 60, représentait un saut majeur en avant dans l'abstraction et l'expressivité de la programmation. Ces langages, tels que Fortran, Cobol et Algol, ont introduit des constructions de haut niveau telles que des instructions conditionnelles, des boucles et des sous-programmes, ce qui a permis aux programmeurs d'écrire plus facilement des programmes complexes.

La quatrième génération de langages de programmation (4GL), qui a émergé dans les années 80 et 90, a introduit des abstractions encore plus puissantes, telles que la programmation orientée objet, qui a permis un code plus modulaire et réutilisable. Cette ère a également vu l'émergence de langages de script tels que Perl et Python, qui ont été conçus pour le prototypage rapide et le développement interactif.

La cinquième génération de langages de programmation, qui a émergé dans les années 90 jusqu'à présent, sont des langages de programmation qui visent à permettre aux ordinateurs de comprendre le langage humain naturel. Ils sont basés sur des techniques de l'intelligence artificielle et ont pour objectif de permettre aux programmeurs de spécifier ce qu'ils veulent faire, plutôt que de spécifier comment le faire. Les langages de cinquième génération incluent Prolog et Lisp.

Toutefois, il y encore la programmation visuelle qui peut être considérée comme une approche différente pour la création de logiciels par rapport aux langages de programmation traditionnels.

Avec la programmation visuelle, les développeurs créent des applications en utilisant une interface graphique plutôt qu'en écrivant du code manuellement.

Les langages de programmation visuelle incluent des outils qui permettent aux développeurs de créer des programmes en assemblant des blocs graphiques qui représentent des instructions de programmation. Les blocs peuvent être déplacés et connectés pour créer des flux de programmes, ce qui facilite la création d'applications pour les débutants.

Bien que la programmation visuelle puisse être considérée comme une évolution de la programmation traditionnelle, elle ne constitue pas une génération distincte de langages de programmation. Au lieu de cela, elle peut être considérée comme une approche alternative pour la création de logiciels.

Étapes clés du développement du langage de programmation

Les langages de programmation ont évolué considérablement au cours des dernières décennies, avec de nombreux étapes clés marquant des tournants majeurs dans leur développement. Voici quelques-uns des étapes les plus importantes de l'histoire du langage de programmation.

1949 : Langage d'assemblage

Le langage d'assemblage a été le premier langage de programmation à fournir un niveau d'abstraction entre le langage machine et les langages de programmation de haut niveau. Le langage d'assemblage a utilisé des codes mnémoniques pour représenter les instructions du langage de la machine, ce qui facilite l'écriture du code des programmeurs.

1957 : Fortran

Fortran a été le premier langage de programmation de haut niveau et a été conçu pour les applications scientifiques et d'ingénierie. Fortran a permis une programmation plus naturelle et lisible par rapport au langage d'assemblage.

1959 : COBOL

COBOL a été conçu pour les applications commerciales et a été le premier langage de programmation à être standardisé. COBOL a introduit de nombreuses fonctionnalités pour l'informatique commerciale, telles que l'arithmétique décimale et le traitement de fichiers.

1960 : Algol

Algol était le premier langage conçu pour la programmation algorithmique et était largement utilisé dans la recherche scientifique. Algol a introduit de nombreuses nouvelles constructions de programmation, telles que les « while » et « for » boucles, qui sont toujours utilisées dans les langages de programmation modernes.

1964 : BASIC

BASIC a été conçu pour que les débutants et les étudiants apprennent la programmation. Il était facile à apprendre et à utiliser et à populariser l'utilisation d'ordinateurs personnels.

1972 : C

C a été créé par Dennis Ritchie à Bell Labs et est devenu l'un des langages de programmation les plus utilisés. C a introduit de nombreuses fonctionnalités qui sont encore utilisées dans les langages de programmation modernes, tels que les structures de données et la structure de contrôle.

1983 : Ada

L'Ada a été conçue par le ministère américain de la défense pour des systèmes critiques de sécurité tels que les aéronefs et les missiles. Ada était la première langue à inclure des fonctionnalités d'ingénierie du logiciel, telles que des packages et des exceptions.

1991 : Python

Python a été conçu pour être facile à apprendre et à lire, en mettant l'accent sur la simplicité et la lisibilité du code. Python est depuis devenu l'un des langages de programmation les plus populaires pour l'informatique scientifique, le développement Web et l'intelligence artificielle.

1995 : Java

Java a été développé par Sun Microsystems et est l'un des langages de programmation les plus populaires utilisés aujourd'hui. La philosophie « Écrire une fois, exécuter n'importe où » de Java et la prise en charge de la programmation orientée objet, en a fait un choix populaire pour la construction d'applications d'entreprise. Il a été conçu pour être sûr et fiable, et il est utilisé pour des applications pour de nombreuses plateformes, comme le web, les appareils mobiles et les ordinateurs de bureau.

2001 : C#

C# est un langage de programmation orienté objet développé par Microsoft, et principalement utilisé pour la création d'applications pour les systèmes Windows et la plateforme .NET. Il offre un ensemble de fonctionnalités avancées, telles que la gestion automatique de la mémoire, la vérification de type statique et la prise en charge des événements. C# est un langage fortement typé qui permet aux développeurs de créer des applications robustes et sécurisées.

2009 : Go

Le langage Go, également connu sous le nom de Golang, est un langage de programmation open-source développé par Google. Il a été conçu pour résoudre les problèmes liés à la complexité, à la lenteur et à l'inefficacité des langages de programmation existants. Il est particulièrement adapté pour le développement de logiciels système.

2010 : Rust

Rust est un langage de programmation open source développé par Mozilla qui se concentre sur la sécurité, la vitesse et la concurrence. Il est conçu pour être performant et permettre aux développeurs de créer des logiciels fiables, efficaces et sûrs, tout en offrant des fonctionnalités avancées telles que la gestion automatique de la mémoire, la programmation parallèle et la sécurité des threads.

2014 : Swift

Swift est un langage de programmation moderne et open source créé par Apple pour le développement d'applications pour ses plateformes, telles que iOS, macOS, watchOS et tvOS. Il est conçu pour être rapide, sûr et facile à utiliser, avec une syntaxe concise et expressive.

Types de langages de programmation

Dans ce chapitre, nous allons explorer les différents types de langages de programmation et leurs caractéristiques distinctives. Les langages de programmation sont utilisés pour développer des applications logicielles, des sites web et des programmes informatiques pour diverses plates-formes.

Nous allons discuter des différences entre les langages de bas niveau, haut niveau, scripts et spécifique à un domaine.

Les langages de bas niveau

Les langages de bas niveau sont proches de la structure physique de l'ordinateur et des instructions qu'il peut exécuter directement. Ils sont généralement plus difficiles à utiliser car ils nécessitent une connaissance approfondie de l'architecture de l'ordinateur et de ses instructions, mais pour cela ils sont plus efficaces en termes de performance. Les exemples de langages de bas niveau sont :

Le langage assembleur : C'est un langage de bas niveau qui permet d'écrire des programmes en utilisant des mnémoniques pour représenter les instructions de l'ordinateur. Il est spécifique à chaque architecture de processeur et est généralement utilisé pour écrire des programmes de systèmes ou des programmes qui nécessitent des performances élevées.

Le langage machine : C'est un langage de bas niveau qui consiste en suites de bits (0 et 1) qui représentent directement les instructions d'un ordinateur. Il est spécifique à chaque architecture de processeur et il est généralement utilisé pour déboguer des programmes écrits en assembleur ou pour écrire des programmes qui nécessitent des performances maximales.

Langages de bas niveau de haut niveau : Il existe des langages de programmation qui sont considérés comme des langages de bas niveau mais qui ont une syntaxe plus proche des langages de haut niveau. C'est le cas par exemple du C qui est souvent considéré comme un langage de bas niveau de haut, car il est proche de l'architecture de l'ordinateur et de son fonctionnement mais il est plus facile à utiliser que l'assembleur ou le langage machine.

Les langages de haut niveau

Les langages de haut niveau, d'autre part, sont conçus pour être plus proche de la façon dont les humains pensent et communiquent. Ils ont une syntaxe plus facile à lire et à comprendre et utilisent des mots et des expressions familiers. Ainsi, ils permettent aux programmeurs d'écrire du code de manière plus intuitive et abstraite. Il permet de s'abstraire de la complexité des détails techniques du fonctionnement de l'ordinateur et de se concentrer sur la résolution des problèmes liés à l'application à développer. Certains langages de programmation de haut niveau populaires incluent Python, Java, Ruby et JavaScript. Ces langages ont une syntaxe facile à lire et à écrire, ce qui permet aux programmeurs de se concentrer sur la logique de leur code plutôt que sur la syntaxe du langage lui-même.

Dans l'ensemble, les langages de programmation de haut niveau ont révolutionné la façon dont nous écrivons et développons des logiciels. En rendant la programmation plus accessible et intuitive, ils ont ouvert de nouvelles possibilités pour les personnes de tous horizons de créer des solutions logicielles innovantes. Il existe de nombreux autres langages de haut niveau, chacun ayant ses propres caractéristiques et utilisations spécifiques.

Langages de scripts

Les langages de script sont des langages de programmation de haut niveau qui sont généralement utilisés pour automatiser des tâches, traiter des données et créer des applications de petite à moyenne taille. Contrairement aux langages compilés tels que C ou Java, qui nécessitent que le code source soit compilé en code machine avant l'exécution, les langages de script sont interprétés au moment de l'exécution. Cela signifie que le code est exécuté ligne par ligne, au fur et à mesure qu'il est rencontré, par un logiciel appelé interpréteur.

Les langages de script sont souvent utilisés pour écrire des scripts qui automatisent les tâches répétitives, telles que les sauvegardes de fichiers, l'administration système ou les mises à jour de pages Web. Ils sont également utilisés pour créer des applications de petite à moyenne taille, telles que des applications Web, des jeux ou des simulations scientifiques.

Certains langages de programmation de scripts populaires incluent Python, Ruby, Perl, JavaScript et PHP. Ces langages sont souvent choisis pour leur simplicité, leur flexibilité et leur facilité d'utilisation. Ils ont généralement moins de lignes de code que les langages compilés et nécessitent moins d'installation et de configuration.

Dans l'ensemble, les langages de programmation de scripts sont un outil polyvalent et puissant pour automatiser les tâches, créer des applications et résoudre des problèmes dans un large éventail de domaines.

Langages spécifiques à un domaine

Les langages spécifiques à un domaine (DSL pour Domain-Specific Language en anglais) sont des langages de programmation qui ont été conçus pour résoudre des problèmes dans un domaine spécifique, tels que la finance, la biologie, l'ingénierie, la robotique, etc.

Contrairement aux langages de programmation généraux, qui peuvent être utilisés pour résoudre des problèmes dans de nombreux domaines, les DSL sont conçus pour être spécifiques à un domaine particulier, en se concentrant sur les concepts et les termes propres à ce domaine. Cela permet aux utilisateurs de communiquer et de travailler efficacement avec le code dans le cadre de leur domaine d'application.

Les DSL peuvent être créés de différentes manières. Par exemple, ils peuvent être créés en utilisant des extensions de langages de programmation existants pour ajouter des fonctionnalités spécifiques au domaine, ou en créant un tout nouveau langage de programmation spécifiquement pour ce domaine.

Les DSL peuvent avoir des syntaxes et des structures très différentes des langages de programmation généraux, car ils sont conçus pour refléter les concepts propres au domaine. Cela signifie que les utilisateurs doivent souvent apprendre une nouvelle syntaxe et une nouvelle grammaire pour utiliser un DSL.

Les avantages des DSL sont nombreux. Ils permettent aux programmeurs de travailler plus efficacement dans leur domaine, en utilisant un langage de programmation qui reflète les concepts et les termes propres à leur domaine. Cela peut réduire les erreurs et les problèmes de communication, et permettre une meilleure compréhension du code source. Les DSL peuvent également simplifier la création de programmes pour les utilisateurs sans compétences en programmation, car ils peuvent être conçus pour être plus accessibles et plus faciles à comprendre.

Conception et mise en œuvre d'un langage

Syntaxe et sémantique

La syntaxe et la sémantique sont deux aspects fondamentaux des langages de programmation qui définissent respectivement la structure et la signification du code.

La syntaxe fait référence à l'ensemble de règles et de conventions qui dictent comment les programmes doivent être écrits dans un langage particulier. Ces règles définissent la structure et l'agencement des mots clés, opérateurs, identificateurs et autres éléments qui composent un programme. La syntaxe est importante car elle garantit que les programmes sont écrits de manière cohérente et claire, ce qui facilite la compréhension du code par les humains et les ordinateurs. Lorsqu'un programmeur écrit du code qui ne respecte pas la syntaxe du langage, le compilateur ou l'interpréteur génère généralement une erreur de syntaxe, empêchant l'exécution du programme.

La sémantique, quant à elle, fait référence à la signification et au comportement des constructions dans un langage de programmation. La sémantique définit comment un programme doit se comporter lorsqu'il est exécuté, en fonction des éléments syntaxiques utilisés dans le code. Par exemple, la sémantique d'un langage de programmation particulier peut dicter comment les variables sont stockées en mémoire, comment les boucles et les instructions conditionnelles doivent être exécutées, ou comment les fonctions sont appelées et leurs résultats renvoyés.

Pour comprendre la différence entre la syntaxe et la sémantique, considérons l'exemple suivant : En français, la phrase « Les idées vertes incolores dorment furieusement » est grammaticalement correcte et suit les règles de la syntaxe, mais elle n'a aucun sens, et la phrase est sémantiquement incorrecte. De même, dans les langages de programmation, un programme peut avoir une syntaxe correcte mais peut ne pas être sémantiquement correct s'il n'exécute pas l'opération souhaitée.

En résumé, la syntaxe et la sémantique sont des concepts essentiels dans les langages de programmation qui, ensemble, dictent comment le code doit être écrit (syntaxe) et ce qu'il signifie lorsqu'il est exécuté (sémantique). Une compréhension approfondie des deux aspects est nécessaire pour que les programmeurs puissent écrire un code efficace, précis et facilement maintenable.

Paradigmes d'un langage

Un paradigme de programmation est un ensemble de principes fondamentaux, concepts et styles de programmation qui définit une approche globale pour résoudre des problèmes de programmation. Il s'agit d'une façon de penser la conception et la mise en œuvre d'un programme informatique. Les différents paradigmes de programmation définissent donc, des manières différentes d'aborder la résolution de problèmes de programmation.

Comprendre les différents paradigmes de programmation est important pour les développeurs, car cela leur permet de choisir la meilleure approche pour résoudre un problème spécifique. De plus, la connaissance de plusieurs paradigmes de programmation peut aider les développeurs à écrire des programmes plus efficaces et plus lisibles, tout en améliorant leur capacité à comprendre et à maintenir le code existant.

Programmation impérative

La programmation impérative est un paradigme de programmation qui se concentre sur la manière dont un programme doit être exécuté pour produire un résultat souhaité. Elle implique la définition d'un ensemble d'instructions ou de commandes qui doivent être exécutées séquentiellement pour effectuer une tâche spécifique.

En programmation impérative, le programmeur décrit l'algorithme sous forme de séquences d'instructions qui modifient l'état du programme. Ces instructions sont exécutées dans un ordre précis, en utilisant des structures de contrôle telles que les boucles et les instructions conditionnelles pour contrôler l'exécution du programme.

Des exemples de langages de programmation impératifs comprennent C, Java, Python et Ruby. Ces langages fournissent au programmeur la capacité de spécifier la manière dont le programme doit s'exécuter en utilisant des instructions telles que "if", "while" et "for". La programmation impérative convient bien à la résolution de problèmes qui nécessitent un contrôle précis de l'ordre dans lequel les opérations sont effectuées et de la manipulation de l'état du programme.

Dans l'ensemble, la programmation impérative est un paradigme puissant qui a été utilisé pour développer de nombreuses applications logicielles et systèmes que nous utilisons aujourd'hui.

Programmation déclarative

La programmation déclarative consiste à décrire ce que l'on veut obtenir plutôt que de décrire les étapes à suivre pour y arriver. Au lieu de donner des instructions détaillées sur la façon dont le programme doit fonctionner, le programmeur spécifie simplement les résultats souhaités, et le programme se charge de déterminer la meilleure façon d'y parvenir.

En d'autres termes, plutôt que de dire au programme comment faire quelque chose, on lui dit ce qu'il doit faire, et il se débrouille pour trouver la solution. Ce qui rend la programmation déclarative intéressante, c'est que cela permet souvent de simplifier le code et de le rendre plus facile à lire et à comprendre.

Il existe plusieurs sous-paradigmes de la programmation déclarative, tels que la programmation logique, la programmation fonctionnelle et la programmation par contraintes.

Programmation logique

La programmation logique est un paradigme de programmation qui se concentre sur la manipulation de relations logiques entre les données plutôt que sur la manipulation de données elles-mêmes. Elle repose sur la logique mathématique et la théorie des ensembles pour représenter les connaissances et les relations entre les données.

Dans la programmation logique, le programmeur spécifie les faits et les règles logiques qui décrivent les relations entre les données. Le programme utilise ensuite ces faits et ces règles pour inférer de nouvelles informations et répondre à des questions sur les données.

Les programmes logiques sont souvent écrits dans un langage de programmation dédié tel que Prolog. Le programmeur spécifie les faits et les règles logiques en utilisant une syntaxe formelle appelée "règles de Horn", qui est basée sur la logique des prédicats. Les programmes logiques peuvent être utilisés pour résoudre des problèmes de logique, de planification, de recherche, de traitement de langage naturel, de reconnaissance de formes, et bien d'autres.

Un exemple de programme logique simple pourrait consister à trouver tous les chemins possibles entre deux villes sur une carte. Dans ce cas, le programmeur spécifierait les villes et les connexions entre elles sous forme de faits, et utiliserait des règles logiques pour trouver toutes les combinaisons de connexions qui relient les deux villes.

Programmation fonctionnelle

La programmation fonctionnelle est un paradigme de programmation qui se concentre sur l'utilisation de fonctions pour représenter et manipuler les données. Dans ce paradigme, les fonctions sont considérées comme des entités de première classe, c'est-à-dire qu'elles peuvent être passées en paramètre à d'autres fonctions, retournées comme résultat d'une fonction ou stockées dans des variables.

La programmation fonctionnelle est basée sur les mathématiques et la théorie des fonctions. Les programmes fonctionnels sont souvent écrits dans des langages dédiés tels que Haskell, Lisp ou Scala.

Dans la programmation fonctionnelle, les programmes sont construits à partir de fonctions qui prennent des arguments en entrée et retournent des résultats en sortie, sans modifier l'état global du programme ou de ses variables. Les fonctions sont souvent composées pour former des programmes plus complexes, en utilisant des techniques telles que l'évaluation paresseuse et la récursivité.

Un aspect important de la programmation fonctionnelle est l'immutabilité des données. Les données sont considérées comme immuables, c'est-à-dire qu'une fois créées, elles ne peuvent pas être modifiées. Au lieu de cela, les fonctions créent de nouvelles données à partir des anciennes en utilisant des techniques telles que la transformation, la filtration et l'agrégation.

La programmation fonctionnelle est souvent utilisée pour résoudre des problèmes de traitement de données, de calcul scientifique, de traitement de signaux, d'intelligence artificielle et de traitement de langage naturel. Elle est également de plus en plus utilisée dans le développement de logiciels de haut niveau, tels que les applications web et mobiles.

En somme, la programmation fonctionnelle est un paradigme de programmation basé sur les fonctions, les mathématiques et l'immutabilité des données.

Programmation par contrainte

La programmation par contraintes est un paradigme de programmation qui se concentre sur la spécification de contraintes sur les valeurs possibles d'un ensemble de variables, et sur la recherche d'une solution satisfaisant toutes les contraintes.

Dans la programmation par contraintes, le programmeur spécifie les variables, les domaines de valeurs possibles pour chaque variable, ainsi que les contraintes qui limitent les combinaisons de valeurs que les variables peuvent prendre. Le programme utilise ensuite un solveur de contraintes pour chercher une solution satisfaisant toutes les contraintes.

Les programmes par contraintes sont souvent utilisés pour résoudre des problèmes de planification, d'ordonnancement, d'affectation, de conception de circuits, d'optimisation et d'autres problèmes qui impliquent des contraintes complexes. Ils sont souvent utilisés dans les applications d'intelligence artificielle, de recherche opérationnelle et de planification.

Un exemple de programme par contraintes serait de trouver les valeurs de deux variables x et y telles que $x + y = 5$ et $x \cdot y = 6$. Dans ce cas, le programmeur spécifierait les variables x et y, les domaines de valeurs possibles pour x et y (par exemple, $x \in \{1; 2; 3; 4; 5\}$ et $y \in \{1; 2; 3; 4; 5\}$, et les contraintes $x + y = 5$ et $x \cdot y = 6$. Le solveur de contraintes chercherait ensuite une solution qui satisfait à la fois la contrainte d'addition et la contrainte de multiplication.

En somme, la programmation par contraintes est un paradigme de programmation qui se concentre sur la spécification de contraintes sur les valeurs possibles d'un ensemble de variables, et sur la recherche d'une solution satisfaisant toutes les contraintes. Ce paradigme est souvent utilisé pour résoudre des problèmes de planification, d'ordonnancement, d'affectation, de conception de circuits et d'optimisation.

Ainsi, la programmation déclarative est souvent utilisée pour résoudre des problèmes complexes qui impliquent la manipulation de données structurées. Elle est utilisée dans des domaines tels que l'intelligence artificielle, les systèmes de gestion de bases de données, l'analyse de données, la modélisation de systèmes et bien d'autres.

En somme, la programmation déclarative est un paradigme de programmation qui se concentre sur la description de ce que le programme doit faire, plutôt que sur la façon dont le programme doit le faire. Ce paradigme peut être utilisé pour résoudre un large éventail de problèmes de programmation, en particulier ceux qui impliquent la manipulation de données structurées.

Programmation orienté objet

La programmation orientée objet (POO) est un paradigme de programmation qui utilise la notion d'objet pour structurer le code. Dans ce paradigme, les objets sont des entités qui encapsulent des données et des méthodes (ou fonctions) qui agissent sur ces données. Les objets peuvent interagir les uns avec les autres en utilisant des méthodes, qui sont des fonctions définies dans une classe.

Les principaux concepts en POO sont :

La classe : qui définit les propriétés et les méthodes associées à un objet.

L'objet : qui est une instance d'une classe.

L'héritage simple respectivement multiple : qui permet à une classe d'hériter des propriétés et méthodes d'une respectivement plusieurs super-classes.

L'encapsulation : Les données et les méthodes qui agissent sur ces données sont regroupées en un seul objet, ce qui permet de limiter l'accès aux données et de les protéger contre les modifications non autorisées.

L'abstraction : qui permet de se concentrer sur les propriétés et les comportements essentiels d'un objet en cachant les détails d'implémentation.

Le polymorphisme : permet à un objet de prendre plusieurs formes, c'est-à-dire d'être traité comme plusieurs types d'objets différents.

La POO est souvent utilisée pour modéliser des systèmes complexes et pour résoudre des problèmes qui impliquent la manipulation de données complexes. Elle est utilisée dans de nombreux domaines, tels que le développement de jeux vidéo, la conception de logiciels, les applications de traitement d'image, la modélisation de systèmes, la robotique, et bien d'autres.

En résumé, la programmation orientée objet est un paradigme de programmation qui se concentre sur la représentation de concepts du monde réel en tant qu'objets. Ce paradigme utilise des classes pour définir les propriétés et les comportements d'un objet, et utilise l'encapsulation, l'héritage, le polymorphisme et l'abstraction pour organiser et structurer le code de manière efficace.

Compilateur vs interpréteur

La machine ne comprend pas le langage en soi. Pour la machine ce ne sont que des mots dans le vide. Pour cela il est nécessaire d'avoir un traducteur. Ce pont qui va lier l'humain à la machine peut être soi un compilateur ou un interprète.

Compilateur

Un compilateur est un programme informatique qui traduit le code source écrit dans un langage de programmation en un code objet, qui peut être exécuté par un ordinateur par la suite. Le processus de compilation implique plusieurs étapes, notamment l'analyse lexicale, l'analyse syntaxique, l'analyse sémantique, la génération de code, et l'optimisation du code.

L'analyse lexicale consiste à scanner le code source pour identifier les jetons (tokens) qui sont les éléments de base du langage de programmation, tels que les mots-clés, les opérateurs et les identificateurs. L'analyse syntaxique consiste à analyser les jetons pour déterminer la structure grammaticale du code source et créer un arbre de syntaxe abstrait. L'analyse sémantique consiste à vérifier que le code source respecte les règles sémantiques du langage de programmation, telles que les types de données et les déclarations de variables.

La génération de code consiste à traduire l'arbre de syntaxe abstrait en code objet, qui est spécifique à l'architecture matérielle de l'ordinateur cible. Le code généré est généralement stocké dans un fichier objet qui peut être lié à d'autres fichiers objets pour former un programme exécutable. L'optimisation du code consiste à apporter des modifications au code généré pour améliorer ses performances, telles que la réduction du nombre d'opérations ou l'utilisation de registres de processeur plus rapides.

Les compilateurs sont souvent utilisés pour compiler des programmes dans des langages de programmation tels que C, C++, Java, Python, Ruby, et bien d'autres. Ils permettent aux programmeurs d'écrire des programmes dans un langage de haut niveau, plus facile à comprendre et à écrire, et de les exécuter sur des ordinateurs sans avoir à écrire le code directement en langage machine.

En résumé, un compilateur est un programme informatique qui traduit le code source en un code objet qui peut être exécuté directement par un ordinateur. Ainsi contrairement aux interpréteurs, qui exécutent les instructions ligne par ligne, grâce aux compilateurs un programme est exécuté plus rapidement et le code source est moins accessible.

Interpréteur

Un interpréteur est un programme informatique qui lit et exécute le code source d'un programme ligne par ligne, sans le compiler en un code objet exécutable à l'avance. L'interpréteur analyse et exécute chaque instruction du code source au fur et à mesure qu'elle est rencontrée.

Contrairement à un compilateur qui traduit le code source en code objet, l'interpréteur traduit le code source en code machine à l'exécution. Cela signifie que le code source n'est pas compilé en avance et qu'il n'y a pas de fichier objet exécutable créé. Le code est plutôt exécuté directement à partir du code source.

Les programmes interprétés sont souvent plus lents que les programmes compilés, car chaque instruction doit être traduite à l'exécution. Cependant, l'utilisation d'un interpréteur a l'avantage de rendre le débogage plus facile, car ils signalent les erreurs dès qu'ils les rencontrent, et de permettre une interactivité directe avec le code source, où les instructions peuvent être saisies et exécutées à la volée.

Les langages de programmation les plus couramment utilisés avec des interpréteurs sont les langages de script, tels que Python, Ruby, Perl, et JavaScript. Les interpréteurs sont également utilisés dans les environnements de développement intégrés (IDE) pour permettre l'exécution interactive du code pendant le processus de développement.

Optimisation des performances

La performance d'un langage de programmation dépend de nombreux facteurs, tels que la vitesse d'exécution, la consommation de mémoire, la facilité d'utilisation et la maintenabilité du code. Voici quelques-uns des facteurs qui peuvent influencer la performance d'un langage de programmation :

1. Compilation vs Interprétation : Les langages compilés, comme C, C++ ou Rust, sont généralement plus rapides que les langages interprétés, comme Python, Ruby ou JavaScript. Cela est dû au fait que les langages compilés sont transformés en code machine optimisé avant l'exécution, tandis que les langages interprétés sont traduits en code machine à la volée.
2. Gestion de la mémoire : Les langages qui nécessitent une gestion manuelle de la mémoire, comme C ou C++, peuvent être plus performants car le développeur a un contrôle total sur l'allocation et la libération de la mémoire. Cependant, cela peut rendre le code plus difficile à écrire et à maintenir. Les langages avec une gestion de la mémoire automatique, comme Java ou Python, peuvent être moins performants car ils nécessitent des opérations supplémentaires pour allouer et libérer de la mémoire.
3. Typage : Les langages typés statiquement, comme Java ou C++, peuvent être plus performants car les erreurs de typage sont détectées à la compilation plutôt qu'à l'exécution. Les langages typés dynamiquement, comme Python ou Ruby, peuvent être moins performants car ils nécessitent des opérations supplémentaires pour vérifier le type des objets à l'exécution.
4. Primitives de bas niveau : Les langages qui offrent un accès de bas niveau au matériel, comme C ou Rust, peuvent être plus performants pour les applications nécessitant une performance maximale. Cependant, cela peut rendre le code plus difficile à écrire, à comprendre et à optimiser puisqu'il faut une connaissance approfondie de l'architecture de l'ordinateur et de ses instructions.
5. Bibliothèques : Les langages avec des bibliothèques bien conçues et performantes peuvent être plus performants car ils permettent aux développeurs d'utiliser des fonctions et des algorithmes optimisés sans avoir à les écrire eux-mêmes. Cependant, cela dépend également de la qualité et de la compatibilité des bibliothèques avec le reste du code.

En fin de compte, le choix d'un langage de programmation dépend des besoins spécifiques d'une application donnée, ainsi que des compétences et des préférences du développeur.

Les langages de programmation les plus populaires aujourd'hui

Dans ce chapitre, nous allons voir la popularité de différents langages de programmation. Nous allons discuter de l'indice TIOBE, du Stack Overflow Survey et finalement comparer les principaux langages de programmation.

Indice TIOBE

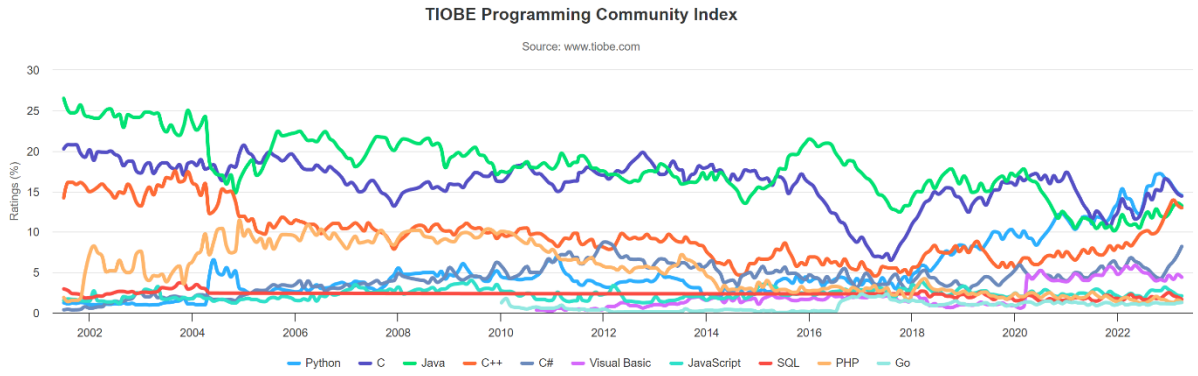
Bien sûr, l'indice TIOBE est un classement qui mesure la popularité des langages de programmation. Il est mis à jour une fois par mois par la société TIOBE Software BV, basée aux Pays-Bas.

L'indice TIOBE est calculé en se basant sur le nombre de résultats de recherche sur les moteurs de recherche tels que Google, Bing, Yahoo, Wikipédia et autres sites web. Plus précisément, l'indice mesure le pourcentage de recherche d'un langage de programmation par rapport à l'ensemble des requêtes de recherche de programmation. Ainsi, plus un langage de programmation est recherché sur le web, plus son indice TIOBE est élevé.

Cependant, il est important de noter que l'indice TIOBE ne mesure pas la qualité ou la performance des langages de programmation, mais plutôt leur popularité. Par conséquent, un langage de programmation peut avoir un indice TIOBE élevé même s'il est considéré comme obsolète ou moins performant que d'autres langages de programmation.

L'indice TIOBE est souvent utilisé comme un indicateur de tendances dans l'industrie du développement de logiciels et de la demande de compétences en programmation.

Ci-dessous se trouve le graphique de l'indice TIOBE allant du 30 juin 2001 au 1er avril 2023. Il indique que C et Java ont constamment rivalisé pour la première place. Cependant, C++ et Python ont gagné en popularité et maintenant ces quatre langages de programmation se disputent le podium.



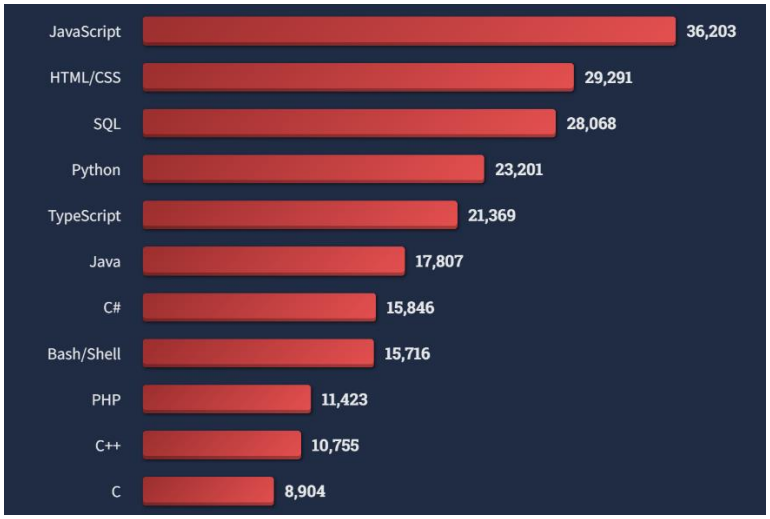
Stack Overflow Survey

Stack Overflow Survey est une enquête annuelle menée par la plateforme de développement Stack Overflow, qui est l'un des sites web les plus populaires pour les développeurs du monde entier. L'enquête est conçue pour recueillir des données sur les développeurs et leurs pratiques, leurs préférences de langage de programmation, leur expérience professionnelle et d'autres aspects de leur vie professionnelle.

Chaque année, l'enquête est ouverte aux développeurs du monde entier, qui peuvent y répondre en ligne. Les questions posées couvrent un large éventail de sujets, tels que les langages de programmation préférés, les systèmes d'exploitation, les outils et les technologies, les pratiques de développement, la formation et les compétences, les salaires et les avantages sociaux, et d'autres sujets liés au développement de logiciels.

Les résultats de l'enquête sont publiés sous forme de rapport annuel et sont largement utilisés par les développeurs, les entreprises et l'industrie pour comprendre les tendances et les préférences des développeurs du monde entier. Les données de l'enquête peuvent être utilisées pour prendre des décisions éclairées sur les technologies à utiliser, les stratégies de recrutement, les politiques de rémunération et d'autres aspects de l'industrie du développement de logiciels.

Les résultats du sondage Stack Overflow Survey de l'année 2022 sur les technologies les plus populaires auprès des développeurs professionnels sont présentés ci-dessous. Les participants ont été interrogés sur cette question : « Which programming, scripting, and markup languages have you done extensive development work in over the past year, and which do you want to work in over the next year ? (If you both worked with the language and want to continue to do so, please check both boxes in that row.) » Un total de 53 421 développeurs professionnels ont répondu à cette question. Ainsi comme on peut le voir ci-dessous JavaScript serait le langage le plus populaire.



Comparaison des principaux langages

Les langages de programmation sont des outils essentiels pour les développeurs de logiciels dans la création d'applications et de systèmes informatiques. Cependant, avec tant de langages différents disponibles, il peut être difficile pour les développeurs de choisir le langage le mieux adapté à leur projet spécifique. Dans ce chapitre, je vais examiner les langages de programmation les plus populaires et les plus utilisés aujourd'hui.

Python

Python est un langage de programmation interprété, orienté objet et de haut niveau. C'est un langage très polyvalent, mais est toutefois souvent utilisé pour la science des données, l'apprentissage automatique, l'analyse de données et l'automatisation des tâches. Python est réputé pour sa syntaxe simple et facile à lire, qui le rend facile à apprendre pour les débutants. De plus, il est disponible sur toutes les plateformes et dispose d'une grande communauté de développeurs.

Java

Java est un langage de programmation orienté objet populaire pour les applications de bureau et mobiles. Il est également largement utilisé dans le développement d'applications Web. Java est connu pour sa portabilité, sa sécurité et sa fiabilité. Il est également très populaire pour les applications d'entreprise, les applications de traitement de données et les applications de jeu.

C

C est un langage de programmation bas niveau qui est souvent utilisé pour les systèmes d'exploitation, les pilotes de périphériques, les logiciels intégrés et les jeux. C est considéré comme l'un des langages de programmation les plus rapides et les plus efficaces, car il est très proche de la machine. Cependant, il peut être difficile à apprendre pour les débutants en raison de sa complexité.

JavaScript

JavaScript est un langage de script orienté objet populaire pour les applications Web. Il est souvent utilisé pour la création de pages Web interactives et la manipulation de données côté client. JavaScript est très flexible et facile à apprendre pour les débutants, car il ne nécessite pas l'installation d'un environnement de développement complet.

Ruby

Ruby est un langage de programmation interprété, orienté objet et dynamique. Il est souvent utilisé pour le développement Web, en particulier pour les sites Web de commerce électronique et les applications Web complexes. Ruby est réputé pour sa syntaxe lisible et concise, ainsi que pour sa communauté de développeurs active.

C++

C++ est un langage de programmation orienté objet et bas niveau. Il est souvent utilisé pour les applications de jeu, les applications de traitement d'images et les logiciels intégrés. C++ est considéré comme l'un des langages de programmation les plus rapides et les plus efficaces, mais il peut être difficile à apprendre pour les débutants.

PHP

PHP est un langage de script populaire pour le développement Web. Il est souvent utilisé pour les applications Web dynamiques et les sites Web de commerce électronique. PHP est facile à apprendre pour les débutants, car il peut être incorporé dans des pages HTML. Il est également disponible sur toutes les plateformes et dispose d'une grande communauté de développeurs.

Swift

Swift est un langage de programmation orienté objet développé par Apple. Il est principalement utilisé pour le développement d'applications iOS, macOS et watchOS. Swift est un langage relativement nouveau, mais il est rapidement devenu populaire car il est plus facile à lire et à écrire qu'Objective-C, le langage de programmation précédemment utilisé pour le développement d'applications Apple.

TypeScript

TypeScript est un langage de programmation open-source développé par Microsoft. Il est souvent utilisé pour les applications Web et les projets de grande envergure. TypeScript est une extension de JavaScript qui ajoute des fonctionnalités de typage statique et de programmation orientée objet. Il permet une meilleure maintenabilité et une plus grande sécurité du code.

Go

Go est un langage de programmation open-source développé par Google. Il est souvent utilisé pour les applications Web, les outils de ligne de commande et les logiciels système. Go est réputé pour sa rapidité, sa simplicité et sa concision. Il est facile à apprendre pour les débutants et permet une programmation parallèle efficace.

Kotlin

Kotlin est un langage de programmation orienté objet développé par JetBrains. Il est principalement utilisé pour le développement d'applications Android, mais peut également être utilisé pour les applications Web et les applications de bureau. Kotlin est réputé pour sa sécurité, sa concision et sa compatibilité avec Java.

Rust

Rust est un langage de programmation open-source développé par Mozilla. Il est souvent utilisé pour les logiciels système, les applications de bas niveau et les projets de sécurité. Rust est réputé pour sa sécurité, sa rapidité et sa stabilité. Il permet également une programmation parallèle efficace et une gestion de la mémoire sans faille.

C#

C# est un langage de programmation orienté objet développé par Microsoft. Il est souvent utilisé pour les applications Windows, les applications Web, les jeux vidéo et les applications mobiles. C# est similaire à Java et offre une syntaxe facile à comprendre pour les développeurs qui ont déjà travaillé avec d'autres langages orientés objet. Il est réputé pour sa sécurité, sa stabilité, sa performance et sa capacité à se connecter facilement avec d'autres technologies Microsoft. C# est également utilisé pour les projets .NET et Unity, ce qui en fait un choix populaire pour les développeurs de jeux vidéo.

En résumé, chaque langage de programmation a ses avantages et son domaine spécifique dans lequel il excelle. Ainsi il n'y a pas réellement un langage mieux qu'un autre. Toutefois l'on peut reconnaitre pourquoi certains langages sont plus populaires qu'un autre. Ainsi JavaScript est un langage très populaire car il est non seulement facile à apprendre, mais aussi essentiel pour la création de pages Web. Étant donné que l'Internet est omniprésent dans notre vie quotidienne, beaucoup de développeurs travaillent avec ce langage. Les langages complémentaires HTML, CSS et SQL suivent naturellement JavaScript pour le développement de pages Web. Ensuite, Python occupe la quatrième place du classement de la Stack Overflow Survey en raison de sa facilité d'apprentissage et de sa polyvalence. Sa popularité a rapidement augmenté, comme en témoigne l'indice TIOBE.

Avenir des langages de programmation

À mesure que la technologie continue de progresser, plusieurs tendances émergentes influencent le développement des langages de programmation. Ces tendances comprennent :

Apprentissage automatique et intelligence artificielle (IA) :

L'apprentissage automatique et l'IA sont des domaines en évolution rapide avec des implications significatives pour les langages de programmation. Les langages de programmation sont développés ou améliorés afin de fournir un meilleur support pour des tâches telles que l'analyse de données, la modélisation prédictive et les réseaux neuronaux.

Big Data et science des données :

La prolifération des données dans différents domaines a conduit à la nécessité de langages de programmation capables de gérer et d'analyser des ensembles de données massifs. Des langages tels que R et Python, ainsi que des frameworks tels qu'Apache Spark, sont largement utilisés en science des données et en traitement des big data. Ces langages offrent de puissantes bibliothèques et outils pour la manipulation des données, la visualisation et l'analyse statistique.

Internet of Things (IoT) :

La croissance des appareils de l'IoT a créé une demande de langages de programmation capables de gérer efficacement les systèmes embarqués et les réseaux de capteurs. Des langages tels que C et C++ sont couramment utilisés pour la programmation de bas niveau dans les applications de l'IoT, offrant un accès direct au matériel et une gestion de la mémoire.

Informatique quantique :

L'informatique quantique, bien qu'elle en soit encore à ses débuts, est censée révolutionner la puissance de calcul et les capacités de résolution de problèmes. Des chercheurs développent des langages de programmation spécialisés et des frameworks pour les ordinateurs quantiques, tels que Q#, afin d'exploiter leurs propriétés uniques.

Développement web et frameworks frontend :

Le web continue d'être une plateforme dominante, stimulant les avancées dans le développement front-end. JavaScript, avec des frameworks modernes tels que React, Angular et Vue.js, permet aux développeurs de créer des applications web riches et interactives.

Ces tendances émergentes façonnent l'avenir des langages de programmation, avec de nouveaux langages en cours de développement ou des langages existants améliorés pour répondre aux exigences particulières de ces domaines. Il est probable que nous assistions à l'émergence de nouvelles fonctionnalités et de nouveaux paradigmes, à des améliorations de performance, à l'apparition de nouveaux langages spécifiques à des domaines particuliers, ainsi qu'à une augmentation de la sécurité et de la fiabilité.

Conclusion – Pourquoi ne développerait-on pas un langage de programmation universelle ?

L'idée d'un langage de programmation universel est séduisante, car cela pourrait simplifier le développement logiciel et réduire le besoin d'apprendre de multiples langages. Cependant, il y a plusieurs raisons pour lesquelles un langage de programmation universel n'a pas encore été développé et pourrait ne jamais l'être :

1. Domaines d'application variés : Les langages de programmation sont conçus pour répondre à des besoins spécifiques et sont adaptés à des domaines d'application particuliers. Par exemple, Python malgré sa grande polyvalence est populaire pour l'analyse de données, tandis que JavaScript excellera pour le développement web et y sera largement utilisé. Il est difficile de créer un langage universel qui répond aux exigences de tous les domaines.
2. Performances et optimisation : Les langages de programmation ont des compromis en termes de performances et d'optimisation. Un langage de programmation universel pourrait ne pas offrir les meilleures performances dans tous les domaines et pour toutes les applications.
3. Préférences personnelles et adoption : Les développeurs ont souvent des préférences personnelles pour certains langages en raison de leur familiarité, de leur facilité d'utilisation ou de leur communauté. Un langage de programmation universel pourrait ne pas être adopté rapidement par l'ensemble de la communauté des développeurs.
4. Évolution technologique : La technologie évolue constamment, et de nouveaux langages de programmation sont créés pour répondre à des besoins émergents. Un langage de programmation universel devrait être évolutif et adaptable pour suivre le rythme des progrès technologiques.
5. Maintenance et support : La maintenance et le support d'un langage de programmation universel pourraient être coûteux et complexes, en raison de la diversité des domaines d'application et des besoins des développeurs.

Bien qu'il soit peu probable qu'un langage de programmation universel voie le jour, les initiatives visant à faciliter l'interopérabilité entre les langages et à promouvoir des normes communes peuvent contribuer à réduire la complexité et à améliorer la collaboration entre les développeurs.

Sources

https://www.albertschool.com/blog/une-breve-histoire-des-langages-de-programmation

https://en.wikipedia.org/wiki/Domain-specific_language

https://survey.stackoverflow.co/2022/#most-popular-technologies-language-prof

https://www.tiobe.com/tiobe-index/

https://fr.wikipedia.org/wiki/Chronologie_des_langages_de_programmation

https://le-m-verbatem.fr/chronologie-langages-programmation/

https://www.bocasay.com/fr/evolutions-tendances-langages-developpement-web/

https://calcul.math.cnrs.fr/attachments/evt/2019-05-atelier-optimisation/support02.pdf

https://fr.wikipedia.org/wiki/Typage_dynamique

https://programmation.developpez.com/actu/253829/Programmation-une-etude-revele-les-langages-les-plus-voraces-en-energie-Perl-Python-et-Ruby-en-tete-C-Rust-et-Cplusplus-les-langages-les-plus-verts/

https://fr.wikipedia.org/wiki/H%C3%A9ritage_(informatique)

https://www.organisation-performante.com/la-programmation-fonctionnelle-solution-pour-les-developpeurs/

https://www.i3s.unice.fr/~malapert/thesis/split/chapitre2.pdf

https://fr.wikipedia.org/wiki/R%C3%A9cursivit%C3%A9

https://fr.wikipedia.org/wiki/Programmation_par_contraintes

https://datascience.eu/fr/programmation-informatique/programmation-fonctionnelle/

https://fr.wikipedia.org/wiki/Agr%C3%A9gation_(programmation)

https://www.techno-science.net/definition/11396.html

https://www.geeksforgeeks.org/functional-programming-paradigm/

https://support.esri.com/fr-fr/gis-dictionary/low-level-language

https://fr.wikipedia.org/wiki/Langage_de_programmation_de_bas_niveau

https://waytolearnx.com/2019/04/difference-entre-langage-haut-niveau-et-langage-bas-niveau.html?utm_content=cmp-true

https://code-garage.fr/blog/que-signifient-bas-niveau-et-haut-niveau-en-programmation/

https://tech-lib.fr/langage-de-bas-niveau/

https://tech-lib.fr/langage-de-haut-niveau/

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/quest-ce-quun-langage-de-script/

https://fr.wikipedia.org/wiki/Langage_de_script

https://www.jetbrains.com/fr-fr/mps/concepts/domain-specific-languages/

https://fr.wikipedia.org/wiki/Langage_d%C3%A9di%C3%A9

https://fr.theastrologypage.com/domain-specific-language

https://learn.microsoft.com/fr-fr/visualstudio/modeling/about-domain-specific-languages?view=vs-2022

https://dept-info.labri.fr/ENSEIGNEMENT/INITINFO/initinfo/supports/book/node63.html

https://fr.wikipedia.org/wiki/S%C3%A9mantique

https://fr.wikipedia.org/wiki/Syntaxe

https://people.montefiore.uliege.be/geurts/Cours/iti/2012/04-semantique-part1-2012-2013.pdf

https://fr.wikipedia.org/wiki/Paradigme_(programmation)

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/paradigmes-de-programmation/

https://sites.google.com/site/cours1900/master-01-cours-1/paradigmes-des-langages-de-programmation

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/programmation-imperative/

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/programmation-declarative/

https://librecours.net/module/js/js02/declarative.xhtml

https://24pm.com/117-definitions/408-programmation-logique

https://www.jedha.co/formation-python/programmation-orientee-objet

https://www.lemagit.fr/definition/Programmation-orientee-objet

https://www.futura-sciences.com/tech/definitions/informatique-programmation-orientee-objet-19301/

https://fr.wikipedia.org/wiki/Compilateur

https://fr.wikipedia.org/wiki/Interpr%C3%A8te_(informatique)

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/compilateur-vs-interpreteur/

https://datascientest.com/python-tout-savoir

https://fr.wikipedia.org/wiki/Java_(langage)

https://aws.amazon.com/fr/what-is/java/

https://fr.tuto.com/blog/2020/11/le-langage-c.htm

https://fr.tuto.com/c-+-+/

https://fr.wikipedia.org/wiki/JavaScript

https://www.ruby-lang.org/fr/about/

https://www.epsi.fr/ruby-avantages-inconvenients/

https://www.php.net/

https://fr.wikipedia.org/wiki/PHP

https://fr.wikipedia.org/wiki/Swift_(langage_d%27Apple)

https://www.typescriptlang.org/

https://go.dev/

https://www.techtarget.com/searchitoperations/definition/Go-programming-language

https://fr.wikipedia.org/wiki/Kotlin_(langage)

https://kotlinlang.org/

https://www.rust-lang.org/

https://emeritus.org/blog/coding-rust-programming-language/

https://talks.freelancerepublik.com/pourquoi-comment-apprendre-c-sharp/

https://fr.wikipedia.org/wiki/C_sharp

https://www.intelligence-artificielle-school.com/actualite/quel-est-le-role-de-la-programmation-dans-l-ia/

https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022

https://aws.amazon.com/fr/what-is/quantum-computing/

https://learn.microsoft.com/fr-fr/azure/quantum/overview-what-is-qsharp-and-qdk