I'm not a bot

Il y a beaucoup de ressources disponibles qui peuvent t'aider à apprendre Python en un rien de temps. Des options gratuites aux options payantes, tu peux trouver une large sélection de cours en ligne et de sites web pour te guider à travers les leçons. En utilisant ces ressources, tu peux rapidement progresser du débutant à avancé. Dans cet article, nous allons expliquer ce qu'est Python, à quoi il sert et pourquoi tu pourrais vouloir l'apprendre. Ensuite, nous te fournirons quelques-uns des meilleurs endroits où tu peux apprendre Python le plus rapidement possible. Allons-y ! Python est un langage de programmation polyvalent, de haut niveau et orienté objet qui t'aide à résoudre des problèmes rapidement : Pour te donner une idée de la sophistication et de la praticité de ce langage de programmation, considère que certaines des plus grandes marques du monde ont utilisé Python dans leurs projets. Ces entreprises incluent Netflix, Google et Spotify. C'est également une langue préférée par de nombreux développeurs. L'année dernière, une étude de RedMonk a révélé que, parmi les développeurs, Python était le deuxième langage de programmation le plus populaire (derrière JavaScript). Python 2 et Python 3 sont des versions différentes du langage de programmation Python open-source. Cependant, Python 2 n'est plus pris en charge par la communauté Python. Python 3 est la dernière version et est recommandée pour tous les nouveaux projets. Python est utilisé dans de nombreux domaines, y compris : Développement web Calcul scientifique Analyse de données Intelligence Artificielle (IA) Apprentissage automatique Automatisation et script Test et prototypage de logiciels Tu peux également utiliser Python pour créer toutes sortes d'applications, y compris web, bureau et mobiles. L'un des cas d'utilisation les plus courants de Python est le développement web, qui est le processus de création de sites web et d'applications. Le développement web peut être divisé en trois parties : le développement frontend, le développement backend et le développement full-stack. Les développeurs front-end se concentrent sur l'interface utilisateur (UI) et l'expérience utilisateur d'un site web ou d'une application web. Ils utilisent HTML, CSS et JavaScript pour construire l'interface graphique utilisateur (GUI). Les développeurs backend se concentrent sur l'application côté serveur qui alimente le GUI frontend. Ils utilisent des langues comme Python pour écrire du code qui s'exécute sur un serveur web. Enfin, les développeurs Full-stack sont compétents à la fois en développement front-end et back-end. Abonnez-vous maintenant pour recevoir toutes les dernières mises à jour directement dans votre boîte de réception. Python est une langue populaire pour le développement web pour de nombreuses raisons. Il y a également une variété d'avantages à l'apprendre. Pour les débutants, Python est relativement facile à apprendre et dispose de bibliothèques puissantes pour les utilisateurs avancés. Il possède également une syntaxe simple qui est facile à lire et à comprendre. Ces caractéristiques font de Python un langage idéal pour les débutants qui commencent tout juste à coder. De plus, le langage Python est assez polyvalent pour divers usages. Cette large applicabilité en fait un bon choix pour les développeurs qui souhaitent apprendre une langue qu'ils peuvent utiliser pour plusieurs types de projets. C'est également rapide. Le code Python est compilé en bytecode, qui est ensuite exécuté par la machine virtuelle Python. Cette étape de compilation signifie que le code Python s'exécute plus rapidement que les langages interprétés comme PHP et Ruby. Un autre avantage est que Python dispose de bibliothèques puissantes qui facilitent le développement web. Ces bibliothèques incluent Django, Flask, et Bottle. Elles fournissent des outils utiles pour construire des sites web et des applications web. Par exemple, Django a un module appelé Django ORM qui facilite le travail avec les bases de données en Python. De plus, Flask propose un module appelé flask-restful qui facilite la création d'API REST. Enfin, étant donné que c'est un langage de programmation très populaire, Python bénéficie également d'une communauté robuste pour offrir du soutien et des ressources. Cela signifie que si tu rencontres des problèmes ou des préoccupations liés à Python, il y aura probablement un développeur expérimenté capable de fournir une solution. Techniquement, tu n'as pas besoin de connaître d'autres langages de programmation avant d'apprendre Python. Cependant, maîtriser différents langages peut rendre l'apprentissage du Python plus facile. Si tu commences juste avec le développement web, nous te recommandons de suivre un cours d'introduction à la programmation. Au minimum, il est préférable de bien maîtriser HTML, JavaScript et CSS. Il pourrait également être utile de comprendre les bases d'autres langages de programmation tels que PHP et des bibliothèques JavaScript comme React. Il y a deux manières d'utiliser Python pour le développement web : avec ou sans un framework. Un framework est un ensemble de bibliothèques et d'outils qui facilitent la création d'un site web ou d'une application web. Comme nous l'avons mentionné, les frameworks Python les plus populaires pour le développement web sont Django, Flask et Bottle. Nous te recommandons de te familiariser avec l'un de ces frameworks si tu débutes. Si tu as au moins une expérience modérée avec Python, tu pourrais essayer de développer sans utiliser de Framework. Cette approche est parfois appelée micro-framing car tu utilises uniquement les bibliothèques dont tu as besoin pour ton projet spécifique. Cette méthode peut être plus rapide et plus facile que d'utiliser un framework complet comme Django ou Flask. Le temps nécessaire pour apprendre Python dépendra de ton niveau d'expérience et de l'effort que tu es prêt à investir. En général, tu peux apprendre Python assez rapidement si tu peux consacrer quelques heures chaque jour. Maintenant que nous avons abordé ce qu'est Python et ses cas d'utilisation, discutons de comment tu peux commencer à l'apprendre. Voici cinq ressources utiles pour démarrer ! Le site officiel Python propose de nombreuses ressources, guides et tutoriels qui peuvent te présenter les bases des langages de programmation. Si tu commences à partir de zéro, nous recommandons le Guide du débutant, disponible sous l'onglet Docs: Ce guide est gratuit et accessible à tous. Il offre également de nombreux liens vers d'autres ressources que tu peux utiliser tout au long de ton parcours d'apprentissage. En plus de ce qui est essentiellement une carte pour ton programme Python, tu peux explorer divers quiz et devoirs, des feuilles de triche, des vérificateurs de style et d'autres outils. Nous recommandons de prendre des notes pour assimiler toutes les informations. Disponible également sur le site web de Python, un Guide du développeur. Cette ressource sera utile à consulter une fois que tu auras maîtrisé les bases et seras prêt à passer au niveau suivant. Udemy est une plateforme populaire où tu peux trouver des tonnes de cours en ligne couvrant une large gamme de sujets. Le site propose des leçons gratuites et payantes, y compris des options spécifiquement destinées aux développeurs web. Ces cours incluent un cours gratuit d'Introduction à la Programmation Python qui est parfait pour les débutants : Il comprend une heure et 39 minutes de contenu vidéo à la demande, couvrant les bases de Python, y compris les scripts et les fonctions. Tu peux également passer à la version payante, qui inclut un certificat de fin de cours et la messagerie directe avec l'instructeur. Lorsque tu es prêt à avancer, tu peux t'inscrire au cours Python From Beginner to Intermediate in 30 Mins. C'est une autre classe gratuite basée sur des vidéos qui t'enseignera des concepts de programmation Python plus sophistiqués. Codecademy est une plateforme d'apprentissage en ligne populaire et robuste destinée aux développeurs et programmeurs. C'est une ressource merveilleuse pour explorer et s'inscrire à des cours couvrant presque tous les sujets en science des données, développement web, informatique, et plus encore. Bien que non gratuit, le plan Pro propose un cours sur Python 3 qui peut servir d'excellente introduction à Python ainsi qu'aux concepts de programmation de base : Les leçons couvrent une variété de sujets, incluant : Syntaxe Python Conditionnels et contrôle de flux Fonctions Structures de données : listes et répertoires Et plus encore Le cours ne nécessite aucun prérequis. Si tu t'inscris au plan Pro, tu recevras un certificat à la fin. De plus, le cours prend environ 25 heures pour être complété. Si tu préfères les matériaux d'apprentissage basés sur le texte, LearnPython.org est une option solide : Le site est dédié à aider les personnes de tous niveaux d'expérience à apprendre le langage de programmation et à accéder à des ressources qui peuvent simplifier le processus. Il comprend des tutoriels de base et avancés, couvrant des sujets tels que : Variables et leurs types Leçons de science des données Générateurs, ensembles, compréhensions de listes, arguments de fonction, etc Les leçons sont interactives. Tu peux également trouver un groupe Facebook dédié aux cours où tu peux discuter du matériel avec d'autres étudiants. Une autre plateforme que tu peux utiliser pour trouver des cours gratuits et payants sur Python est Free Code Camp. Le site propose divers matériaux sur le codage et le développement web, incluant des cours, des boot camps et des actualités. Free Code Camp propose un cours Apprendre Python : De Zéro à Héros qui est gratuit et basé sur le texte : Il explique les bases de Python dans des sections simples et faciles à comprendre. Certains des sujets incluent : Comment fonctionnent les variables, les instructions conditionnelles et les boucles Comment utiliser les listes Itérations à travers les structures de données Objets et classes Si tu aimes le contenu basé sur des vidéos, tu peux également trouver une version YouTube où les mêmes sujets sont abordés. Le cours vidéo discute également de la façon d'installer PyCharm et Python. Ce cours comprend quatre heures et vingt minutes de contenu vidéo. Si tu complètes le cours pour débutants et que tu souhaites passer à des concepts plus avancés et des projets, tu peux également participer au One Month Python Bootcamp. C'est un cours payant, mais le matériel intensif basé sur des projets est à ton propre rythme. Cela te donnera les bases nécessaires pour devenir un développeur Python. Si tu commences juste avec Python, il existe de nombreux cours en ligne, plateformes et ressources que tu peux utiliser (dont beaucoup sont gratuits !) pour t'aider à apprendre rapidement le langage de programmation. Nous te recommandons de commencer par un cours d'introduction gratuit ou par des matériaux, tels que ceux proposés sur Python.org. Une fois que tu maîtriseras les bases, tu pourras passer à des enseignements plus approfondis et spécialisés proposés sur des plateformes tierces et premium. Cherches-tu une solution d'hébergement fiable et abordable pour tes projets de développement web ? Découvre les plans d'Shared Hosting DreamHost pour commencer ! Nous nous assurons que ton site est rapide, sécurisé et toujours disponible afin que tes visiteurs te fassent confiance. Les plans commencent à partir de $1.99/mo. Choisis Ton Plan Python est un langage de programmation extrêmement polyvalent. Vous pouvez l'utiliser pour créer des sites Web, des algorithmes de Machine Learning et même des drones autonomes. Un pourcentage énorme de programmeurs dans le monde utilise Python. Cela vous donne le pouvoir de créer presque n'importe quoi. Mais vous devez d'abord l'apprendre. Apprendre n'importe quel langage de programmation peut être intimidant. Alors, comment apprendre Python ? L'une des choses qui m'ait le plus frustrée lorsque j'apprenais Python était le caractère générique de toutes les ressources d'apprentissage. Je voulais apprendre à créer des sites Web en Python, mais il semblait que toutes les ressources d'apprentissage voulaient que je passe deux mois longs et ennuyeux sur la syntaxe Python avant même que je puisse penser à faire ce qui m'intéressait. La plupart des tutoriels supposent que vous devez apprendre toute la syntaxe Python avant de pouvoir commencer à faire quelque chose d'intéressant. C'est ce qui conduit à des mois consacrés à la syntaxe, alors que ce que vous voulez vraiment faire est d'analyser des données, de créer un site Web ou de créer un drone autonome. C'est ce qui fait que votre motivation s'estompe et que vous vous contentez de tout annuler. Après plusieurs essais, j'ai trouvé un processus qui fonctionnait mieux pour moi. Ce qui a bien fonctionné a été d'associer l'apprentissage des bases à la construction de choses intéressantes. J'ai passé le moins de temps possible à apprendre les bases, puis j'ai immédiatement plongé dans la création de choses qui m'intéressaient. Dans cet article, je vais vous expliquer comment apprendre Python étape par étape, quelle que soit la raison pour laquelle vous souhaitez apprendre Python. 1. Déterminez ce qui vous motive Avant de vous lancer dans l'apprentissage de Python, il convient de se demander pourquoi vous voulez l'apprendre. C'est parce que le voyage va être long et parfois douloureux. Sans suffisamment de motivation, vous ne réussirez probablement pas. Par exemple, j'ai dormi pendant les cours de programmation en école d'ingénieur quand je devais mémoriser la syntaxe et je n'étais pas motivé. D'autre part, lorsque j'ai eu besoin d'utiliser Python pour créer un site Web, je suis resté éveillé toute la nuit pour le terminer. Déterminer ce qui vous motive vous aidera à déterminer un objectif final et un chemin qui vous mènera là-bas sans ennui. Vous n'avez pas à trouver un projet précis, mais simplement un domaine général qui vous intéresse. Choisissez un domaine qui vous intéresse, tel que: Data Science / Apprentissage automatique Application mobile Sites Internet Jeux Matériel / Capteurs / Robots Des scripts pour automatiser votre travail Oui, vous pouvez créer des site web en utilisant Python ! Déterminez un ou deux domaines qui vous intéressent et sur lesquels vous êtes prêt à passer du temps. Vous allez orienter votre apprentissage vers eux et, éventuellement, construire des projets. 2. Apprendre la syntaxe de base Malheureusement, cette étape ne peut être ignorée. Vous devez apprendre les bases de la syntaxe Python avant de plonger plus profondément dans la zone de votre choix. Vous voulez passer le moins de temps possible là-dessus, car ce n'est pas très motivant. Personnellement, je l'ai fait à environ 30% dans les tutoriels Python, ce qui était suffisant. Voici quelques bonnes ressources pour vous aider à apprendre les bases: Codeacademy – fait un bon travail d'enseignement de la syntaxe de base et s'appuie bien sur lui-même. Learn Python the Hard Way – un livre qui enseigne les concepts Python des bases aux programmes plus approfondis. Syntaxe du langage Python Je ne saurais trop insister sur le fait que vous ne devriez consacrer que le minimum de temps possible à la syntaxe de base. Plus vite vous pourrez travailler sur des projets, plus vite vous apprendrez. Vous pouvez toujours vous référer à la syntaxe lorsque vous êtes bloqué plus tard. Vous ne devriez idéalement passer que quelques semaines sur cette phase, et certainement pas plus d'un mois. 3. Faire des projets structurés Une fois que vous avez appris la syntaxe de base, il est possible de créer vos propres projets. Les projets sont un excellent moyen d'apprendre, car ils vous permettent d'appliquer vos connaissances. Si vous n'appliquez pas vos connaissances, il vous sera difficile de les conserver. Les projets vont renforcer vos capacités, vous aider à apprendre de nouvelles choses et vous aider à construire un portefeuille à montrer aux employeurs potentiels. Cependant, les projets très libres à ce stade seront douloureux – vous serez souvent bloqué et devrez vous référer à la documentation. Pour cette raison, il est généralement préférable de créer des projets plus structurés jusqu'à ce que vous vous sentiez suffisamment à l'aise pour les réaliser entièrement par vous-même. De nombreuses ressources d'apprentissage proposent des projets structurés. Ces projets vous permettent de construire des choses intéressantes dans les domaines qui vous intéressent tout en vous évitant de rester bloqués. 4. Travaillez seul sur des projets Une fois que vous avez terminé certains projets structurés, il est temps de travailler sur vos propres projets. Vous continuerez à consulter des ressources et des concepts d'apprentissage, mais vous travaillerez sur ce sur quoi vous voulez travailler. Avant de vous lancer dans vos propres projets, vous devez vous sentir à l'aise de corriger les erreurs et les problèmes liés à vos programmes. Voici quelques ressources que vous devriez connaître: StackOverflow – un site de questions et réponses de la communauté où les gens discutent de problèmes de programmation. Vous pouvez trouver des questions spécifiques à Python ici. Google – l'outil le plus couramment utilisé par tous les programmeurs expérimentés. Très utile pour tenter de résoudre les erreurs Documentation Python – un bon endroit pour trouver du matériel de référence sur Python. Une fois que vous avez une bonne maîtrise des problèmes de débogage, vous pouvez commencer à travailler sur vos propres projets. Vous devriez travailler sur des choses qui vous intéressent. Par exemple, j'ai travaillé sur des outils pour négocier automatiquement des actions très rapidement après avoir appris la programmation. Conseils pour trouver des projets intéressants Étendez les projets sur lesquels vous travailliez précédemment et ajoutez plus de fonctionnalités. Rendez-vous aux rencontres Python dans votre région et trouvez des personnes qui travaillent sur des projets intéressants. Trouvez des packages open source auxquels contribuer. Voyez si des organisations à but non lucratif locales recherchent des développeurs bénévoles. Trouvez des projets que d'autres personnes ont réalisés et voyez si vous pouvez les étendre ou les adapter. Github est un bon endroit pour les trouver. Parcourez les articles de blog d'autres personnes pour trouver des idées de projets intéressantes. Pensez à des outils qui faciliteraient votre quotidien et construisez-les. N'oubliez pas de commencer très petit. Commencez par des choses très simples pour gagner en confiance. Il est préférable de démarrer un petit projet que vous finissiez plutôt qu'un énorme projet qui ne sera jamais réalisé. Il est également utile de trouver d'autres personnes avec qui travailler pour plus de motivation. Si vous ne pouvez vraiment pas penser à de bonnes idées de projet, en voici quelques unes: Data Science / Apprentissage automatique Une carte pour visualise les résultats des élections par région. Un algorithme qui prédit la météo. Un outil qui prédit le marché boursier. Un algorithme qui résume automatiquement les articles de presse. Application mobile Une application pour suivre la distance parcourue chaque jour. Une application qui vous envoie des notifications météo. Chat en temps réel basé sur la localisation. Sites Internet Un site qui vous aide à planifier vos repas hebdomadaires. Un site qui permet aux utilisateurs de revoir les jeux vidéo. Une plateforme de prise de notes. Jeux Un jeu mobile basé sur la localisation, où vous capturez un territoire. Un jeu où vous programmez pour résoudre des énigmes. Un exemple de jeu que vous pouvez faire avec Pygame. Capteurs / Robots Des capteurs qui surveillent la température de votre maison et vous permettent de surveiller votre maison à distance. Un réveil plus intelligent. Un robot autonome qui détecte les obstacles. Des scripts pour automatiser votre travail Un script pour automatiser la saisie de données. Un outil pour extraire des données du Web. La clé est de choisir une idée et de la faire. Si vous êtes trop pressé pour choisir le projet parfait, vous risquez de ne jamais en réaliser un. 5. Continuer à travailler sur des projets plus difficiles Continuez à augmenter la difficulté et la portée de vos projets. Si vous êtes complètement à l'aise avec ce que vous construisez, cela signifie qu'il est temps d'essayer quelque chose de plus fort. Voici quelques idées pour ce moment-là: Essayez d'enseigner à un novice comment construire un projet que vous avez réalisé. Pouvez-vous augmenter la performance de votre outil? Peut-il fonctionner avec plus de données ou peut-il gérer plus de trafic? Est-il possible de rendre votre outil utile pour plus de gens? Comment commercialiseriez-vous ce que vous avez fabriqué? Aller de l'avant En fin de compte, Python évolue tout le temps. Seules quelques personnes peuvent légitimement prétendre comprendre parfaitement ce langage. Vous devrez constamment apprendre et travailler sur des projets. Si vous faites cela correctement, vous vous retrouverez à regarder votre code d'il y a 6 mois et à vous dire à quel point il est terrible. Si vous arrivez à ce point, vous êtes sur la bonne voie. Travailler uniquement sur des sujets qui vous intéressent signifie que vous ne vous ennuierez jamais. Python est un langage vraiment amusant et enrichissant, et je pense que tout le monde peut atteindre un haut niveau de maîtrise de ce langage. J'espère que cet article vous a aidé a savoir comment apprendre Python. DSA to Development: A Complete GuideBeginner to AdvanceJAVA Backend Development - LiveIntermediate and AdvanceTech Interview 101 - From DSA to System Design for Working ProfessionalsBeginner to AdvanceFull Stack Development with React & Node JS - LiveBeginner to AdvanceJava Programming Online Course [Complete Beginner to Advanced]Beginner to AdvanceC++ Programming Course Online - Complete Beginner to AdvancedBeginner to AdvancePage 2Our website uses cookiesWe use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy Programming paradigm based on the concept of objects "Object-oriented" redirects here. For other meanings of object-oriented, see Object-orientation. UML notation for a class. This Button class has variables for data, and functions. Through inheritance, a subclass can be created as a subset of the Button class. Objects are instances of a class. Object-oriented programming (OOP) is a programming paradigm based on the concept of objects.[1] Objects can contain data (called fields, attributes or properties) and have actions they can perform (called procedures or methods and implemented in code). In OOP, computer programs are designed by making them out of objects that interact with one another.[2][3] Many of the most widely used programming languages (such as C++, Java,[4] and Python) support object-oriented programming to a greater or lesser degree, typically as part of multiple paradigms in combination with others such as imperative programming and declarative programming. Significant object-oriented languages include Ada, ActionScript, C++, Common Lisp, C#, Dart, Eiffel, Fortran 2003, Haxe, Java,[4] JavaScript, Kotlin, Logo, MATLAB, Objective-C, Object Pascal, Perl, PHP, Python, R, Raku, Ruby, Scala, SIMSCRIPT, Simula, Smalltalk, Swift, Vala and Visual Basic.NET. The idea of "objects" in programming started with the artificial intelligence group at MIT in the late 1950s and early 1960s. Here, "object" referred to LISP atoms with identified properties (attributes).[5][6] Another early example was Sketchpad created by Ivan Sutherland at MIT in 1960–1961. In the glossary of his technical report, Sutherland defined terms like "object" and "instance" (with the class concept covered by "master" or "definition"), albeit specialized to graphical interaction.[7] Later, in 1968, AED-0, MIT's version of the ALGOL programming language, connected data structures ("plexes") and procedures, prefiguring what were later termed "messages", "methods", and "member functions".[8][9] Topics such as data abstraction and modular programming were common points of discussion at this time. Meanwhile, in Norway, Simula was developed during the years 1961–1967.[8] Simula introduced essential object-oriented ideas, such as classes, inheritance, and dynamic binding.[10] Simula was used mainly by researchers involved with physical modelling, like the movement of ships and their content through cargo ports.[10] Simula is generally accepted as being the first language with the primary features and framework of an object-oriented language.[11] I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning – it took a while to see how to do messaging in a programming language efficiently enough to be useful). Alan Kay, [1] Influenced by both MIT and Simula, Alan Kay began developing his own ideas in November 1966. He would go on to create Smalltalk, an influential object-oriented programming language. By 1967, Kay was already using the term "object-oriented programming" in conversation.[1] Although sometimes called the "father" of object-oriented programming,[12] Kay has said his ideas differ from how object-oriented programming is commonly understood, and has implied that the computer science establishment did not adopt his notion.[1] A 1976 MIT memo co-authored by Barbara Liskov lists Simula 67, CLU, and Alphard as object-oriented languages, but does not mention Smalltalk.[13] In the 1970s, the first version of the Smalltalk programming language was developed at Xerox PARC by Alan Kay, Dan Ingalls and Adele Goldberg. Smalltalk-72 was notable for use of objects at the language level and its graphical development environment.[14] Smalltalk was a fully dynamic system, allowing users to create and modify classes as they worked.[15] Much of the theory of OOP was developed in the context of Smalltalk, for example multiple inheritance.[16] In the late 1970s and 1980s, object-oriented programming rose to prominence. The Flavors object-oriented Lisp was developed starting 1979, introducing multiple inheritance and mixins.[17] In August 1981, Byte Magazine highlighted Smalltalk and OOP, introducing these ideas to a wide audience.[18] LOOPS, the object system for Interlisp-D, was influenced by Smalltalk and Flavors, and a paper about it was published in 1982.[19] In 1986, the first Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) was attended by 1,000 people. This conference marked the beginning of efforts to consolidate Lisp object systems, eventually resulting in the Common Lisp Object System. In the 1980s, there were a few attempts to design processor architectures that included hardware support for objects in memory, but these were not successful. Examples include the Intel iAPX 432 and the Linn Smart Rekursiv. In the mid-1980s, new object-oriented languages like Objective-C, C++, and Eiffel emerged. Objective-C was developed by Brad Cox, who had used Smalltalk at ITT Inc. Bjarne Stroustrup created C++ based on his experience using Simula for his PhD thesis.[14] Bertrand Meyer produced the first design of the Eiffel language in 1985, which focused on software quality using a design by contract approach.[20] In the 1990s, object-oriented programming became the main way of programming, especially as more languages supported it. These included Visual FoxPro 3.0,[21][22] C++,[23] and Delphi[citation needed]. OOP became even more popular with the rise of graphical user interfaces, which used objects for buttons, menus and other elements. One well-known example is Apple's Cocoa framework, used on Mac OS X and written in Objective-C. OOP toolkits also enhanced the popularity of event-driven programming.[citation needed] At ETH Zürich, Niklaus Wirth and his colleagues created new approaches to OOP. Modula-2 (1978) and Oberon (1987), included a distinctive approach to object orientation, classes, and type checking across module boundaries. Inheritance is not obvious in Wirth's design since his nomenclature looks in the opposite direction: It is called type extension and the viewpoint is from the parent down to the inheritor. Many programming languages that existed before OOP have added object-oriented features, including Ada, BASIC, Fortran, Pascal, and COBOL. This sometimes caused compatibility and maintainability issues, as these languages were not originally designed with OOP in mind. In the new millenium, new languages like Python and Ruby have emerged that combine object-oriented and procedural styles. The most commercially important "pure" object-oriented languages continue to be Java, developed by Sun Microsystems, as well as C# and Visual Basic.NET (VB.NET), both designed for Microsoft's .NET platform. These languages show the benefits of OOP by creating abstractions from implementation. The .NET platform supports cross-language inheritance, allowing programs to use objects from multiple languages together. See also: Comparison of programming languages (object-oriented programming) and List of object-oriented programming terms Object-oriented programming focuses on working with objects, but not all OOP languages have every feature linked to OOP. Below are some common features of languages that are considered strong in OOP or support it along with other programming styles. Important exceptions are also noted.[24][25][26][27] Christopher J. Date pointed out that comparing OOP with other styles, like relational programming, is difficult because there isn't a clear, agreed-upon definition of OOP.[28] Further information: Imperative programming and Structured programming Features from imperative and structured programming are present in OOP languages and are also found in non-OOP languages. Variables hold different data types like integers, strings, lists, and hash tables. Some data types are built-in while others result from combining variables using memory pointers. Procedures – also known as functions, methods, routines, or subroutines – take input, generate output, and work with data. Modern languages include structured programming constructs like loops and conditionals. Support for modular programming lets programmers organize related procedures into files and modules. This makes programs easier to manage. Each modules has its own namespace, so items in one module will not conflict with items in another. Object-oriented programming (OOP) was created to make code easier to reuse and maintain.[29] However, it was not designed to clearly show the flow of a program's instructions—that was left to the compiler. As computers began using more parallel processing and multiple threads, it became more important to understand and control how instructions flow. This is difficult to do with OOP.[30][31][32][33] Main article: Object (computer science) An object is a type of data structure that has two main parts: fields and methods. Fields may also be known as members, attributes, or properties, and hold information in the form of state variables. Methods are actions, subroutines, or procedures, defining the object's behavior in code. Objects are usually stored in memory, and in many programming languages, they work like pointers that link directly to a contiguous block containing the object instance's data. Objects can contain other objects. This is called object composition. For example, an Employee object might have an Address object inside it, along with other information like "first_name" and "position". This type of structures shows "has-a" relationships, like "an employee has an address". Some believe that OOP places too much focus on using objects rather than on algorithms and data structures.[34][35] For example, programmer Rob Pike pointed out that OOP can make programmers think more about type hierarchy than composition.[36] He has called object-oriented programming "the Roman numerals of computing".[37] Rich Hickey, creator of Clojure, described OOP as overly simplistic, especially when it comes to representing real-world things that change over time.[35] Alexander Stepanov said that OOP tries to fit everything into a single type, which can be limiting. He argued that sometimes we need multisorted algebras—families of interfaces that span multiple types, such as in generic programming. Stepanov also said that calling everything an "object" doesn't add much understanding.[34] Sometimes, objects represent real-world things and processes in digital form.[38] For example, a graphics program may have objects such as "circle", "square", and "menu". An online shopping system might have objects such as "shopping cart", "customer", and "product". Niklaus Wirth said, "This paradigm [OOP] closely reflects the structure of systems in the real world and is therefore well suited to model complex systems with complex behavior".[39] However, more often, objects represent abstract entities, like an open file or a unit converter. Not everyone agrees that OOP makes it easy to copy the real world exactly or that doing so is even necessary. Bob Martin suggests that because classes are software, their relationships don't match the real-world relationships they represent.[40] Bertrand Meyer argues in Object-Oriented Software Construction, that a program is not a model of the world but a model of some part of the world; "Reality is a cousin twice removed".[41] Steve Yegge noted that natural languages lack the OOP approach of strictly prioritizing things (objects/nouns) before actions (methods/verbs), as opposed to functional programming which does the reverse.[42] This can sometimes make OOP solutions more complicated than those written in procedural programming.[43] Most OOP languages allow reusing and extending code through "inheritance". This inheritance can use either "classes" or "prototypes", which have some differences but use similar terms for ideas like "object" and "instance". In class-based programming, the most common type of OOP, every object is an instance of a specific class. The class defines the data format, like variables (e.g., name, age) and methods (actions the object can take). Every instance of the class has the same set of variables and methods. Objects are created using a special method in the class known as a constructor. Here are a few key terms in class-based OOP: Class variables – belong to the class itself, so all objects in the class share one copy. Instance variables – belong to individual objects; every object has its own version of these variables. Member variables – refers to both the class and instance variables that are defined by a particular class. Class methods – linked to the class itself and can only use class variables. Instance methods – belong to individual objects, and can use both instance and class variables Classes may inherit from other classes, creating a hierarchy of "subclasses". For example, an "Employee" class might inherit from a "Person" class. This means the Employee object will have all the variables from Person (like name variables) plus any new variables (like job position and salary). Similarly, the subclass may expand the interface with new methods. Most languages also allow the subclass to override the methods defined by superclasses. Some languages support multiple inheritance, where a class can inherit from more than one class, and other languages similarly support mixins or traits. For example, a mixin called UnicodeConversionMixin might add a method unicode_to_ascii() to both a FileReader and a WebPageScraper class. Some classes are abstract, meaning they cannot be directly instantiated into objects; they're only meant to be inherited into other classes. Other classes are utility classes which contain only class variables and methods and are not meant to be instantiated or subclassed.[44] In prototype-based programming, there aren't any classes. Instead, each object is linked to another object, called its prototype or parent. In Self, an object may have multiple or no parents,[45] but in the most popular prototype-based language, Javascript, every object has exactly one prototype link, up to the base Object type whose prototype is null. The prototype acts as a model for new objects. For example, if you have an object fruit, you can make two objects apple and orange, based on it. There is no fruit class, but they share traits from the fruit prototype. Prototype-based languages also allow objects to have their own unique properties, so the apple object might have an attribute sugar_content, while the orange or fruit objects do not. Some languages, like Go, don't use inheritance at all.[46] Instead, they encourage "composition over inheritance", where objects are built using smaller parts instead of parent-child relationships. For example, instead of inheriting from class Person, the Employee class could simply contain a Person object. This lets the Employee class control how much of Person it exposes to other parts of the program. Delegation is another language feature that can be used as an alternative to inheritance. Programmers have different opinions on inheritance. Bjarne Stroustrup, author of C++, has stated that it is possible to do OOP without inheritance.[47] Rob Pike has criticized inheritance for creating complicated hierarchies instead of simpler solutions.[48] See also: Object-oriented design People often think that if one class inherits from another, it means the subclass "is a" more specific version of the original class. This presumes the program semantics are that objects from the subclass can always replace objects from the original class without problems. This concept is known as behavioral subtyping, more specifically the Liskov substitution principle. However, this is often not true, especially in programming languages that allow mutable objects, objects that change after they are created. In fact, subtype polymorphism as enforced by the type checker in OOP languages cannot guarantee behavioral subtyping in most if not all contexts. For example, the circle-ellipse problem is notoriously difficult to handle using OOP's concept of inheritance. Behavioral subtyping is undecidable in general, so it cannot be easily implemented by a compiler. Because of this, programmers must carefully design class hierarchies to avoid mistakes that the programming language itself cannot catch. When a method is called on an object, the object itself—not outside code—decides which specific code to run. This process, called dynamic dispatch, usually happens at run time by checking a table linked to the object to find the correct method. In this context, a method call is also known as message passing, meaning the method name and its inputs are like a message sent to the object for it to act on. If the method choice depends on more than one type of object (such as other objects passed as parameters), it's called multiple dispatch. Dynamic dispatch works together with inheritance: if an object doesn't have the requested method, it looks up to its parent class (delegation), and continues up the chain until it finds the method or reaches the top. Data abstraction is a way of organizing code so that only certain parts of the data are visible to related functions (data hiding). This helps prevent mistakes and makes the program easier to manage. Because data abstraction works well, many programming styles, like object-oriented programming and functional programming, use it as a key principle. Encapsulation is another important idea in programming. It means keeping the internal details of an object hidden from the outside code. This makes it easier to change how an object works on the inside without affecting other parts of the program, such as in code refactoring. Encapsulation also helps keep related code together (decoupling), making it easier for programmers to understand. In object-oriented programming, objects act as a barrier between their internal workings and external code. Outside code can only interact with an object by calling specific public methods or variables. If a class only allows access to its data through methods and not directly, this is called information hiding. When designing a program, it's often recommended to keep data as hidden as possible. This means using local variables inside functions when possible, then private variables (which only the object can use), and finally public variables (which can be accessed by any part of the program) if necessary. Keeping data hidden helps prevent problems when changing the code later.[49] Some programming languages, like Java, control information hiding by marking variables as private (hidden) or public (accessible).[50] Other languages, like Python, rely on naming conventions, such as starting a private method's name with an underscore. Intermediate levels of access also exist, such as Java's protected keyword, (which allows access from the same class and its subclasses, but not objects of a different class), and the internal keyword in C#, Swift, and Kotlin, which restricts access to files within the same module.[51] Abstraction and information hiding are important concepts in programming, especially in object-oriented languages.[52] Programs often create many copies of objects, and each one works independently. Supporters of this approach say it makes code easier to reuse and intuitively represents real-world situations.[53] However, others argue that object-oriented programming does not enhance readability or modularity.[54][55] Eric S. Raymond has written that object-oriented programming languages tend to encourage thickly layered programs that destroy transparency.[56] Raymond compares this unfavourably to the approach taken with Unix and the C programming language.[56] One programming principle, called the "open/closed principle", says that classes and functions should be "open for extension, but closed for modification". Luca Cardelli has stated that OOP languages have "extremely poor modularity properties with respect to class extension and modification", and tend to be extremely complex.[54] The latter point is reiterated by Joe Armstrong, the principal inventor of Erlang, who is quoted as saying:[55] The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle. Leo Brodie says that information hiding can lead to copying the same code in multiple places (duplicating code),[57] which goes against the don't repeat yourself rule of software development.[58] Polymorphism is the use of one symbol to represent multiple different types.[59] In object-oriented programming, polymorphism more specifically refers to subtyping or subtype polymorphism, where a function can work with a specific interface and thus manipulate entities of different classes in a uniform manner.[60] For example, imagine a program has two shapes: a circle and a square. Both come from a common class called "Shape." Each shape has its own way of drawing itself. With subtype polymorphism, the program doesn't need to know the type of each shape, and can simply call the "Draw" method for each shape. The programming language runtime will ensure the correct version of the "Draw" method runs for each shape. Because the details of each shape are handled inside their own classes, this makes the code simpler and more organized, enabling strong separation of concerns. In object-oriented programming, objects have methods that can change or use the object's data. Many programming languages use a special word, like this or self, to refer to the current object. In languages that support open recursion, a method in an object can call other methods in the same object, including itself, using this special word. This allows a method in one class to call another method defined later in a subclass, a feature known as late binding. This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. (August 2009) (Learn how and when to remove this message) See also: List of object-oriented programming languages OOP languages can be grouped into different types based on how they support and use objects: Pure OOP languages: In these languages, everything is treated as an object, even basic things like numbers and characters. They are designed to fully support and enforce OOP. Examples: Ruby, Scala, Smalltalk, Eiffel, Emerald,[61] JADE, Self, Raku. Mostly OOP languages: These languages focus on OOP but also include some procedural programming features. Examples: Java, Python, C++, C#, Delphi/Object Pascal, VB.NET. Retrofitted OOP languages: These were originally designed for other types of programming but later added some OOP features. Examples: PHP, JavaScript, Perl, Visual Basic (derived from BASIC), MATLAB, COBOL 2002, Fortran 2003, ABAP, Ada 95, Pascal. Unique OOP languages: These languages have OOP features like classes and inheritance but use them in their own way. Examples: Oberon, BETA. Object-based languages: These support some OOP ideas but avoid traditional class-based inheritance in favor of direct manipulation of objects. Examples: JavaScript, Lua, Modula-2, CLU, Go. Multi-paradigm languages: These support both OOP and other programming styles, but OOP is not the predominant style in the language. Examples include Tcl, where TclOO allows both prototype-based and class-based OOP, and Common Lisp, with its Common Lisp Object System. The TIOBE programming language popularity index graph from 2002 to 2023. In the 2000s the object-oriented Java (orange) and the procedural C (dark blue) competed for the top position. Many popular programming languages, like C++, Java, and Python, use object-oriented programming. In the past, OOP was widely accepted,[62] but recently, some programmers have criticized it and prefer functional programming instead.[63] A study by Potok et al. found no major difference in productivity between OOP and other methods.[64] Paul Graham, a well-known computer scientist, believes big companies like OOP because it helps manage large teams of average programmers. He argues that OOP adds structure, making it harder for one person to make serious mistakes, but at the same time restrains smart programmers.[65] Eric S. Raymond, a Unix programmer and open-source software advocate, argues that OOP is not the best way to write programs.[56] Richard Feldman says that, while OOP features helped some languages stay organized, their popularity comes from other reasons.[66] Lawrence Krubner argues that OOP doesn't offer special advantages compared to other styles, like functional programming, and can make coding more complicated.[67] Luca Cardelli says that OOP is slower and takes longer to compile than procedural programming.[54] In recent years, object-oriented programming (OOP) has become very popular in dynamic programming languages. Some languages, like Python, PowerShell, Ruby and Groovy, were designed with OOP in mind. Others, like Perl, PHP, and ColdFusion, started as non-OOP languages but added OOP features later (starting with Perl 5, PHP 4, and ColdFusion version 6). On the web, HTML, XHTML, and XML documents use the Document Object Model (DOM), which works with the JavaScript language. JavaScript is a well-known example of a prototype-based language. Instead of using classes like other OOP languages, JavaScript creates new objects by copying (or "cloning") existing ones. Another language that uses this method is Lua. When computers communicate in a client-server system, they send messages to request services. For example, a simple message might include a length field (showing how big the message is), a code that identifies the type of message, and a data value. These messages can be designed as structured objects that both the client and server understand, so that each type of message corresponds to a class of objects in the client and server code. More complex messages might include structured objects as additional details. The client and server need to know how to serialize and deserialize these messages so they can be transmitted over the network, and map them to the appropriate object types. Both clients and servers can be thought of as complex object-oriented systems. The Distributed Data Management Architecture (DDM) uses this idea by organizing objects into four levels: Basic message details - Information like message length, type, and data. Objects and collections - Similar to how objects work in Smalltalk, storing messages and their details. Managers - Like file directories, these organize and store data, as well as provide memory and processing power. They are similar to IBM i Objects. Clients and servers - These are full systems that include managers and handle security, directory services, and multitasking. The first version of DDM defined distributed file services. Later, it was expanded to support databases through the Distributed Relational Database Architecture (DRDA). Design patterns are common solutions to problems in software design. Some design patterns are especially useful for object-oriented programming, and design patterns are typically introduced in an OOP context. The following are notable software design patterns for OOP objects.[68] Function object: Class with one main method that acts like an anonymous function (in C++, the function operator, operator()) Immutable object: does not change state after creation First-class object: can be used without restriction Container object: contains other objects Factory object: creates other objects Metaobject: Used to create other objects (similar to a class, but an object) Prototype object: a specialized metaobject that creates new objects by copying itself Singleton object: only instance of its class for the lifetime of the program Filter object: receives a stream of data as its input and transforms it into the object's output A common anti-pattern is the God object, an object that knows or does too much. Main article: Design pattern (computer science) Design Patterns: Elements of Reusable Object-Oriented Software is a famous book published in 1994 by four authors: Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. People often call them the "Gang of Four". The book talks about the strengths and weaknesses of object-oriented programming and explains 23 common ways to solve programming problems. These solutions, called "design patterns," are grouped into three types: Creational patterns (5): Factory method pattern, Abstract factory pattern, Singleton pattern, Builder pattern, Prototype pattern Structural patterns (7): Adapter pattern, Bridge pattern, Composite pattern, Decorator pattern, Facade pattern, Flyweight pattern, Proxy pattern Behavioral patterns (11): Chain-of-responsibility pattern, Command pattern, Interpreter pattern, Iterator pattern, Mediator pattern, Memento pattern, Observer pattern, State pattern, Strategy pattern, Template method pattern, Visitor pattern Main articles: Object-relational impedance mismatch, Object-relational mapping, and Object database Both object-oriented programming and relational database management systems (RDBMSs) are widely used in software today. However, relational databases don't store objects directly, which creates a challenge when using them together. This issue is called object-relational impedance mismatch. To solve this problem, developers use different methods, but none of them are perfect.[69] One of the most common solutions is object-relational mapping (ORM), which helps connect object-oriented programs to relational databases. Examples of ORM tools include Visual FoxPro, Java Data Objects, and Ruby on Rails ActiveRecord. Some databases, called object databases, are designed to work with object-oriented programming. However, they have not been as popular or successful as relational databases. Date and Darwen have proposed a theoretical foundation that uses OOP as a kind of customizable type system to support RDBMSs, but it forbids objects containing pointers to other objects.[70] In responsibility-driven design, classes are built around what they need to do and the information they share, in the form of a contract. This is different from data-driven design, where classes are built based on the data they need to store. According to Wirfs-Brock and Wilkerson, the originators of responsibility-driven design, responsibility-driven design is the better approach.[71] SOLID is a set of five rules for designing good software, created by Michael Feathers: Single responsibility principle: A class should have only one reason to change. Open/closed principle: Software entities should be open for extension, but closed for modification. Liskov substitution principle: Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. Interface segregation principle: Clients should not be forced to depend upon interfaces that they do not use. Dependency inversion principle: Depend upon abstractions, not concretes. GRASP (General Responsibility Assignment Software Patterns) is another set of software design rules, created by Craig Larman, that helps developers assign responsibilities to different parts of a program:[72] Creator Principle: allows classes create objects they closely use. Information Expert Principle: assigns tasks to classes with the needed information. Low Coupling Principle: reduces class dependencies to improve flexibility and maintainability. High Cohesion Principle: designing classes with a single, focused responsibility. Controller Principle: assigns system operations to separate classes that manage flow and interactions. Polymorphism: allows different classes to be used through a common interface, promoting flexibility and reuse. Pure Fabrication Principle: create helper classes to improve design, boost cohesion, and reduce coupling. See also: Formal semantics of programming languages In object-oriented programming, objects are things that exist while a program is running. An object can represent anything, like a person, a place, a bank account, or a table of data. Many researchers have tried to formally define how OOP works. Records are the basis for understanding objects. They can represent fields, and also methods, if function literals can be stored. However, inheritance presents difficulties, particularly with the interactions between open recursion and encapsulated state. Researchers have used recursive types and co-algebraic data types to incorporate essential features of OOP.[73] Abadi and Cardelli defined several extensions of System F