Algorithmique et programmation

Cours d’algorithmique et programmation

Cours de 14h

3 séances de 2h

2 séances de 3h

Une dernière séance de 2h : examen ou dossier (à voir avec vous)

Algorithme

Algorithme : mot dérivé du nom du mathématicien Al Khwarizmi qui a vécu au 9ème siècle, et était membre de l'académie des sciences à Bagdad .

Un algorithme:

- est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat.
- est souvent exprimé avec une notation indépendante de tout langage de programmation.

Algorithme

Un algorithme prend des **données en entrée**, exprime un **traitement particulier** et fournit des **données en sortie**.

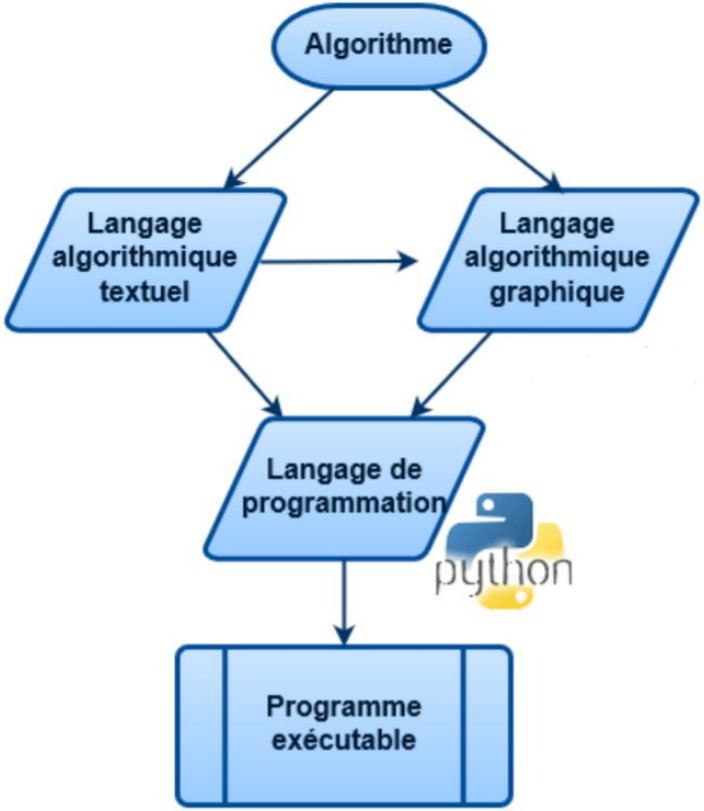
Algorithme et programmation

Un programme est la traduction d'un algorithme en un langage interprétable par un ordinateur.

Les outils utilisés pour exprimer un algorithme

Le langage algorithmique textuel

C'est l'expression de l'algorithme dans un langage parlé comme le français.

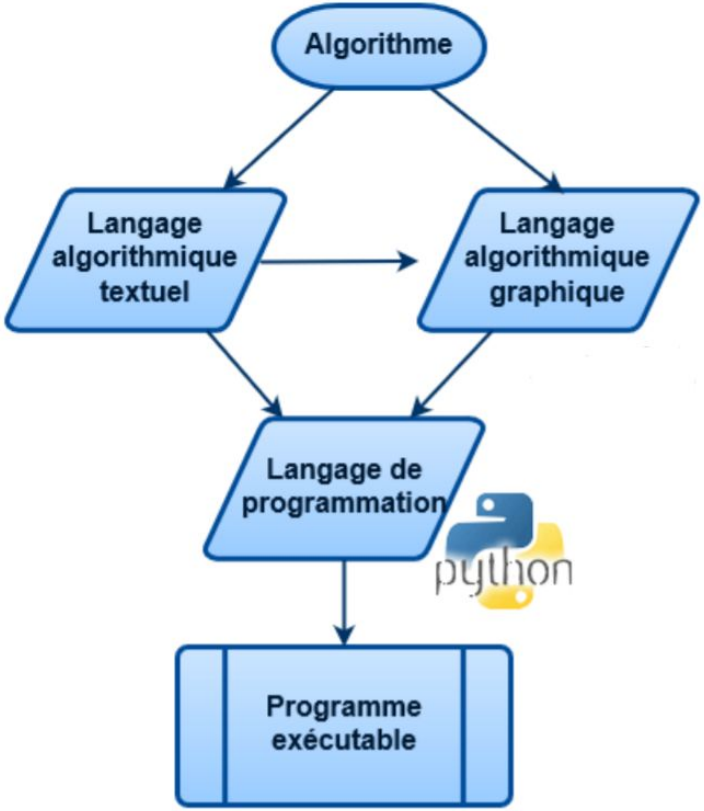


Les outils utilisés pour exprimer un algorithme

Le langage algorithmique graphique

Permet de représenter sous forme graphique les instructions et les opérations de l'algorithme.

Exemple : UML, Blocky

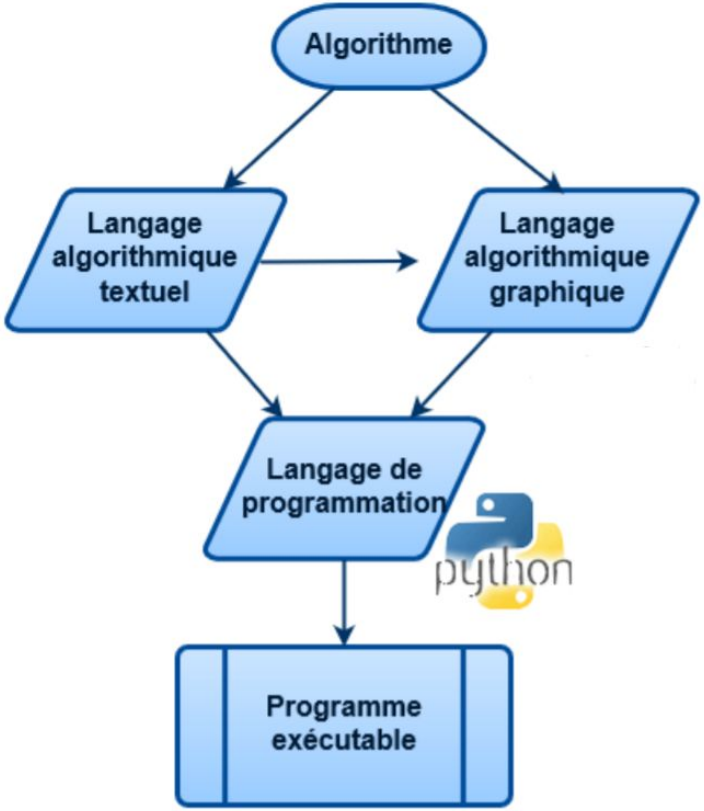


Les outils utilisés pour exprimer un algorithme

Les langages de programmation

Permettent de traduire un algorithme en un programme exécutable par un ordinateur.

Exemple : Python, R, VBA, ..



Les notions de base en algorithmique

Concevoir et écrire des algorithmes c'est décrire comment résoudre un problème particulier.

Il est donc indispensable de :

- Savoir formaliser son raisonnement
- Expliciter son raisonnement

L'ordinateur ne peut pas deviner ce que nous voulons.

Les notions de base en algorithmique

Savoir expliquer comment faire un travail sans la moindre ambiguïté

Utiliser un langage simple

Définir une suite d’actions à entreprendre en respectant une chronologie imposée

Les notions de base en algorithmique

Un algorithme est universel :

- Il ne dépend pas du langage dans lequel il est implémenté
- Ni de la machine qui exécutera le programme correspondant

Un même algorithme doit :

- Pouvoir être implémenté en VBScript et exécuté sur Windows
- Pouvoir être implémenté en bash et exécuté sur un système UNIX/Linux
- Pouvoir être implémenté en JavaScript et exécuté dans un navigateur
- Pouvoir être implémenté en Objective-C et exécuté sur un MacOS

Exemple algorithme calcul TVA

Un algorithme qui calcule la TVA en fonction du taux que l'utilisateur choisit et le montant qu'il saisit.

- **Description de l'algorithme en français**
- **Implémentation en VBScript (Langage exclusif à Windows)**
- **Implémentation en JavaScript (Langage du navigateur web)**
- **Implémentation en Python (langage multiplateforme)**

Les problèmes fondamentaux avec les algorithmes

L'analyse de la complexité d'un algorithme consiste en l'étude formelle de la quantité de ressources nécessaire à l'exécution de cet algorithme.

- En combien de temps un algorithme va -t-il atteindre le résultat escompté?
- De quel espace a-t-il besoin?

Les problèmes fondamentaux avec les algorithmes

Calculabilité d’un algorithme :

Existe-t-il des tâches pour lesquelles il n'existe aucun algorithme ?

Etant donnée une tâche, peut-on dire s'il existe un algorithme qui la résolve ?

Les problèmes fondamentaux avec les algorithmes

Correction

Peut-on être sûr qu'un algorithme réponde au problème pour lequel il a été conçu ?

Exemple calcul TVA

Ressources pour aller plus loin

MicroMasters® Program in Algorithms and Data Structures :
https://www.edx.org/micromasters/ucsandiegox-algorithms-and-data-structures?index=product&queryID=0ef77c72cd0c93b80c630652b068ee10&position=2

Professional Certificate in Data Structures and Algorithms :
https://www.edx.org/professional-certificate/gtx-data-structures-and-algorithms?index=product&queryID=0ef77c72cd0c93b80c630652b068ee10&position=3

JavaScript Algorithms and Data Structures Masterclass :
https://www.udemy.com/course/js-algorithms-and-data-structures-masterclass/

Bibliographie

Algorithmique et programmation,
https://eric.univ-lyon2.fr/jdarmont/docs/m1hn-algo-prog-doc.pdf

Les langages de programmation

Les langages de programmation

Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent.

Un langage de programmation est composé d'un alphabet, d'un vocabulaire, de règles de grammaire, de significations, mais aussi d'un environnement de traduction censé rendre sa syntaxe compréhensible par la machine.

Les langages de programmation

Vidéo : évolution des langages de programmation 1965-2019

https://youtu.be/Og847HVwRSI

Comment l'ordinateur comprend le code qu'on écrit

Le processeur d'un ordinateur se charge d'exécuter les instructions qui lui sont fournies.

Le processeur ne comprend que le langage machine aussi appelé code machine ou encore code natif.

Un programme en code natif est composé d'instructions directement reconnues par un processeur. Le code natif est donc lié à une famille particulière de processeurs partageant le même jeu d'instructions.

Comment l’ordinateur comprend le code qu’on écrit

Les programmeurs n’écrivent pas de code natif directement, ils rédigent des « programmes sources » en suivant les conventions d’un langage de programmation, et la traduction de ces programmes sources en code natif est faite par des programmes (assembleur, compilateur, édition de liens, etc.).

Comment l'ordinateur comprend le code qu'on écrit

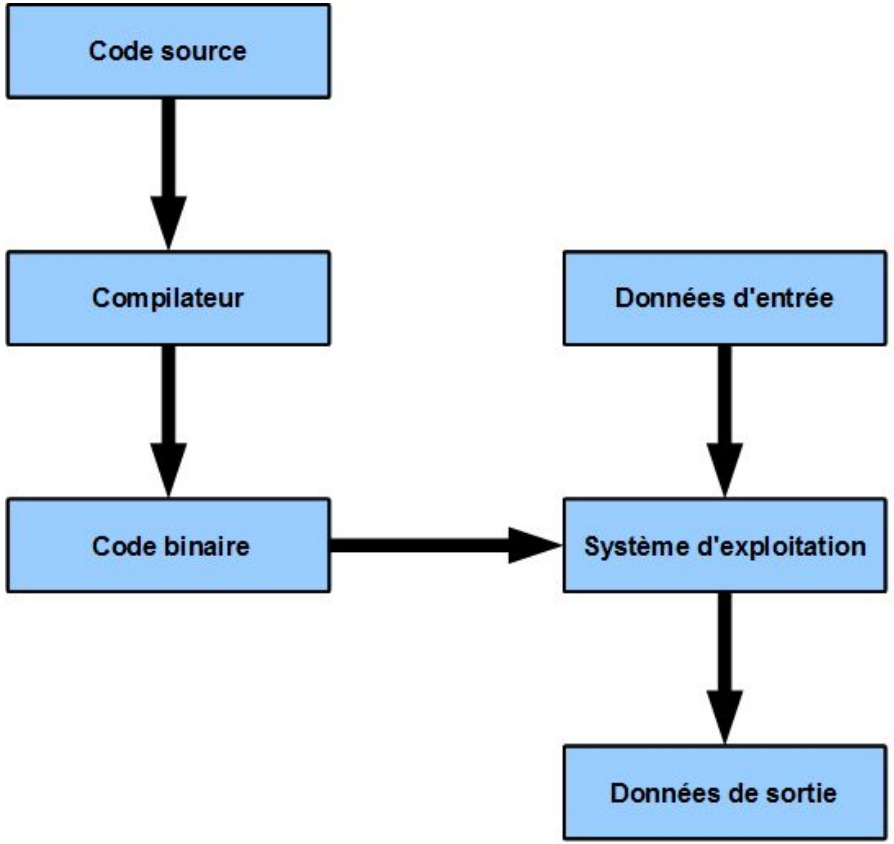
Pour plusieurs raisons comme la portabilité du code, la simplicité du langage tous les langages de programmation ne peuvent pas être compilés en code natif.

On distingue donc 3 principaux types de langages de programmation :

- Les langages compilés
- Les langages interprétés
- Langages compilés en bytecode *(hybride)*

Comment l'ordinateur comprend le code qu'on écrit

Les langages compilés



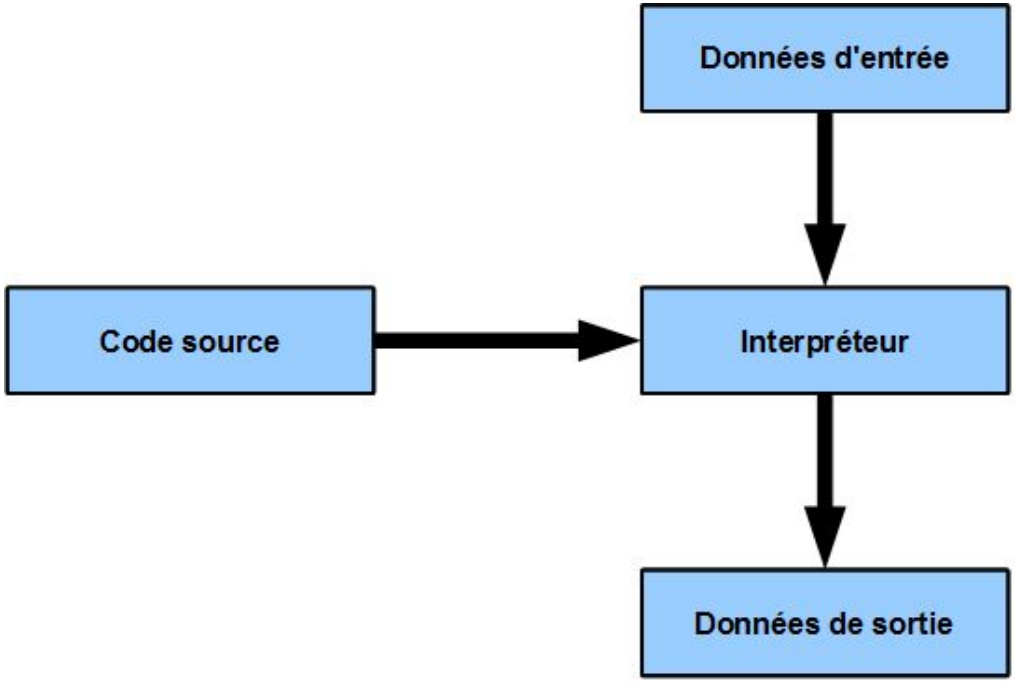
Comment l’ordinateur comprend le code qu’on écrit

Les langages compilés

Dans ces langages, le code source (celui que vous écrivez) est tout d'abord compilé, par un logiciel qu'on appelle compilateur, en un code binaire qu'un humain ne peut pas lire mais qui est très facile à lire pour un ordinateur. C'est alors directement le système d'exploitation qui va utiliser le code binaire et les données d'entrée pour calculer les données de sortie :

Comment l'ordinateur comprend le code qu'on écrit

Langages interprétés



Comment l’ordinateur comprend le code qu’on écrit

Langages interprétés

Dans ces langages, le code source (celui que vous écrivez) est interprété, par un logiciel qu'on appelle interpréteur. Celui-ci va utiliser le code source et les données d'entrée pour calculer les données de sorti.

Le code source va être lu ligne par ligne par l’interpréteur qui dit exactement au processeur ce qu’il doit faire.

Comment l’ordinateur comprend le code qu’on écrit

Langages compilés vs langages interprétés

Dans un langage interprété, le même code source pourra marcher directement sur tout ordinateur. Avec un langage compilé, il faudra (en général) tout recompiler à chaque fois ce qui pose parfois des soucis.

Dans un langage compilé, le programme est directement exécuté sur l'ordinateur, donc il sera en général plus rapide que le même programme dans un langage interprété.

Comment l'ordinateur comprend le code qu'on écrit

Langages compilés vs langages interprétés

Dans le cas d'un langage compilé :

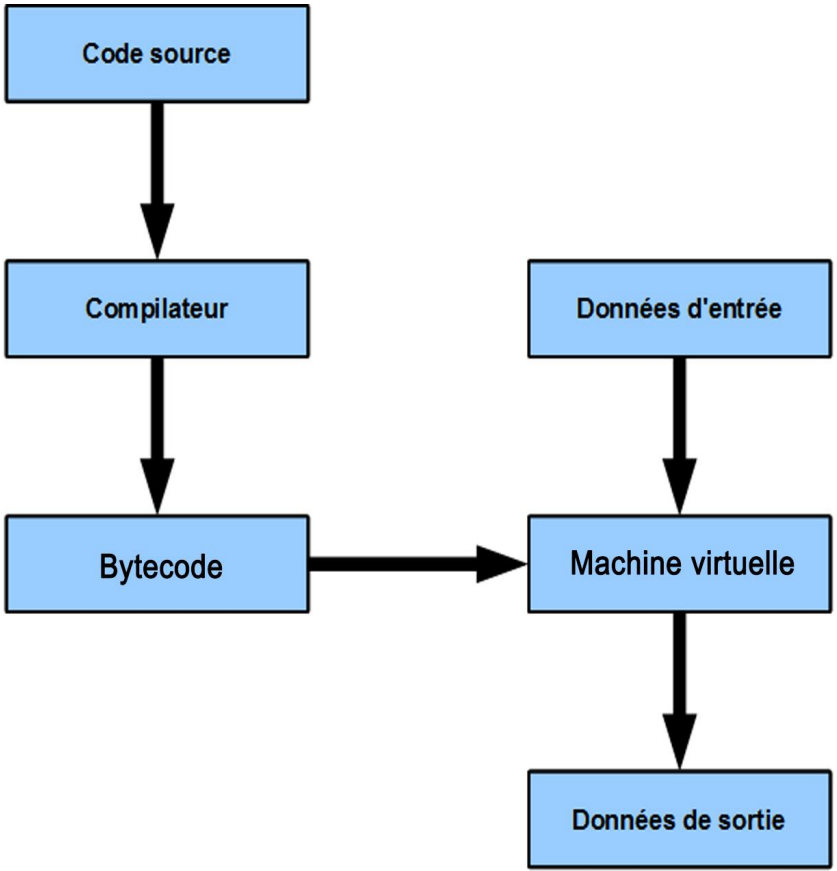
- Le code source n'est pas lisible par l'utilisateur car il est compilé
- Une fois le code compilé, impossible de le modifier

Dans le cas d'un langage interprété :

- Le code source doit être lu par l'interpréteur installé sur la machine.
- Ce qui implique que l'utilisateur peut modifier le code

Comment l'ordinateur comprend le code qu'on écrit

Langages compilés en bytecodes



Comment l'ordinateur comprend le code qu'on écrit

Langages compilés en bytecodes

Pour ce type de langage, le code source est compilé en un **code intermédiaire** qui sera exécuté par une **machine virtuelle**. C'est la machine virtuelle qui se charge de convertir le bytecode en code machine exécutable par le processeur.

Parce que le code source est compilé, il est beaucoup plus rapide à s'exécuter qu'un code source interprété.

Parce que le code source est compris par la machine virtuelle indépendamment de la plateforme, il peut être exécuté sur n'importe quel machine sur qui la machine virtuelle est installée.

Exemples catégories de langages

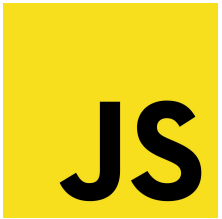
Quelques langages compilés :

C, C++, Fortran, Go, Rust

Exemples catégories de langages

Quelques langages interprété :

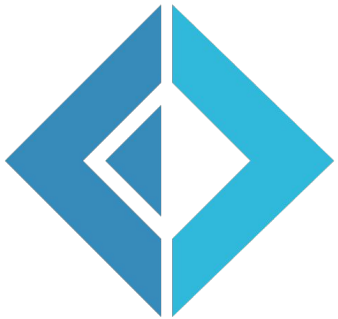
Python, R, JavaScript, Ruby, PHP, Perl



Exemples catégories de langages

Quelques langages compilés en bytecode :

Java, Kotlin, C#, F#, VB.NET,


Exemples catégories de langages

Langage de programmation à usage général :

Ce sont des langages de programmation destinés à être utilisés pour tout domaine que touche la programmation.

Ils sont créés pour être versatiles. Ils possèdent des librairies pour faire certaines tâches spécifiques.

Exemple : **Java, Python, C/C++, C#, F#, VB.NET, Rust, Go**

Python possède un gros écosystème pour la Data Science comparable à R, Matlab bien que ce soit un langage à usage général.

Exemples catégories de langages

Langage de programmation à usage dédié :

Un langage dédié est un langage de programmation dont les spécifications sont conçues pour répondre aux contraintes d'un domaine d'application précis.

Une connaissance du domaine d'application est souvent requis pour utiliser les langages de programmation dédié.

Exemple: **R, SAS, Matlab, Erlang, Perl, PHP, JavaScript**

Certains langages comme R, PHP et JavaScript deviennent de plus en plus universels grâce aux efforts de la communauté d'utilisateurs.

Avec R on peut aujourd'hui créer des applications web (R Shiny), des API (plumber) ...

Langage de programmation et langage informatique

Il existe plusieurs langages informatiques.

Tout langage informatique n'est pas un langage de programmation.

Par exemple : SQL, XML, HTML, CSS, Latex sont des langages informatiques mais ne sont pas des langages de programmation

Pour aller plus loin

Python deep dive : https://www.udemy.com/user/fredbaptiste/

Comment suivre ce cours

Pour les applications :

Il est possible d'utiliser :

- Son ordinateur personnel : installer Python
- Utiliser les services en lignes pour exécuter Python : Google Colab, Kaggle
- Utiliser les PC de la fac

Comment suivre ce cours

Pour installer Python : https://www.python.org/

Les services en ligne : Kaggle, Google Colab