Des éléments communs

Langage

Un langage de programmation sert à traduire des algorithmes pour les exécuter sur un ordinateur.

Langage

Un langage de programmation sert à traduire des algorithmes pour les exécuter sur un ordinateur.
Il est composé

Langage

Un langage de programmation sert à traduire des algorithmes pour les exécuter sur un ordinateur.
Il est composé

- d'un alphabet (ensemble de lettres et de symboles);

Langage

Un langage de programmation sert à traduire des algorithmes pour les exécuter sur un ordinateur.
Il est composé

- d'un alphabet (ensemble de lettres et de symboles);
- d'un vocabulaire (les *mots-clés* du langage);

Langage

Un langage de programmation sert à traduire des algorithmes pour les exécuter sur un ordinateur.
Il est composé

- d'un alphabet (ensemble de lettres et de symboles);
- d'un vocabulaire (les *mots-clés* du langage);
- d'une grammaire (la *syntaxe* du langage).

Notions communes

- une *instruction* est un ordre donné ;

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;
- une *constante* est un nom qui fait référence à une valeur immuable ;

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;
- une *constante* est un nom qui fait référence à une valeur immuable ;
- un *type* qui sert à classifier une variable ou une constante et conditionne les opérations qu'il est possible d'effectuer ;

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;
- une *constante* est un nom qui fait référence à une valeur immuable ;
- un *type* qui sert à classifier une variable ou une constante et conditionne les opérations qu'il est possible d'effectuer ;
- la *déclaration* consiste à renseigner le traducteur du programme sur la nature des données du programme (type, valeur).

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;
- une *constante* est un nom qui fait référence à une valeur immuable ;
- un *type* qui sert à classifier une variable ou une constante et conditionne les opérations qu'il est possible d'effectuer ;
- la *déclaration* consiste à renseigner le traducteur du programme sur la nature des données du programme (type, valeur).
- les *structures de contrôle* telles que le *test* et les *boucles*.

Notions communes

- une *instruction* est un ordre donné ;
- une *variable* est un nom qui fait référence à une donnée manipulée par le programme et susceptible de changer au cours de celui-ci ;
- une *constante* est un nom qui fait référence à une valeur immuable ;
- un *type* qui sert à classifier une variable ou une constante et conditionne les opérations qu'il est possible d'effectuer ;
- la *déclaration* consiste à renseigner le traducteur du programme sur la nature des données du programme (type, valeur).
- les *structures de contrôle* telles que le *test* et les *boucles*.
- une *fonction* (ou procédure, ou méthode) sert à isoler un fragment de programme pour pouvoir l'utiliser (éventuellement plusieurs fois) de manière paramétrée.

Des différences formelles

Suivant les langages

- L'affectation peut être signifiée par `=` (comme en Python, Basic, C, Fortran…) ou par `:=` (Ada, Algol, Go pour partie…). On utilise aussi `<-` en Caml.

Suivant les langages

- L'affectation peut être signifiée par `=` (comme en Python, Basic, C, Fortran…) ou par `:=` (Ada, Algol, Go pour partie…). On utilise aussi `<-` en Caml.
- Les *listes* (ou tableaux) sont très souvent indicées à partir de zéro… Mais pas toujours (Fortran, Lua).

Suivant les langages

- L'affectation peut être signifiée par `=` (comme en Python, Basic, C, Fortran…) ou par `:=` (Ada, Algol, Go pour partie…). On utilise aussi `<-` en Caml.
- Les *listes* (ou tableaux) sont très souvent indicées à partir de zéro… Mais pas toujours (Fortran, Lua).
- Très souvent, les structures de contrôles sont délimitées par des accolades (C, Java, Kotlin…) ou par des `begin` et des `end` (Ruby, Pascal…). Parfois on utilise des variantes du `end` telles que `done`, `END-IF` ou `NEXT` pour signifier une fin de boucle « pour ».

Langages similaires / éloigné

Certains langages ont des syntaxes très similaires : le langage C, JAVA et JAVASCRIPT par exemple (notamment le fait qu'un point-virgule termine une ligne).

Langages similaires / éloigné

Certains langages ont des syntaxes très similaires : le langage C, JAVA et JAVASCRIPT par exemple (notamment le fait qu'un point-virgule termine une ligne).
D'autres ont une syntaxe et une mise en forme particulière, éloignée de tous les autres, comme le BASIC ou le COBOL.

Des différences structurelles

Déclaration des variables

Certains langages obligent à déclarer le type des variables lors de leur création. C'est le cas de C ou JAVA.

Déclaration des variables

Certains langages obligent à déclarer le type des variables lors de leur création. C'est le cas de C ou JAVA.
D'autres obligent même à renseigner les variables et leur type avant toute chose, comme ADA, ALGOL, COBOL. Ce n'est pas le cas en PYTHON ou en RUBY.

Compilé ou interprété

Une autre différence majeure vient de la manière dont est traité le programme par le traducteur :

Compilé ou interprété

Une autre différence majeure vient de la manière dont est traité le programme par le traducteur :

- En C ou en C++, le programme est transformé en *langage-machine* par un *compilateur*. L'intérêt est que l'on gagne en rapidité lors de l'exécution du programme.

Compilé ou interprété

Une autre différence majeure vient de la manière dont est traité le programme par le traducteur :

- En C ou en C++, le programme est transformé en *langage-machine* par un *compilateur*. L'intérêt est que l'on gagne en rapidité lors de l'exécution du programme.
- En PYTHON le programme est *interprété* lors de son exécution, ligne par ligne.

Compilé ou interprété

Une autre différence majeure vient de la manière dont est traité le programme par le traducteur :

- En C ou en C++, le programme est transformé en *langage-machine* par un *compilateur*. L'intérêt est que l'on gagne en rapidité lors de l'exécution du programme.
- En PYTHON le programme est *interprété* lors de son exécution, ligne par ligne.
- Beaucoup de langages utilisent un procédé hybride : le langage utilise une machine virtuelle, et les programmes sont compilés dans le langage de cette machine virtuelle, qui sera ensuite exécuté. La compilation peut être faite avant et le résultat stocké dans un fichier, ou bien être faite à la volée (*Just In Time compilation*).

Compilé ou interprété

Une autre différence majeure vient de la manière dont est traité le programme par le traducteur :

- En C ou en C++, le programme est transformé en *langage-machine* par un *compilateur*. L'intérêt est que l'on gagne en rapidité lors de l'exécution du programme.
- En PYTHON le programme est *interprété* lors de son exécution, ligne par ligne.
- Beaucoup de langages utilisent un procédé hybride : le langage utilise une machine virtuelle, et les programmes sont compilés dans le langage de cette machine virtuelle, qui sera ensuite exécuté. La compilation peut être faite avant et le résultat stocké dans un fichier, ou bien être faite à la volée (*Just In Time compilation*).

Paradigmes

Chaque langage suit un ou plusieurs *paradigmes* de programmation qui change radicalement la manière d'écrire les programmes.

Paradigmes

Chaque langage suit un ou plusieurs *paradigmes* de programmation qui change radicalement la manière d'écrire les programmes.
Nous avons vu la programmation *impérative* (séquentielle, linéaire) où le programme effectue une liste d'instructions pas-à-pas. Il existe la programmation *évènementielle* lors de laquelle on précise à PROCESSING ce qu'il doit faire quand tel ou tel évènement se produit.

Paradigmes

Chaque langage suit un ou plusieurs *paradigmes* de programmation qui change radicalement la manière d'écrire les programmes.
Nous avons vu la programmation *impérative* (séquentielle, linéaire) où le programme effectue une liste d'instructions pas-à-pas. Il existe la programmation *évènementielle* lors de laquelle on précise à PROCESSING ce qu'il doit faire quand tel ou tel évènement se produit.
D'autres paradigmes de programmation telle la programmation *orientée objet* ou la programmation *fonctionnelle* sont très utilisés.

Contexte d'utilisation

Enfin on distinguera des différences dans les contexte d'utilisation de chaque langages : il existe des langages généralistes, des langages qui sont censés être exécutés sur des « serveurs de pages web » tels PHP, d'autres qui (à la base) ont été conçus pour être exécutés au sein même d'un navigateur (JavaScript).

Évolution au cours du temps

Évolution

Les progrès réalisés sur les analyseurs de code permettent de créer des langages avec une syntaxe de plus en plus épurée courte et pour lesquels il n'est pas obligé de déclarer le type de chaque variable.

Évolution

Les progrès réalisés sur les analyseurs de code permettent de créer des langages avec une syntaxe de plus en plus épurée courte et pour lesquels il n'est pas obligé de déclarer le type de chaque variable.
Les nouveaux langages s'inspirent souvent de leurs prédécesseurs.

Évolution

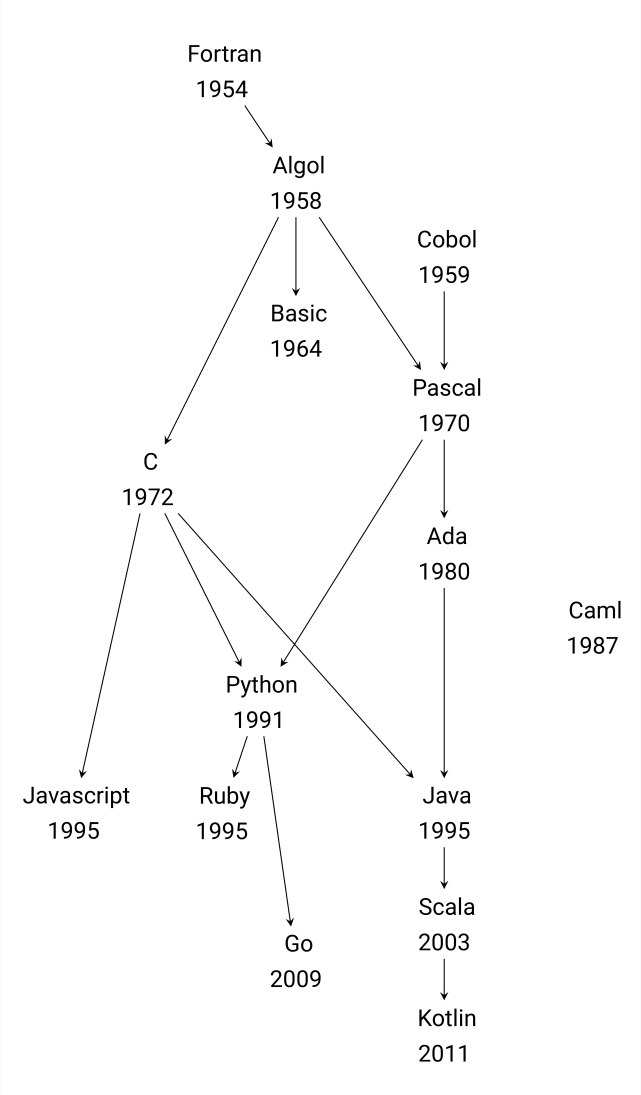
Les progrès réalisés sur les analyseurs de code permettent de créer des langages avec une syntaxe de plus en plus épurée courte et pour lesquels il n'est pas obligé de déclarer le type de chaque variable.
Les nouveaux langages s'inspirent souvent de leurs prédécesseurs.

On continue d'inventer de nouveaux langages : ceux-ci sont créés en fonction des besoins de l'époque.

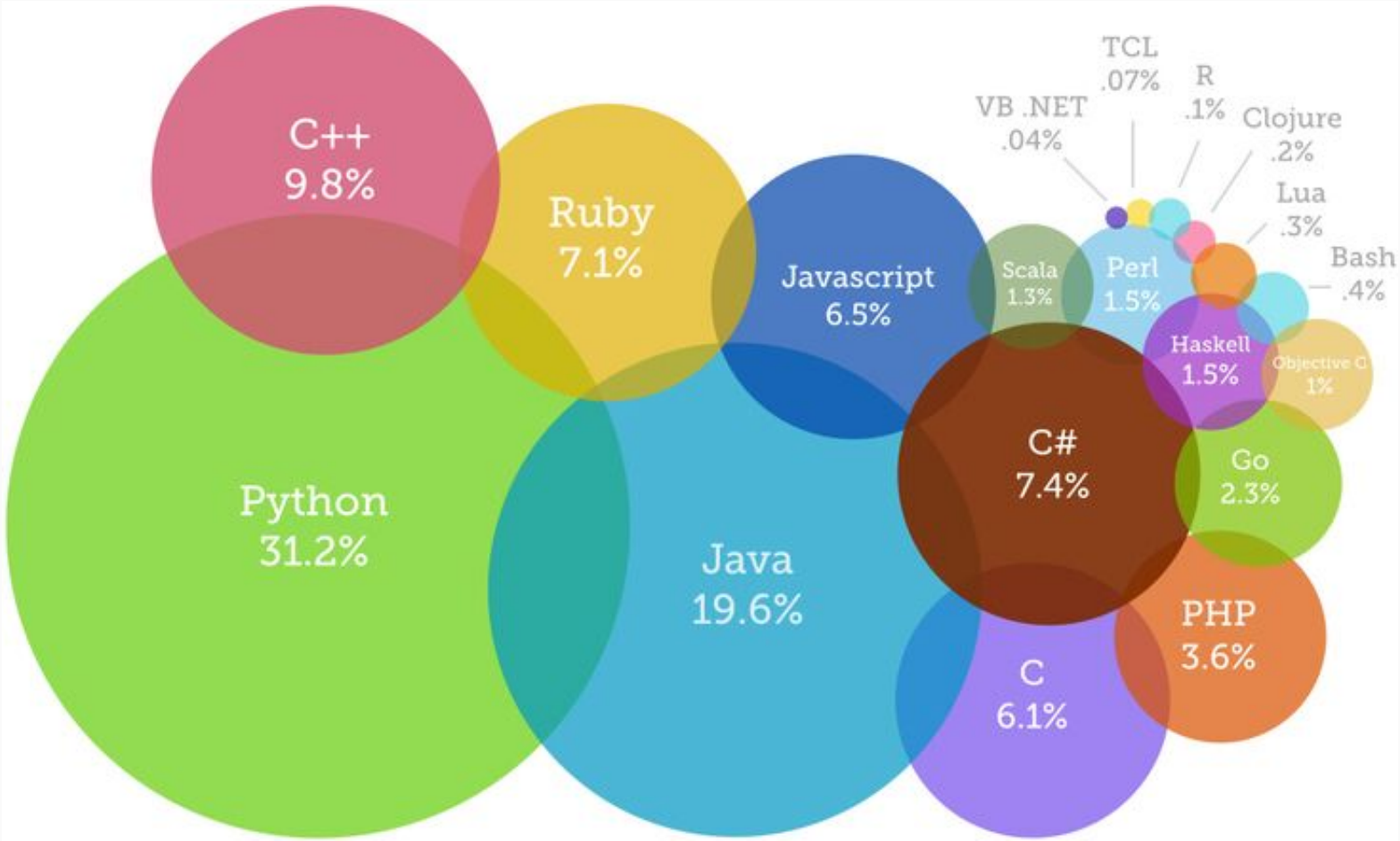
Évolution

Les progrès réalisés sur les analyseurs de code permettent de créer des langages avec une syntaxe de plus en plus épurée courte et pour lesquels il n'est pas obligé de déclarer le type de chaque variable.
Les nouveaux langages s'inspirent souvent de leurs prédécesseurs.

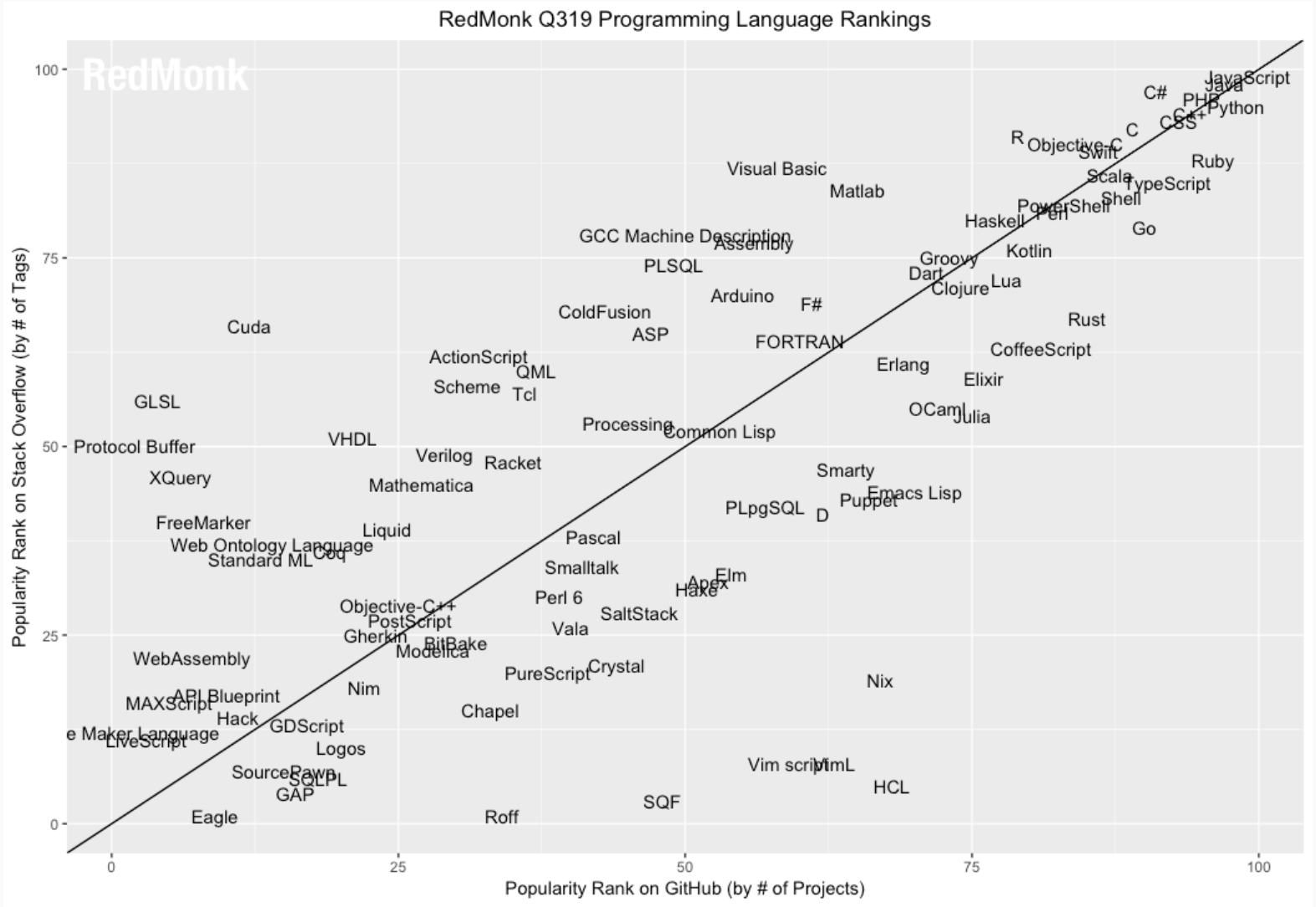
On continue d'inventer de nouveaux langages : ceux-ci sont créés en fonction des besoins de l'époque.
De nos jours, les langages qui permettent de programmer facilement plusieurs tâches simultanées (comme par exemple récupérer des données sur INTERNET et en même temps traiter ces données) ont le vent en poupe.



Voici un graphe indiquant comment les langages anciens ont influencé les nouveaux.



Langages les plus appris en 2019.
Source : `learnworthy.net`.



Stack Overflow est un forum d'entraide à la programmation.
GitHub est un service d'hébergement de projets logiciels.
Le graphique suivant indique la popularité de différents langages sur ces deux sites.
Source : `redmonk.com`.