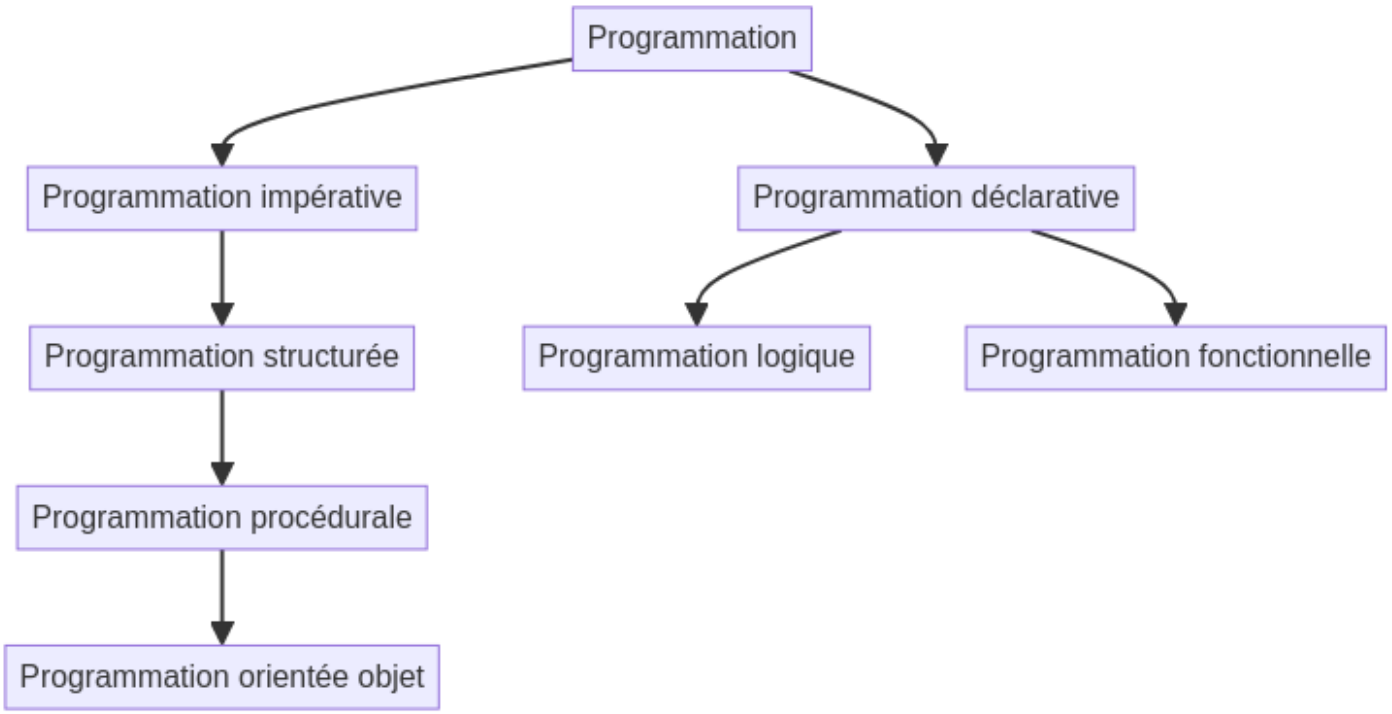
Les paradigmes de programmation
(ne se valent pas tous)

Il existe plusieurs façons d'aborder l'art de la programmation. On appelle cela des **paradigmes de programmation**.

Un peu à la manière des espèces vivantes qui descendent les unes des autres au fil de l'évolution, ces paradigmes forment un arbre d'évolution des techniques de programmation. Ils permettent ainsi de classer les différents langages de programmation. Ou du moins de les caractériser quand il n'est pas possible de les classer de façon stricte, car certains langages sont multiparadigmes à différents degrés (l'hybridation étant beaucoup plus courante dans le domaine des langages de programmation que dans celui des espèces vivantes).



Langages impératifs

Les langages impératifs sont les plus répandus, ce qui ne signifie malheureusement pas qu'ils soient les meilleurs. Ils sont dits impératifs car ils sont constitués de séquences d'instructions qui doivent être exécutées par la machine dans l'ordre dans lequel elles sont données (un peu à la manière des étapes d'une recette de cuisine qui indique **comment** réaliser un plat). On est effectivement dans un vocabulaire qui justifie le qualificatif d'impératif : « instructions », « exécuter », « dans l'ordre »…

Langages structurés

Avec les tous premiers langages de programmation, les programmes devaient être constitués d'une seule séquence monolytique d'instructions. Il est alors apparus que certains agencements d'instructions revenaient naturellement de

façon récurrente. Ces agencements étaient typiquement ceux qui permettent de tester une condition à laquelle une instruction doit être exécutée ou de répéter l'exécution d'un jeu d'instructions sur chacun des éléments constituant une liste de données.

Les langages structurés sont des langages qui en prennent acte et qui apportent des **structures de contrôle** permettant de faciliter l'expression de ces agencements récurrents. Ces structures sont notamment les mots clefs permettant d'exprimer des tests de conditions (`if`) ou différents types de boucles de traitement (`while`, `for` ou encore `do… until`) que l'on retrouve encore dans la totalité des langages de programmation impératifs modernes.

Langages procéduraux

L'étape suivant l'avènement de la programmation structurée fût un nouveau constat de récurrence de certaines séquences d'instructions sur des données issues d'un même ensemble. on pourrait par exemple imaginer un programme qui, une fois qu'il a établi les données relatives à une liste de personnes, doit pour chacune d'elles afficher dans la console son nom, son prénom et son âge.

La programmation structurée apporte déjà la notion de boucle qui permet, pour chacune des personnes de la liste, d'exécuter les trois instructions affichant pour la première le prénom, pour la deuxième le nom et pour la troisième l'âge. Mais imaginons maintenant qu'à un autre endroit du programme il faille réafficher le prénom, le nom et l'âge d'une personne en particulier, par exemple la plus agée de la liste.

On se retrouverait alors avec exactement la même séquence de trois instructions en deux endroits différents du code. Si elles obéissent aux mêmes spécifications fonctionnelles indiquant que c'est ainsi que l'on choisit de présenter à l'écran les informations d'une personne à chaque fois qu'il y a lieu de le faire, il faudra penser à effectuer des modifications à ces deux endroits du code si ces spécifications venaient à changer (par exemple si on décidait d'afficher la date de naissance plutôt que l'âge).

Il est donc apparu utile de pouvoir regrouper de telles instructions au sein d'unités de code qui pourraient ensuite être invoquées en différents endroits au moyen d'une seule ligne de code à chacun de ces endroits. Il suffirait alors de modifier le code de cette unité pour que ces modifications soient répercutées partout où elle est appelée.

Une telle unité de code est appelée une **procédure** et l'on a donc appelé les langages qui ont proposé cette innovation des langages **procéduraux**. Une procédure peut prendre des paramètres en entrée et retourner ou non une valeur en sortie, modifier ou non le contenu des données passées en paramètres ou encore réaliser ou non des opérations d'entrée et/ou de sortie (comme l'affichage à l'écran d'une information).

Dans notre exemple, il s'agirait de définir une procédure `afficher_personne` qui prendrait en paramètre une structure de données contenant toutes les informations relatives à une même

personne et qui se chargerait d'exécuter sur ces données les trois instructions d'affichage demandées.

Quand elles retournent une valeur, on appelle souvent ces procédures des « fonctions ». Mais ce terme est contesté par les puristes de la « véritable » programmation fonctionnelle, qui pointent certaines différences fondamentales entre le comportement de ces procédures retournant des valeurs et celui des « vraies » fonctions, au sens mathématique du terme (lesquelles ne sauraient modifier leurs paramètres d'entrée, provoquer des effets de bord ou encore réaliser des opérations d'entrée / sortie).

Des langages procéduraux notables sont le C, le Pascal ou encore le Basic.

Langages orientés objet

Il semble bien que c'est l'expérience[1] qui dicte l'évolution des langages impératifs. Chaque nouvelle génération se bâtit sur le constat de la récurrence de certains usages adoptés avec les langages de la génération précédente et propose de nouvelles syntaxes pour les rendre plus facile à exprimer.

Pour les langages orientés objet, le constat est cette fois ci que les meilleures pratiques adoptées par les programmeurs utilisant des langages procéduraux consistaient en définir des structures de données représentant des entités du domaine des problèmes à résoudre et à regrouper au sein de bibliothèques les procédures agissant sur ces structures de données. Cela revient à établir des associations pertinentes entre les données et les traitements procéduraux qui doivent opérer dessus.

Cette façon d'associer données et traitements est au cœur du concept de classe, l'élément le plus fondamental de la modélisation orientée objet et de la forme de programmation qui en découle.

Dans l'exemple précédent, introduit dans l'exposé concernant les langages procéduraux, la notion de personne est toute désignée pour constituer une classe dans un langage orienté objet. Cette classe constituera une unité de code qui permettra de regrouper tout ce qui concerne une personne, données (prénom, nom, date de naissance) comme traitements opérant spécifiquement sur ces données (par exemple calculer l'âge de la personne à partir de sa date de naissance).

Des langages orientés objet notables sont le Smalltalk, le C++, le Java, le Ruby ou encore le C#.

---

1. Ici l'expérience envisagée comme force motrice de l'évolution de la famille des langages impératifs peut être opposée à la théorie (notamment mathématique), qui pourrait être une autre fondation sur laquelle bâtir des langages de programmation. On peut d'ailleurs estimer que c'est le cas pour les langages déclaratifs, en particulier les langages logiques d'une part et les langages fonctionnels d'autres part, qui sont essentiellement les mises en œuvres directes de certaines théories mathématiques.

Langages déclaratifs

Là où les langages impératifs s'orientent vers une réponse à la question « comment ? », les langages déclaratifs s'intéressent plus à la question « **quoi** ? ». On n'y parle plus « d'instructions » comme dans les langages impératifs, mais plutôt « d'expressions ». Et là où les instructions des langages impératifs doivent être données dans l'ordre exact dans lequel elles doivent être exécutées, les expressions constituant les programmes déclaratifs permettent généralement à un « moteur d'exécution » de déduire le bon ordre dans lequel les opérations doivent être mises en œuvre.

L'informatique dans son ensemble n'est qu'une forme de mathématiques appliquées. Mais il est à noter que dans leur histoire et leur construction, les langages déclaratifs dont il va être question dans la suite de cet exposé sont plus directement et plus fortement liés à certaines théories mathématiques que ne le sont les langages impératifs. Cela est peut être dû au fait qu'en ne connaissant pas le même succès industriel précoce que les langages impératifs, ils se sont longtemps trouvés confinés dans le domaine de la recherche universitaire.

Langages logiques

Comme leur nom le laisse aisément penser, les langages de cette famille s'appuient sur cette branche des mathématiques qu'est la logique.

La logique est en quelques sortes la mise en équations de la vérité. La notion de `vrai` et de `faux` y sont les informations fondamentales, comme le sont les nombres en algèbre.

Les langages logiques permettent d'exprimer des faits (relativement simples) réputés vrais et des régles de déduction qui, combinées entre elles par des moteurs d'inférence, permettront de déterminer si d'autres faits (généralement plus complexes) sont vrais ou faux. Ou, en remontant le flux des déductions possibles, ils peuvent également permettre de déterminer à quelles conditions un fait exprimé dans le langage peut être vrai.

Ces propriétés les rendent spécialement adaptés à la réalisation de certaines formes d'intelligences artificielles et de systèmes experts.

Un langage logique notable est Prolog.

Langages fonctionnels

La branche des mathématiques sur laquelle s'appuient directement les langages de cette famille est celle de la combinatoire fonctionnelle. Cette discipline s'intéresse à la nature profonde des fonctions mathématiques, ce qui les constitue, quels sont leurs comportements « naturels », comment elles peuvent se construire par combinaisons des unes avec les autres (ou dans une démarche inverse se déconstruire en fonctions plus simples et plus fondamentales) et également comment certaines fonctions peuvent opérer sur d'autres fonctions.

On peut distinguer au sein de cette famille les langages fonctionnels purs, qui s'efforcent de fournir aux programmeurs des façons de définir des fonctions aux caractéristiques aussi proches que possible de celles que prédisent la

théorie et les langages fonctionnels impurs, qui s'autorisent pour des raisons pratiques à transiger sur certains éléments de la théorie (notamment la possibilité d'effets de bord).

Des langages fonctionnels impurs notables sont Lisp, ML, Caml, Scala ou encore Erlang.

Des langages fonctionnels purs notables sont Miranda ou encore Haskell (qui se trouve être le langage de programmation préféré de l'auteur de ces mots).