# La génération du code par l'IA

## Est-ce la fin des programmeurs ?

**24 Juin 2022**



claude@lemarson.com
https://www.lemarson.com

# Sommaire


***La génération du code assistée par l'IA***

- *Le codage est une technique, pas une science*
- *De la génération de texte à la génération du code*
- *Ne pas confondre indentation ou completion avec intelligence*
- *Les faiblesses du code produit : sécurité, performances, respect des patterns et frameworks, bizzareries...*
- *Ce que l'on peut raisonnablement espérer : un échéancier*
- *Quelques solutions intéressantes... et utiles, pour quels langages*
- *L'impact sur le métier des développeurs*
- *Changer nos habitudes...*

Quand le générateur s'appuiera sur 1 000 milliards de paramètres pour un seul langage, il sera difficile de lui résister…

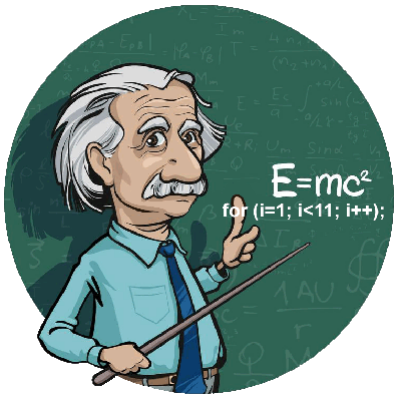
# La programmation n'est pas une science

- Sujet très sensible, qui divise la communauté…
- La conception d'un compilateur est une science, appliquer sa syntaxe est une technique.
- Les bonnes pratiques d'un langage se réfèrent à l'expérience. On sait ou on ne sait pas.
- Le risque est grand de se réfugier derrière l'expérience d'un langage et de la confondre avec la connaissance.
- Il est beaucoup plus facile de se spécialiser sur un langage que d'essayer de prendre du recul et de le repositionner dans son contexte.
- La « compétence » en matière de codage, ne se décrit plus de la même manière qu'avant.
- La véritable compétence se situe désormais chez les prestataires, Facebook, IBM, Google, pas chez les clients, chez qui elle devient inutile.




# La programmation n'est pas une science

- Le développeur applique des « patterns », des recommandations, comme un joueur d'échec dans une situation donnée cherche à reproduire ce qui a déjà fonctionné.
- Pour faire communiquer des objets ou séparer l'interface utilisateur, du traitement métier et de la restitution client (MVC). Pourquoi refaire moins bien, ce qui a fait ses preuves ailleurs.
- On est aujourd'hui dans un contexte de réutilisation systématique et de généricité des composants.
- On ne se lance pas dans une application, sans s'informer des meilleures API et composants réutilisables… Imposés par l'entreprise.
  Mais… s'approprier une API et des patterns ne confère pas à celui qui le fait, le statut de scientifique de haut niveau.
- **Le codage n'est pas une science. C'est une technique relativement simple, qui nécessite d'avoir de la mémoire et de savoir s'inspirer des bonnes pratiques déjà mises en œuvre…**
- Si on analyse le corpus d'une application, on s'aperçoit que plus de 80 % de son codage n'est qu'une mise en situation de composants, de patterns et de frameworks qui existent déjà.



# Trois époques pour l'assistance au codage

**Assistance à la saisie** (depuis les années 80)
- De nombreux éditeurs dédiés à un ou plusieurs langages.
- Autocompletion, suggestion de noms de variables, de fonctions, de méthodes ou de classes, à partir des premiers caractères tapés, etc.
- Se limitent à la syntaxe des langages et à leur organisation.

**Analyse statique** (période actuelle)
- Analyse statique du code pour détecter des anomalies, sans avoir à l'exécuter.
- Deux formes d'analyse statique : de base ou avancée.
  - Analyse de base : l'outil effectue une analyse linéaire, comme le ferait un correcteur orthographique et recherche des erreurs le plus souvent syntaxiques, en fonction du langage retenu.
  - Analyse avancée : consiste à extraire des informations du code, des valeurs possibles de variables, des chemins d'exécution, etc, pour en déduire des anomalies potentielles.

**Génération du code en 2 phases**
- Débuts de la génération du code : on fait confiance aux outils et IDE pour générer un code de qualité et l'intégrer dans des ensembles plus vastes : à partir de 2025
- Génération intelligente complète et "aveugle" à partir d'un descriptif textuel, modèle ou vocal (on ne contrôle plus) : pas avant 2040

**Epoque 1**
**(70-80)**

Autocompletion, Proposition de noms de variables, méthodes, classes, fonctions...

**Epoque 2**
**(80-2020)**

Outils d'analyse du code source pour détecter les anomalies potentielles : sécurité, performances, bugs

**Epoque 3**
**Actuelle et future**

Proposition "intelligente" de code à partir de commentaires, de noms de fonctions ou méthodes (Copilot)



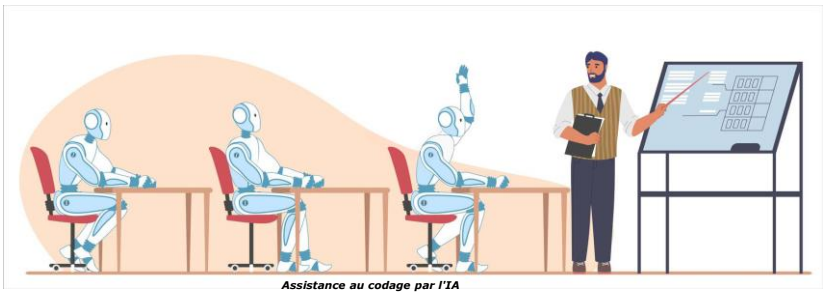
# La génération n'est pas nouvelle

- De nombreuses techniques permettent de générer du code.
- Une méthode courante est de partir d'un modèle UML, un diagramme de classe statique et de générer le code de l'application.
- Autre solution avec un pseudo-code (LDA : Langage de Description d'Algorithmes).
  - Dans la pratique il est aussi difficile d'apprendre un pseudo-langage que le langage cible… ça ne sert pas à grand-chose.
- Moteurs de règles : on définit et on code une fois pour toutes les règles de gestion, que l'on assemble par un pseudo-code.
- Low Code.
- Transpilers : compilateurs de source à source.

```
Fonction factorielle de n
        r = 1
        Pour i allant de 1 à n avec un incrément de 1
                r = r*i
        Fin pour
        Renvoyer r
Fin fonction
```

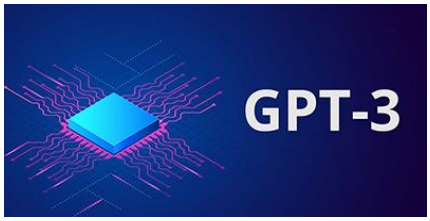
```
def factorielle(n):
        r = 1
        for i in range(1,n+1):
                r = r*i
        return r
```

D'un pseudo-code au code



# GPT-3, un générateur de texte surpuissant

- GPT-3, un réseau neuronal de génération de textes, créé par OpenAI pour « écrire » des articles, traduire des livres, décoder intelligemment les réponses à une requête libre en BI…
- GPT-3 est une API de la famille des outils de traitement du langage naturel, capable de comprendre une demande et de de fabriquer des textes cohérents, plus ou moins longs
- GPT-3 fonctionne en deux étapes : entraînement et inférence.
  - Le réseau neuronal a été entraîné avec un grand nombre de textes, venant d'Internet, des réseaux sociaux, de Common Crawl, de manière à ce qu'il s'habitue à la présence et à la fréquence des mots, qu'il comprenne les liens sémantiques pouvant exister entre eux, etc.
  - On dit qu'il fonctionne en « autocompletion », une technique qui consiste à ce qu'on lui propose une série de caractères, à charge pour lui de la compléter, par des mots judicieux. Et « in fine » de fabriquer des phrases, par répétition de cette étape.

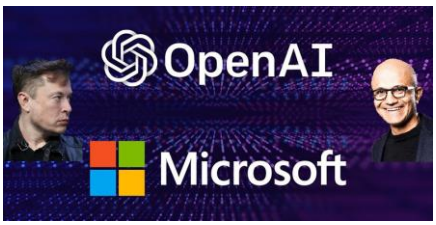



- GPT-3 fonctionne en mode question-réponse, l'algorithme jouant sur son expérience pour compléter les phrases proposées dans le « prompt », avec un ou plusieurs mots.
- Si on lui soumet la question :
  - *Je voudrais assister au prochain match de hockey à Montréal du…*
  - Il répondra :
  - *Je voudrais assister au prochain match de hockey à Montréal du canadien*
  - Car il connaît le nom de l'équipe de cette ville et que la probabilité est grande qu'il s'agisse du canadien. Ce qu'il a appris en parcourant les nombreux textes qui lui ont été soumis dans sa phase d'apprentissage.
  - Le mot *canadien* a été déterminé en apprentissage et il lui a été attribué une probabilité conditionnelle pour qu'il apparaisse dans un certain contexte. En d'autres termes, GPT-3 a appris l'existence de *canadien*, mais en plus, il sait désormais le distinguer par une probabilité, sa capacité à apparaître dans un texte précis.
  - On pourra poursuivre l'interrogation et rechercher les mots qui devraient logiquement suivre *canadien*. C'est ici que se forment les phrases, elles-aussi issues de l'expérience de l'algorithme.
  - *Je voudrais assister au prochain match du canadien, cette remarquable équipe créée en 1909 et qui a remporté 83 fois la coupe Stanley*
  - Ce qui est vrai et faux à la fois, l'algorithme pouvant se tromper, comme n'importer quel être humain…



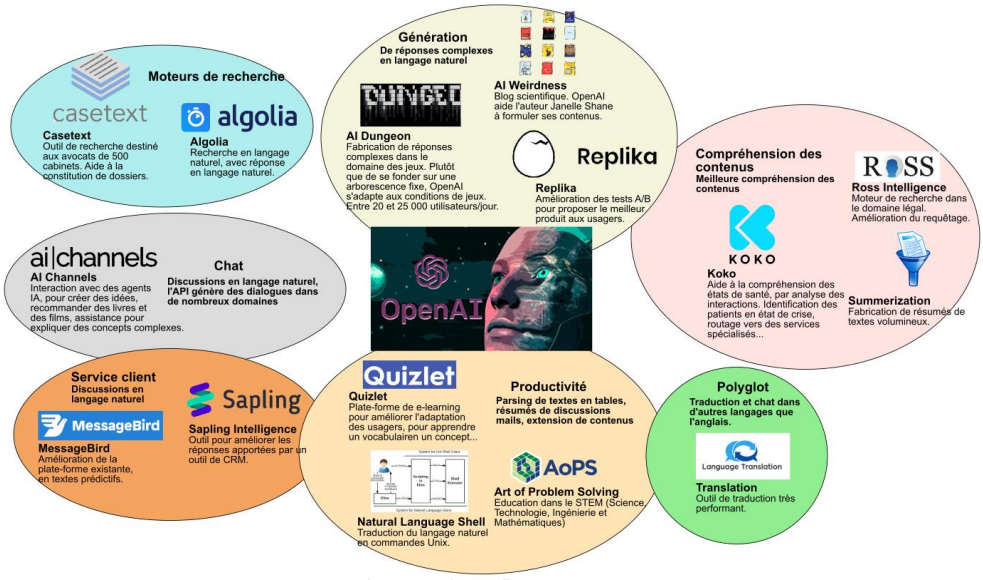
# GPT-3, un générateur de texte surpuissant

- GPT-3 peut aller au-delà du simple remplissage par des données factuelles et réelles.
- On peut lui exprimer notre admiration pour le poète Arthur Raimbaud, par la question :
  - *Le « dormeur du val » d'Arthur Raimbaud commence par*
  - A laquelle GPT-3 peut répondre par :
  - *Le « dormeur du val » d'Arthur Raimbaud commence par « C'est un trou de verdure où chante une rivière »*
  - Mais GPT-3 peut aller plus loin et nous suggérer un poème imaginaire, qu'il aura composé, en tenant compte des caractéristiques d'écriture de l'écrivain, du rythme de ses vers, etc.
  - *En 2021, Rimbaud aurait écrit « La tragédie du covid » et commencé par « Il dort devant sa machine, la main sur la poitrine »*
- C'est cette capacité à s'étendre et donc de construire des corpus textuels qui est utilisée, GPT-3 pouvant produire des textes parfaits en termes de syntaxe, alimentés par des données que nous n'aurions pas eu l'idée d'aller chercher… Et donc des programmes, après une phase de spécialisation par apprentissage.



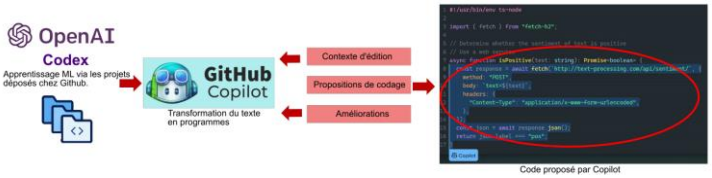

- A la fin 2020, Microsoft et OpenAI ont signé un accord qui fait de Microsoft le « parrain » de la technologie, dont il bénéficie désormais en exclusivité.
- Microsoft place en même temps son Cloud Azure, qui devient l'hébergeur officiel de GPT-3, pour lequel il a construit un HPC (« High Performance Computer »).
- Microsoft utilisera GPT-3, sans doute en l'améliorant, pour ses propres besoins et introduira de l'IA dans ses outils, à commencer par Office.

# Les applications concrètes de GPT-3





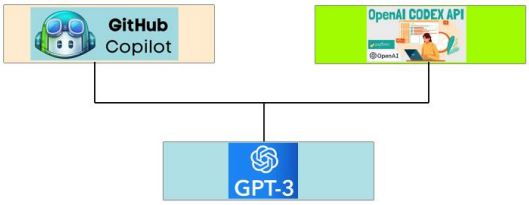

# Copilot, l'usage de GPT-3 dans le codage





Code proposé par Copilot

- Copilot, outil d'assistance à la programmation de la mouvance OpenAI
- Microsoft a investi 1 milliard $ dans OpenAI, dont il est le partenaire exclusif.
- Copilot "fabrique" du code à partir :
  - Des commentaires
  - Du nom de fonctions ou méthodes
- Entraîné avec GPT-3




Copilot et Codex d'OpenAI sont entraînés sur GPT-3.
Copilot génère un code de programmation à partir d'une description de haut niveau, alors qu'on accède à Codex uniquement par une API

# Comment dialoguer avec Copilot

Commentaires : on exprime ce que l'on veut et Copilot détecte les pistes d'écriture

```
# generate 20 random numbers
```

Copilot génère le code :

```
import random
for i in range(20):
print (random.randint(1, 100))
```

Autre solution : taper un nom de fonction significative et laisser Copilot se débrouiller pour la suite :

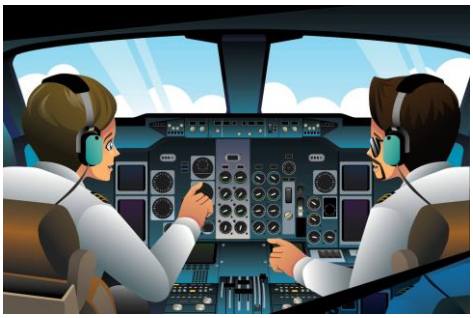
```
def max_sum_slice(xs) :

if not xs:
     return 0
max_ending = max_slice
for x in xs :
     max_ending = max(0, max_ending + x)
     max_slice = max(max_slice, max_ending)
return max_slice
```

On pourra encore commencer à écrire le code et à court d'inspiration, laisser Copilot le continuer, voire le terminer. Ce sera le cas le plus fréquent.



# Copilot n'est pas (encore) crédible

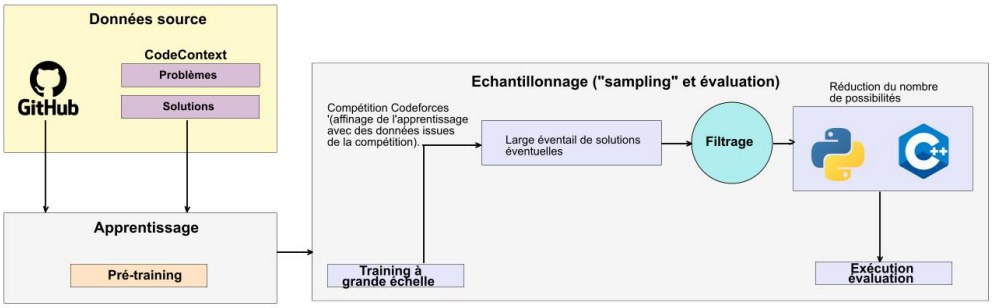
- Tendance à être fantaisiste. **Quand il n'est pas totalement nul**… à faire frémir un programmeur débutant.
- Ses principales limites viennent du **contexte dans lequel il doit fonctionner**. Si celui-ci est complexe, avec des API en grand nombre et des architectures avancées, il aura des comportements "curieux".
- Dans le domaine de la sécurité : résultats d'une étude récente qui s'est intéressée à ce que produirait Copilot dans le contexte des vulnérabilités décrites par Mitre, les "2021 CWE Top 25 Most Dangerous Software Weaknesses"). Les auteurs se sont demandés si dans des scénarii propres à générer ces vulnérabilités, Copilot serait meilleur ou moins bon que le "commun des mortels".
- Sur 1692 programmes générés, 40 % comportaient des vulnérabilités toutes aussi dangereuses. Ce que les chercheurs ont expliqué en rappelant que le processus d'apprentissage ML ne les mettait pas à l'abri de reproduire les anomalies présentes dans les "repositories" de Github.



En pilotage automatique, les pilotes surveillent les paramètres. Théoriquement les algorithmes informatiques sont plus sûrs que le pilotage manuel. Mais les failles peuvent provoquer des catastrophes…

# AlphaCode de DeepMind (Alphabet Google)

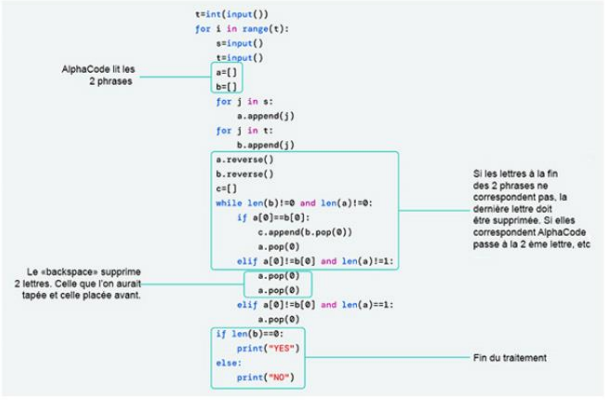
- DeepMind : compagnie britannique rachetée par Google en 2010
- Objectif : "trouver les meilleures techniques d'apprentissage automatique pour construire des algorithmes puissants d'apprentissage généraliste".
- AlphaCode a été confronté à 5 000 concurrents dans 10 concours : il s'est globalement situé dans la moyenne des programmeurs experts.






# AlphaCode de DeepMind (*)

```
// RATING: 1200
// TAGS: *special, implementation
// LANGUAGE IS python3
// CORRECT SOLUTION
// Vous indiquez 2 chaînes de car s et t, en minuscules anglaises
// Vous tapez la chaîne s, caractère par caractère, du premier au dernier.
// Quand au lieu de taper un caractère, vous faites un "backspace",
// vous supprimez le lerniuer car que vous avez tapé, parmi ceux
// qui ne l'ont pas été (ou rien s'il n'y a plus de caractères dans
// la chaîne courante).
// Ex : si s est "ababa" et que l'on fait un backspace au lieu de taper le
// 1er et 4 ème caractère, vous obtiendrez la chaîne "ba" (le premier
// "backspace" ne supprime rien et le deuxième supprime le caractère "c).
// Le code demandé à AlphaCode consiste à obtenir une chaîne t en tapant
// s et en se servant des "backspaces".
// Aucun moyen d'obtenir "bb" avec "ababa".
// Pour obtenir "ababa" avec s = "aababa", il faut faire un "backspace"
// sur le premier caractère et taper les autres normalement.
```




**(*) Cet exemple est fourni par DeepMind**

# Les avantages actuels

- Un gain de temps évident, même avec des API et frameworks que l'on ne domine pas.
- Des idées, originales ou non, auxquelles on n'aurait pas pensé.
- Une adaptation souvent étonnante à la "plomberie" : habitudes, défauts, "façons" de faire, qui nous caractérisent.
- Une aide évidente aux tâches énervantes : documentation… On peut se concentrer su ce qui constitue la valeur ajoutée. On peut toujours rêver…
- Même si le code généré était plus pertinent qu'il ne l'est, cela n'empêchera pas les développeurs de devoir les analyser, pour le comprendre et ensuite pour évaluer son intérêt. Avant de pouvoir lui faire confiance totalement. Ce qui est particulièrement chronophage.
- Tant que nous resterons dans cette incertitude, au milieu du gué, **un bon développeur mettra moins de temps à fabriquer ses applications, sans générateur, qu'avec générateur**…




# La légitimité du processus

- Nombreuses polémiques…
- L'apprentissage des algorithmes peut entraîner des violations de propriété :
  - Copie de séquences de code qui ne sont pas libres
  - Risque de blanchiment d'un code récupéré par couper-coller à partir d'une source sous licence : portée généralement faible, car individuellement ne concerne que quelques dizaines de lignes
  - Certains produits sont payants… et fondés en partie sur un code Open Source
- Copilot
  - Ses concepteurs affirment que moins de 0,1 % des programmes générés peuvent contenir du code protégé (ex de la licence Copyleft) : c'est très optimiste.
  - C'est le principe de l'apprentissage qui est attaqué : la FSF ("Free Software Foundation") de Richard Stalmann, estime que l'apprentissage par réseaux neuronaux n'est pas une technologie "loyale", qui préserve les intérêts des concepteurs d'origine, ni les droits d'auteurs.
  - En termes de productivité, l'objectif est de gagner du temps, tout en étant performant.
  - Si on génère des pans entiers de code, on n'aura plus à les écrire et tout le monde sera content.
  - Sauf que c'est illusoire.

La génération automatique de code se heurte à la législation et aux licences Open Source.

# Quelques alternatives

bayou

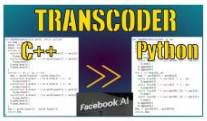
Générateur Java et Python. Utilie les techniques de "Neural Sketch Learning".

cogram

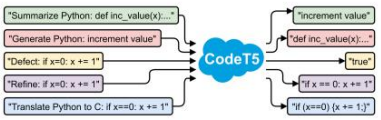
Pour les programmeurs Python qui utilisent des requêtes SQL. L'expression du besoin est en mode libre, qu'il traduit en SQL.

PolyCoder
An Open Source Code Generator

Alternative Open Source à Codex. Développé à Carnegie Mellon, basé sur GPT-3, entraîné sur 249 GB de code écrits avec 12 langages.




Transpiler écrit par une équipe de chercheurs de Facebook. Transpiler qui traduit les souorces entre C++, Python et Java. Entraîné sur les exemples GitHub.




CodeT5 a été écrit par les chercheurs de SalesForce. Fondé sur T5 de Google (Text-to-Text Transfer Transformer) sur GitHub. Autocompletion et génération du code.

NVIDIA.

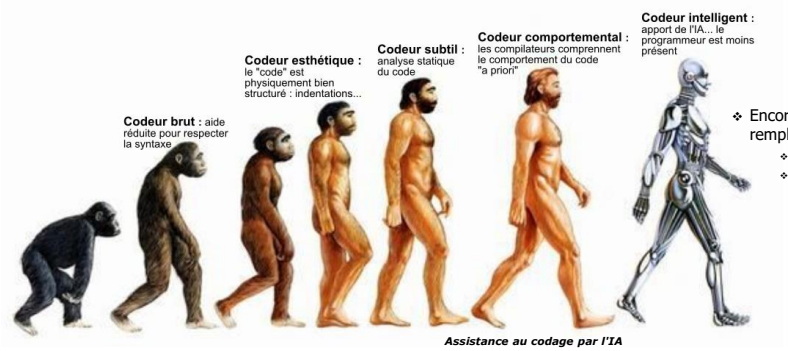
Megatron-Turing de NVidia. Transformateur de textes sources anglais. Fondé sur 530 G paramètres.



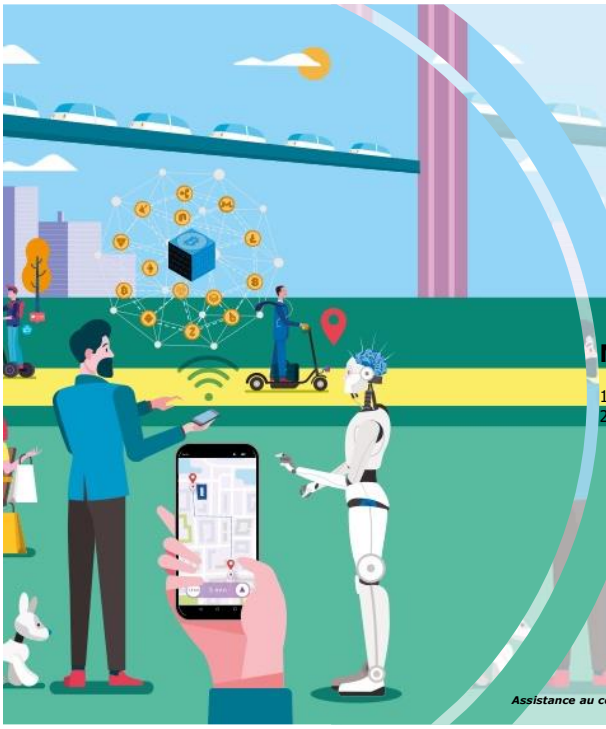
# Le métier de programmeur

## Changer nos habitudes...

- L'assistance IA au codage continue de se déployer, mais ce n'est pas de la génération.
- Disposer d'un programmeur "pair" qui regarde par-dessus notre épaule, nous alerte quand nous nous trompons dans la syntaxe, nous suggère de bonnes idées, sans chercher à nous remplacer, est aussi incontournable qu'un correcteur orthographique ou un GPS.
- S'arrêter au fait que l'assistant est mauvais, qu'il génère un code de débutant, qu'il viole les règles en vigueur, c'est faire un mauvais calcul.
- De la même manière que les voies de chemin de fer ont été électrifiées, les générateurs de demain seront plus efficaces et contextuels. Ce qui ne veut pas dire que tout le monde les adoptera.
- Fin du métier de codeur : en grande partie.
- Fin du métier de concepteur : NON, tant que l'IA restera stupide.
- Ces métiers vont changer, mais pendant quelques années, il y aura encore des personnes qui trouveront amusant d'écrire du code et seront plus efficaces qu'avec les outils d'IA.
- L'avenir appartient aux générateurs, c'est certain… mais il faut laisser du temps au temps…




- Encore impensable d'imaginer le "grand" remplacement des codeurs par des algorithmes.
  - L'IA restera longtemps une aide contextuelle
  - La vraie révolution interviendra quand un algorithme d'IA sera capable de dialoguer avec un autre algorithme, pour lui faire générer du code : horizon 2040



# La génération du code par l'IA

## Est-ce la fin des programmeurs ?

### Nos prochains webinaires

| | |
|---|---|
| 1 Juillet 2022 : | **La programmation du comportement des réseaux (IBN)** |
| 2 Septembre 2022 : | **Le "tout en un" de l'hyperconvergence** |

claude@lemarson.com
https://www.lemarson.com