

ChatGPT remplacera-t-il les programmeuses et programmeurs ?

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 3 janvier 2023

<https://www.bortzmeyer.org/chatgpt-programmation.html>

Si Ada Lovelace était la première programmeuse, les programmeuses et programmeurs actuelles sont-elles les dernières ? Un programme « intelligent » comme ChatGPT va-t-il les condamner à la reconversion ou au chômage <<https://twitter.com/FelixMaHass/status/1604439627187748866>> ? Je continue ici mon exploration de ChatGPT <<https://www.bortzmeyer.org/chatgpt.html>>.

ChatGPT a beaucoup été utilisé pour la programmation <<https://twitter.com/HomerSigma/status/1606326904054767617>>. Voici une session où je lui demande des programmes (ce sera la seule image, dans le reste de l'article, je copierai le texte).

Exemple d'utilisation :

Le programme est écrit en Elixir car mes questions précédentes portaient sur ce langage et ChatGPT mémorise les conversations pendant une session (et peut-être au-delà donc ne lui donnez pas d'informations confidentielles). L'utilisation d'Elixir montre que ChatGPT connaît beaucoup de choses, y compris des langages assez rarement utilisés. C'est logique : pour le quantitatif (la mémorisation et la restitution d'énormes quantités d'informations), les ordinateurs sont certainement meilleurs que les humains. D'autre part, celles et ceux qui ont lu mon précédent article <<https://www.bortzmeyer.org/chatgpt.html>> noteront que j'ai maintenant un compte ChatGPT, ayant trouvé un numéro de téléphone sacrificable.

Beaucoup de gens ont été impressionnés par les qualités de ChatGPT <<https://twitter.com/MothNoises/status/1606296583376494592>> et, voyant la facilité avec laquelle il génère des programmes (et les explique!), se sont dit que les programmeuses et les programmeurs allaient disparaître. Un tel discours fait rêver bien des patrons, qui apprécieraient de remplacer des développeurs et développeuses payés chers par une IA docile et obéissante. Ce discours est donc très ancien dans l'informatique, Cobol avait déjà été promu avec cet argument. Et c'était également le cas de Fortran (« *FORTRAN should virtually eliminate coding and debugging* » <<https://www.softwarepreservation.org/>>

[projects/FORTRAN/BackusEtAl-Preliminary%20Report-1954.pdf](#)> »). Même si ces affirmations semblent techniquement ridicules aujourd’hui, à l’époque, le progrès par rapport au langage d’assemblage était tel que cela ne semblait pas absurde. Puis on a entendu à nouveau ce raisonnement du temps des L4G puis récemment du nocode. Le fait que les espérances des patrons aient été déçues à chaque fois ne les empêche pas de garder espoir.

Voyons donc ce que sait faire ChatGPT. D’abord, on l’a dit, il connaît beaucoup de choses. Je demande « Je dois trier une liste de chaînes de caractères [un exercice banal de débutant]. Comment faire? » et ChatGPT répond « Pour trier une liste de chaînes de caractères en Elixir, vous pouvez utiliser la fonction `Enum.sort/1`. Cette fonction prend en entrée une liste et renvoie une liste triée de ses éléments. Voici un exemple de code qui trie une liste de chaînes de caractères en ordre croissant : »

```
strings = ["c", "b", "a", "d"]
sorted_strings = Enum.sort(strings)
IO.inspect(sorted_strings) # ["a", "b", "c", "d"]
```

Il connaît la bibliothèque standard d’Elixir (où il y a bien un `Enum.sort`) et aussi des idiosyncrasies <https://fr.wiktionary.org/wiki/idiosyncrasie> d’Elixir comme le fait de nommer une fonction en terminant par son arité. Et il ajoute spontanément « Si vous voulez trier la liste en ordre décroissant, vous pouvez utiliser la fonction `Enum.sort/2` et lui fournir une fonction de comparaison qui renvoie `:gt` si le premier argument est plus grand que le second, `:lt` si le premier argument est plus petit que le second, et `:eq` s’ils sont égaux. Voici un exemple de code qui trie une liste de chaînes de caractères en ordre décroissant : »

```
strings = ["c", "b", "a", "d"]
sorted_strings = Enum.sort(strings, fn x, y -> if x > y do :gt else if x < y do :lt else :eq end end)
IO.inspect(sorted_strings) # ["d", "c", "b", "a"]
```

(Les `:gt` et `:lt` sont appelés atomes et sont une des particularités d’Elixir. Je vous ai dit qu’il « connaissait » bien le langage.)

Mais ce second code pose déjà des problèmes. D’abord, il ne compile même pas (il manque un `end`). Mais, surtout, ChatGPT n’a pas bien lu la documentation <https://hexdocs.pm/elixir/Enum.html#sort/2> : la fonction passée en argument à `sort` est censée renvoyer un booléen, pas ces atomes. On voit là une première chose à garder en mémoire : on ne peut pas en général utiliser les programmes produits par ChatGPT tels quels <https://twitter.com/benoitfrenay/status/1608379693811601409>, il faut les vérifier, et cette vérification doit être faite par une personne compétente en programmation. Et il n’est pas évident que l’entièreté du processus soit plus rapide et plus sûr que si cette personne compétente avait tout fait elle-même. (L’erreur de syntaxe est la moins grave, car elle est détectée à la compilation. Le pire serait une erreur qui ne planterait pas mais donnerait un résultat faux.)

Ceci dit, ChatGPT est interactif. Voyons cela sur un autre exemple : « Comment décomposer un entier de grande taille en ses facteurs premiers? ». ChatGPT répond « Pour décomposer un entier en ses facteurs premiers en Elixir, vous pouvez utiliser une approche récursive. Voici comment vous pouvez implémenter cette approche : »

```
defmodule PrimeFactors do
  def factors(n, divisor \\ 2) when divisor > n do
    [n]
  end
  def factors(n, divisor \\ 2) do
    if rem(n, divisor) == 0 do
      [divisor | factors(n / divisor, divisor)]
    else
      factors(n, divisor + 1)
    end
  end
end
```

Cette fois, il n'y a pas d'erreur de syntaxe, mais cela ne compile quand même pas, en raison d'une erreur sémantique. Et il y a deux autres bogues (trouver l'une d'elles nécessite de connaître Elixir). Mais comme ChatGPT est interactif, je lui explique : « Attention, on ne peut pas avoir deux définitions avec des valeurs par défaut. Il faut réécrire ton code. » Il me répond que j'ai raison (d'autres personnes ont vu ChatGPT nier des problèmes) et « Vous avez raison, il n'est pas possible d'avoir deux définitions avec des valeurs par défaut pour le même nom de fonction en Elixir. Pour résoudre ce problème, vous pouvez utiliser une seule définition de fonction avec une valeur par défaut pour le diviseur, et ajouter une condition pour gérer le cas où le diviseur est supérieur à n. Voici comment vous pouvez réécrire le code : »

```
defmodule PrimeFactors do
  def factors(n, divisor \\ 2) when divisor > n do
    [n]
  end
  def factors(n, divisor \\ 2) do
    if divisor > n do
      [n]
    else
      if rem(n, divisor) == 0 do
        [divisor | factors(n / divisor, divisor)]
      else
        factors(n, divisor + 1)
      end
    end
  end
end
```

Et il ajoute spontanément « Je vous recommande également de changer le nom de la fonction `factors/2` en `factors/1` et de changer la condition de la clause de cas `factors/1` pour qu'elle vérifie si n est inférieur ou égal à 1. Voici comment le code final devrait ressembler [sic] : »

```
defmodule PrimeFactors do
  def factors(n, divisor \\ 2) when n <= 1 do
    []
  end
  def factors(n, divisor \\ 2) do
    if rem(n, divisor) == 0 do
      [divisor | factors(n / divisor, divisor)]
    else
      factors(n, divisor + 1)
    end
  end
end
```

C'est amusant, le code réécrit a toujours la même bogue sur la définition de fonctions avec des valeurs par défaut. Par contre, ChatGPT a « réfléchi » et a corrigé une des autres bogues (qui faisait que le nombre 1 était listé comme facteur premier). Bon, tant pis, je corrige manuellement et je lui signale la troisième bogue (celle qui nécessite de connaître Elixir) : « Et attention, le / fait une division de réels, il faudrait utiliser div. ». Il le reconnaît : « Vous avez raison, la fonction / en Elixir effectue une division de réels et non une division entière. Pour effectuer une division entière, vous pouvez utiliser la fonction div/2. Voici comment vous pouvez utiliser la fonction div/2 dans la fonction factors/1 : »

```
defmodule PrimeFactors do
  def factors(n, divisor \\ 2) when n <= 1 do
    []
  end
  def factors(n, divisor \\ 2) do
    if rem(n, divisor) == 0 do
      [divisor | factors(div(n, divisor), divisor)]
    else
      factors(n, divisor + 1)
    end
  end
end
```

À noter que ChatGPT avait montré qu'il connaissait l'éco-système Elixir, en expliquant comment tester avec l'interpréteur iex :

```
iex> PrimeFactors.factors(12)
[2, 2, 3]
iex> PrimeFactors.factors(2310)
[2, 3, 5, 7, 11]
iex> PrimeFactors.factors(7829)
[7829]
```

Prenons un autre exemple qui illustre la question du cahier des charges. En programmation dans le monde réel, c'est souvent la principale difficulté; le-a programmeur-se passe souvent davantage de temps à comprendre un cahier des charges incomplet, contradictoire et irréaliste qu'à programmer. Commençons simplement : « Écris un programme en Python qui donne la liste des nombres allant jusqu'à N. ». ChatGPT répond :

```
# Demande à l'utilisateur de saisir un entier N
n = int(input("Entrez un entier N : "))

# Initialise une liste vide
l = []

# Pour chaque entier i allant de 1 à N
for i in range(1, n+1):
    # Ajoute i à la liste
    l.append(i)

# Affiche la liste
print(l)
```

Et explique « Ce programme demande à l'utilisateur de saisir un entier N, initialise une liste vide, puis utilise une boucle for pour ajouter chaque entier allant de 1 à N à la liste. Enfin, il affiche la liste. ». Modifions le cahier des charges (ce qui est courant) : « Il faut que cela marche avec des entiers relatifs. » ChatGPT ne se démonte pas pour si peu « Pas de problème, voici une version modifiée du programme qui prend en compte les entiers relatifs : »

```
# Demande à l'utilisateur de saisir un entier N
n = int(input("Entrez un entier N : "))

# Initialise une liste vide
l = []

# Si N est positif
if n > 0:
    # Pour chaque entier i allant de 1 à N
    for i in range(1, n+1):
        # Ajoute i à la liste
        l.append(i)

# Si N est négatif
elif n < 0:
    # Pour chaque entier i allant de N à -1 (inclus)
    for i in range(n, 0):
        # Ajoute i à la liste
        l.append(i)

# Si N est égal à 0
else:
    # Ajoute 0 à la liste
    l.append(0)

# Affiche la liste
print(l)
```

Et explique « Cette version vérifie si N est positif, négatif ou égal à 0 et utilise une boucle for appropriée pour ajouter les entiers à la liste. Si N est égal à 0, 0 est simplement ajouté à la liste. » On voit que ChatGPT est parfaitement capable de faire des exercices de programmation pour grands débutants, par exemple au lycée. Mais comme je suis méchant, je lui demande ensuite un programme qui marche avec les nombres réels, ce qui est impossible. Il me fait alors une variante qui énumère, mais avec un pas (qu'il a choisi arbitrairement) de 0,1. Dois-je le considérer comme très bête ou comme très intelligent d'avoir interprété de manière créative un cahier des charges absurde? En tout cas, cet exemple montre à quoi ressemble souvent le travail de programmation : il faut de nombreuses itérations avant d'avoir une idée claire de ce qui est demandé et un programme comme ChatGPT n'est pas forcément le mieux placé pour cela. (Mes lectrices qui connaissent la programmation auront déjà fait remarquer "*in petto*" que si on ne peut pas énumérer les réels - qui sont en nombre infini dans tout intervalle, quelle que soit sa taille, on pourrait par contre énumérer les nombres en virgule flottante représentables sur la machine. Mais ChatGPT n'y a pas pensé.)

Mais, bref, on est encore loin d'un système qui pourrait permettre de se passer complètement des développeurs et développeuses. Il peut aider, c'est certain, faisant gagner du temps lorsqu'on a oublié comment faire une requête HTTP dans tel ou tel langage de programmation, il peut servir d'« auto-complétion améliorée » <https://mastodon.recherche-et-cie.org/@celliern/109627499278221711> » mais cela ne permet pas de lui confier tout le travail. Alors, certes, il s'agit d'Elixir, mais est-ce que cela marcherait mieux avec des langages de programmation plus répandus? J'en doute. Mémoriser une grande quantité d'informations et pouvoir en ressortir à volonté est justement le point fort des ordinateurs. Contrairement à un humain, apprendre un langage de plus ne nécessite guère d'effort. Les différents messages lus sur les réseaux sociaux semblent indiquer que des langages bien plus communs qu'Elixir ont le même problème (pareil pour la simple algèbre <https://twitter.com/zorky/status/1608136602412322816>).

Parmi les autres faiblesses de ChatGPT qui font obstacle à une utilisation en totale autonomie, notons aussi :

<https://www.bortzmeyer.org/chatgpt-programmation.html>

- ChatGPT n'est pas réflexif, il ne s'analyse pas lui-même et n'est pas capable de donner un degré de certitude sur ses « solutions ». Il affirme souvent des choses fausses avec le même niveau de confiance que les choses les plus indiscutables. Et il **invente** souvent des choses qui n'existent pas (articles scientifiques <<https://twitter.com/MonniauxD/status/1611849617221775361>>, options de logiciel <<https://mastodon.gougere.fr/@bortzmeyer/109632212819684522>>, etc).
- Certains promoteurs de ChatGPT disent ne pas s'inquiéter de la mauvaise qualité (parfois) du code en considérant que, du moment que « ça marche », c'est suffisant. Mais cet argument ne vaut pas pour la sécurité. Si un programme a une faille de sécurité, il va marcher, mais être néanmoins dangereux.

En conclusion, oui, ChatGPT et les logiciels similaires vont changer la programmation (comme Cobol et Fortran l'avaient fait) mais croire qu'ils vont permettre de se passer de programmeur-ses est illusoire <<https://twitter.com/HomerSigma/status/1606326906910973963>>. Les développeur-euses qui produisent du code de basse qualité à la chaîne ont du souci à se faire pour leur avenir professionnel mais cela ne concerne pas celles et ceux qui font du code créatif, difficile, novateur, ou simplement bien adapté à leur problème. ChatGPT est très fort pour des tests simplistes, comme ceux souvent faits à l'embauche <<https://courses.cfte.education/chatgpt-outperforms-4m-programmers/>> mais pour de vrais programmes ? C'est un point souvent oublié par les personnes qui ne connaissent pas la programmation : écrire un petit script qui fait une tâche simple <<https://twitter.com/MikaelBras/status/1606258125710848001>> n'est pas la même chose que de programmer Apache, Louvois, ou un logiciel de calcul scientifique comme Blast.

Quelques réflexions et éléments sur ce sujet :

- Un programme (certes simple mais quand même) entièrement développé par ChatGPT <<https://thetinycto.com/blog/writing-a-game-using-chatgpt>>.
- Un cas où ChatGPT a imaginé une fonction de la bibliothèque standard, fonction qui n'existe pas <<https://elixirforum.com/t/elixirs-enum-transpose-1-written-by-chatgpt/52447/>>.
- En dehors du cas de la programmation, de bonnes réflexions sur Upcycle Commons <<https://upcyclecommons.wordpress.com/2023/01/07/chatgpt-objet-de-tous-les-fantasmes/>>, celles de Marie-Astrid Clair <<https://www.ecoledeslettres.fr/chatgpt-la-nouvelle-intelli>>, ou bien celles de David Madore <<http://www.madore.org/~david/weblog/d.2023-01-10.2741.ai-and-chatgpt.html>> ou encore cette vidéo de Monsieur Phi <<https://indymotion.fr/w/qBCr9P7f9oE2DPtkCZCuCy>>.
- Un article (écrit par le fondateur d'une entreprise qui vend des logiciels d'IA, attention) qui dit le contraire du mien <<https://cacm.acm.org/magazines/2023/1/267976-the-end-of-programming-fulltext>> (les commentaires sont très intéressants).