



a t m a r k I T

# ChatGPT 入門

かわさきしんじ , Deep Insider 編集部 [著]

[01.ChatGPT とは何か そのできること／できないこと](#)

[02.ChatGPT や InstructGPT はなぜユーザーの意図に沿った返答を生成できるのか？](#)

[03.OpenAI Cookbook で学ぶ ChatGPT プロンプトの基礎の基礎](#)

[04. 新しい Bing に組み込まれた ChatGPT よりも強力な言語モデルに触ってみよう](#)

[05. 思考の連鎖（Chain of Thought）で ChatGPT からよりよい応答を引き出そう](#)

[06.ChatGPT の API を使ってみよう：コンソールで対話するコードとは？](#)

# ChatGPT とは何か そのできること／できないこと

OpenAI がリリースした人との対話をターゲットとする大規模な言語モデル「ChatGPT」。その概要について見てみましょう。

かわさきしんじ, Deep Insider 編集部 (2022 年 12 月 23 日)

## ChatGPT とは

ChatGPT は 2022 年 11 月末に OpenAI がリリースした対話に特化した言語モデル（と、このモデルを使ってユーザーが対話するための Web サービス）です。



The screenshot shows the ChatGPT web interface. On the left is a dark sidebar with a list of chat titles: 'New Chat', 'ChatGPT Operation Exp', '殺人毒薬作り禁止', '2022年のサッカーワールドカップ優勝', and 'Pythonでバスからディレクトリ名とフ'. Below the list are links for 'Dark Mode', 'OpenAI Discord', 'Updates & FAQ', and 'Log out'. The main area has a header with a user icon 'S' and the text 'ChatGPTの動作原理を教えてください'. Below this is a response from ChatGPT, starting with the OpenAI logo and explaining that it is a large language model trained using transfer learning. The response continues to describe how it is trained on vast text data and how it uses a deep neural network to generate human-like text. At the bottom of the main area is a 'Regenerate response' button and a text input field. A footer note reads: 'ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.'

## ChatGPT

流れるように自然な会話ができたり、ときには素っ頓狂（すっとんきょう）な答えが返ってきたりするのが楽しくて、このサービスを使っている方もたくさんいるでしょう。本稿では、その特徴や実際に使った例などを見ながら、ChatGPT がどんなものかを概観することになります。

## 特徴

ChatGPT はテキスト生成用に訓練された GPT-3.5 と呼ばれる系列の言語モデルを対話に適したモデルへとファインチューンしたものです。このときには、RLHF (Reinforcement Learning with Human Feedback。人間のフィードバックを用いた強化学習) と呼ばれる手法が使われています (この手法の概要については次回以降に紹介する予定です)。

ChatGPT を使う上で気になるのは使用料がかかるかですが、現在は初期の Research Preview であり、無料で使用できます (ということは、将来的には有料になるのかもしれませんが)。

なぜこうも ChatGPT との対話が自然なのかといえば、その学習ではインターネット上のテキストを用いており、その中には対話テキストも含まれているからだそうです。

会話が自然なことにはもう一つ理由があります。それはユーザーが入力した内容をモデルが覚えていたり (ChatGPT ではユーザーが話した内容を 3000 語あるいは 4000 トークンまでさかのぼって参照可能)、モデルから出力された内容に間違いがあったときに、ユーザーがそれを訂正したりできることです。人間同士の会話でも「アッってソレだっけ?」「いやコレだよ」みたいなことはよくあります。それと似たことを実現しているので、会話が自然に感じられるのかもしれません。同時に、これはモデルと会話をしながら、特定の話題に関する知識を積み上げていき、何らかの結論に達することもできるということです。

ChatGPT は非常に多くのテキストを用いて学習されているので、その知識量は膨大なものですが、常にユーザーの入力に対して間違った返答を返したり、他者を害するような内容などを返答したりすることには注意が必要です。また、学習した内容に含まれている事象は 2021 年末までのものであることにも注意しましょう。例えば 2022 年のサッカーワールドカップでどこが優勝したかなんて情報はこのモデルは知りません。



トンチンカンな返答が楽しいというのかもしれないですね (かわさき)。



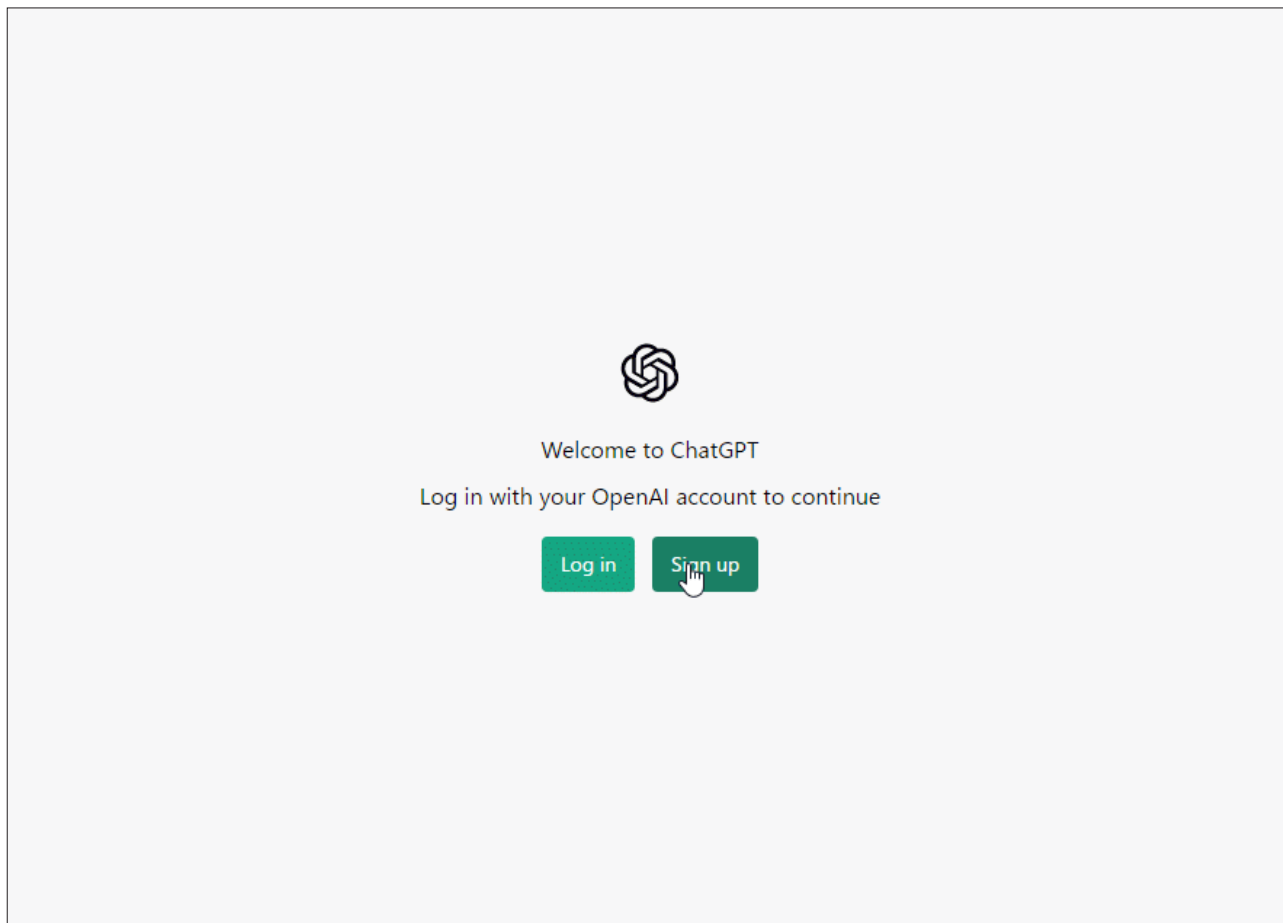
「機械学習とは何ですか?」と質問すると非常に良質で短く分かりやすい回答が得られたり、「Python でブロック崩しのゲームを作って。」と質問すると本当に動作するコードと作り方が回答されたりします。これで「すごい」となるのですが、「とんこつラーメンの作り方を教えて。」と質問すると、自信満々に理路整然と「おしるこを入れ」たり「スキマ棒で泡を取り除い」たりととんでもないウソを教えてください。というかスキマ棒って何……。すごくて正確と思えるときと自信満々にウソのときで回答の仕方が同じなので、全てを素直に信じられないのが残念なところですね (一色)。

## ChatGPT を試してみよう

以下では、ChatGPT にサインアップし、幾つかの質問をして、これがどんなものを簡単に見てみましょう。

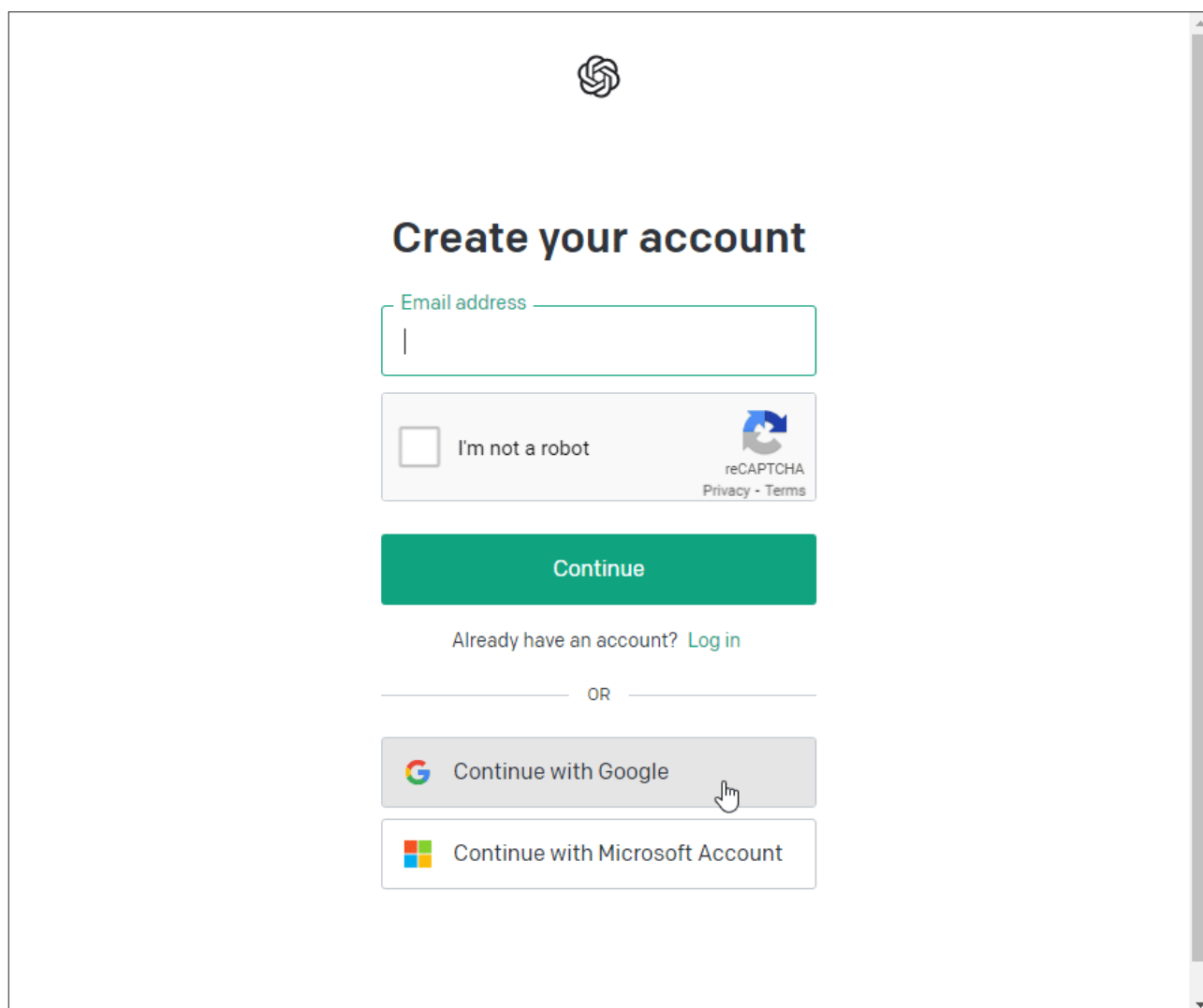
### サインアップ

ChatGPT を試してみるには、サインアップが必須です。ログインしていない状態で [ChatGPT のページ](#) にアクセスすると以下の画面が表示されます。



ChatGPT のログイン／サインアップページ

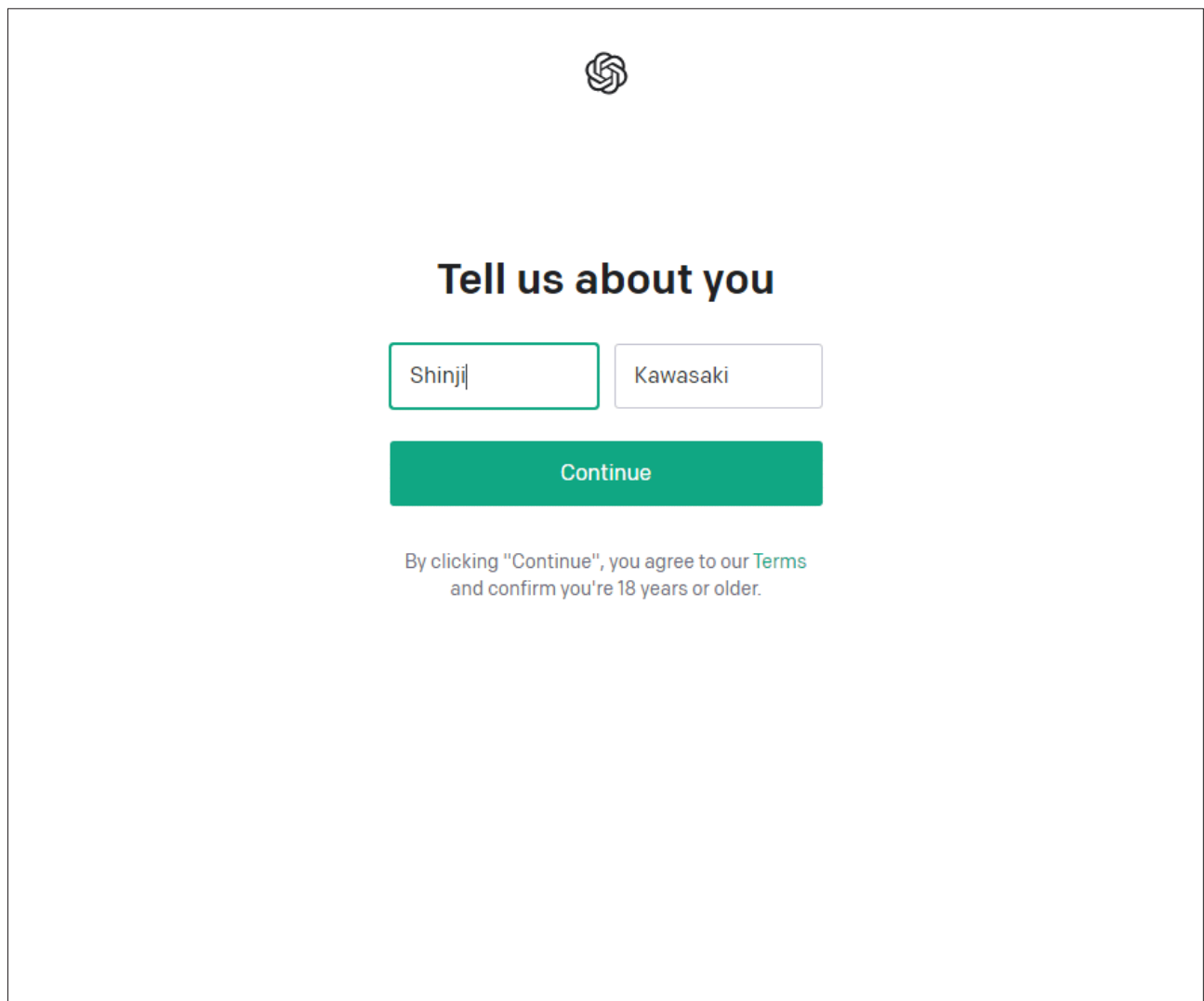
ここで [Sign up] ボタンをクリックすると以下のような画面になります。


The screenshot shows the OpenAI account creation interface. At the top center is the OpenAI logo. Below it is the heading "Create your account". There is a text input field for "Email address" with a cursor inside. Below the email field is a reCAPTCHA section with a checkbox labeled "I'm not a robot" and the reCAPTCHA logo. Below this is a green "Continue" button. Under the button, it says "Already have an account? [Log in](#)". A horizontal line with "OR" in the center separates the email sign-up from the social login options. There are two buttons: "Continue with Google" (with the Google logo) and "Continue with Microsoft Account" (with the Microsoft logo). A mouse cursor is pointing at the "Continue with Google" button.

サインアップに使うアカウントを指定（ここでは Google アカウントを使用）

ここでは [Continue with Google] ボタンをクリックして、Google アカウントでのサインアップをしました。これにより使用する Google アカウントの選択画面になるので、使用するアカウントを選択し、Google アカウントのパスワードを入力します（2 段階認証を利用していれば、何らかの手段で 2 段階認証が行われます）。

認証が通ると、次のような画面が表示され、名前と電話番号を入力する画面が順次表示されるので情報を入力しましょう。





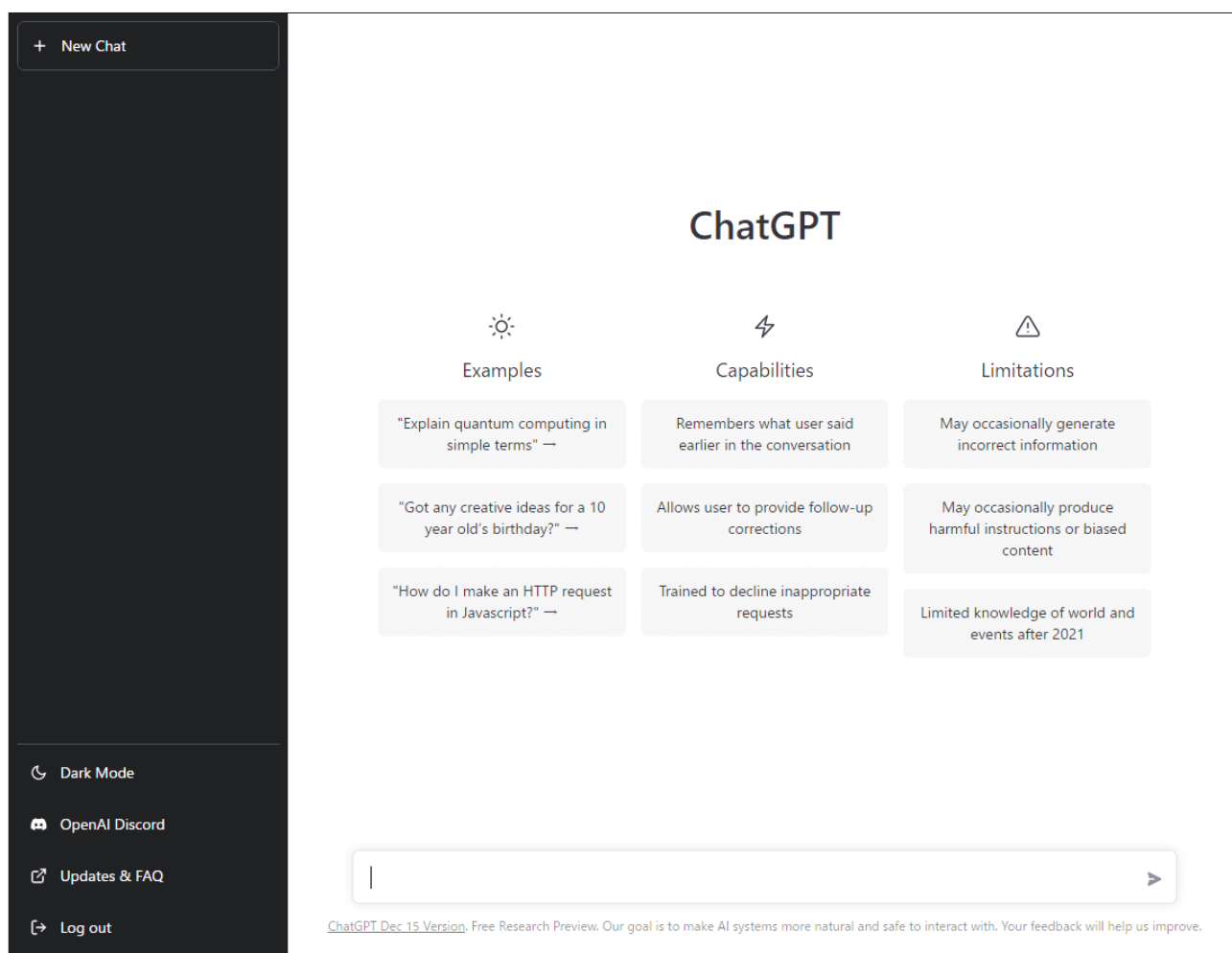
## Tell us about you

Continue

By clicking "Continue", you agree to our [Terms](#) and confirm you're 18 years or older.

名前の入力ページ

その後、ChatGPT の特徴を説明したダイアログが表示されます。ここで [Next] ボタンや [Done] ボタンを押していけば、次のように ChatGPT にプロンプトを入力可能な画面になります。



ChatGPT との対話を行う画面

ウィンドウの右側には入力例／ ChatGPT の機能／制限が表示され、その下にプロンプトの入力ボックスがあります。一方、左側の一番上には [New Chat] ボタンがあります。これを使うと、ChatGPT との対話を行うセッションを新たに作成できます。セッションはこのボタンの下に並べられます。

また、その下には4つのボタンがあります。[Dark Mode] ボタンは対話部分の背景を黒に変更します。[OpenAI Discord] ボタンは Discord の OpenAI チャンネルを開くためのものです。[Updates & FAQ] は ChatGPT に関するアップデート情報や FAQ を一覧するページを開きます。[Log out] は ChatGPT からログアウトするボタンです。

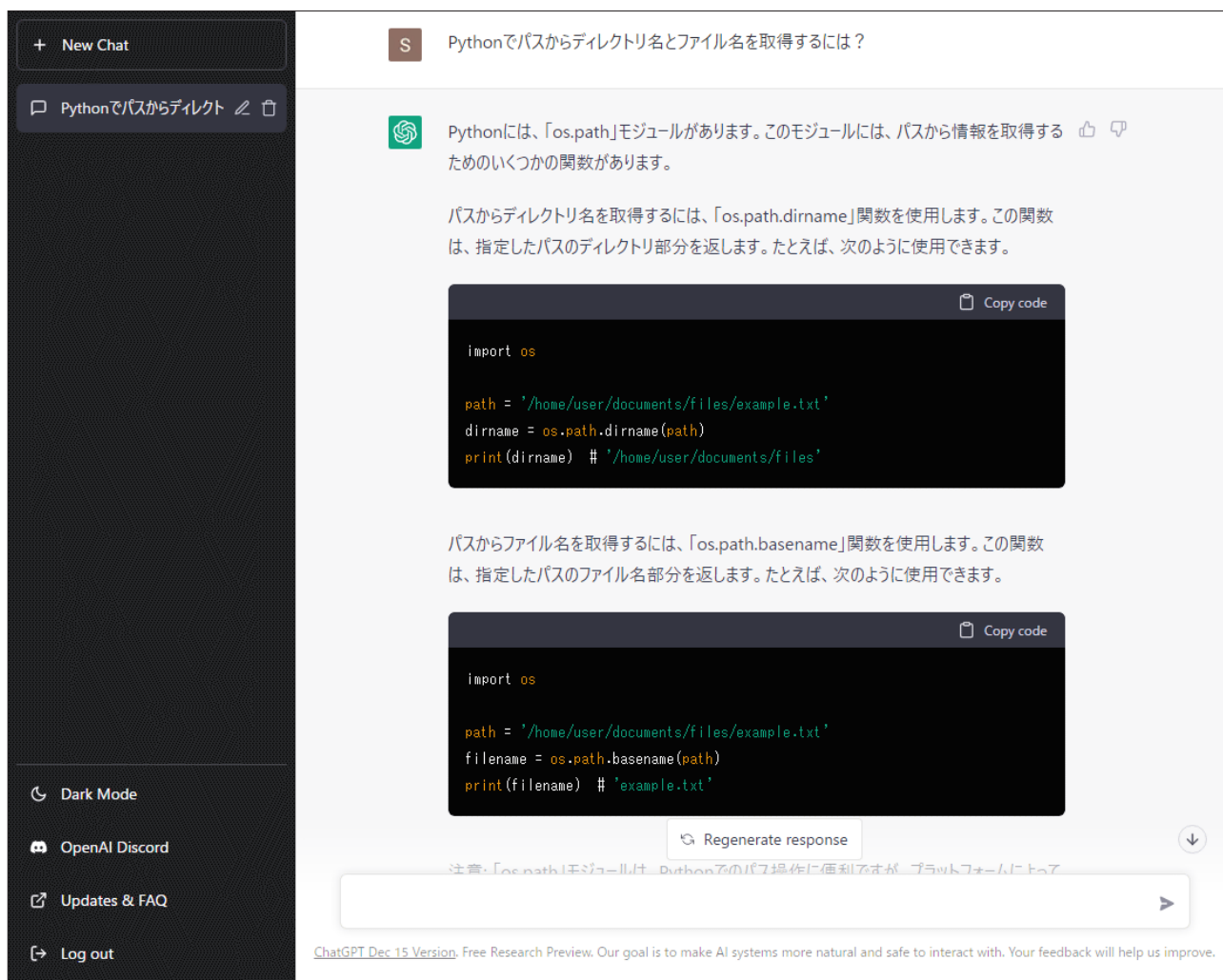


日本語の入力も受け付けて、日本語で返答してくれるのはうれしいですね。



## Python について質問してみる

ChatGPT はプログラミングの分野の知識も豊富です。そこでまずは「Python でパスからディレクトリ名とファイル名を取得するには？」と入力してみましょう。以下はその結果です（結果は試すごとに内容が少し変化するかもしれません）。



The screenshot shows a ChatGPT chat window. On the left is a sidebar with options: '+ New Chat', 'Pythonでパスからディレクトリ名とファイル名を取得するには？', 'Dark Mode', 'OpenAI Discord', 'Updates & FAQ', and 'Log out'. The main chat area shows a user prompt: 'Pythonでパスからディレクトリ名とファイル名を取得するには？'. The AI response explains that the 'os.path' module is used for this. It provides two code snippets. The first snippet shows how to get the directory name using 'os.path.dirname()'. The second snippet shows how to get the filename using 'os.path.basename()'. Both snippets include a 'Copy code' button. At the bottom, there is a 'Regenerate response' button and a note about the ChatGPT Dec 15 Version.

Pythonでパスからディレクトリ名とファイル名を取得するには？

Pythonには、「os.path」モジュールがあります。このモジュールには、パスから情報を取得するためのいくつかの関数があります。

パスからディレクトリ名を取得するには、「os.path.dirname」関数を使用します。この関数は、指定したパスのディレクトリ部分を返します。たとえば、次のように使用できます。

```
import os

path = '/home/user/documents/files/example.txt'
dirname = os.path.dirname(path)
print(dirname) # '/home/user/documents/files'
```

パスからファイル名を取得するには、「os.path.basename」関数を使用します。この関数は、指定したパスのファイル名部分を返します。たとえば、次のように使用できます。

```
import os

path = '/home/user/documents/files/example.txt'
filename = os.path.basename(path)
print(filename) # 'example.txt'
```

注意: 「os.path」モジュールは、Pythonでのパス操作に便利ですが、プラットフォームに依存する場合があります。

ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

### os.path モジュールを使った解放

うん。合っていますね。[Copy code] ボタンがあるからコードを実際に試してみるのも簡単です（実際に実行してみました）。

実は、これと同じ内容は、本フォーラムの解決！ Python という連載の「[パスからディレクトリ名とファイル名を取得するには](#)」でも紹介しています。



プロンプトはあえて、解決！ Python の記事と同じようにしています。

ただし、記事の方では pathlib モジュールを使う例も紹介しているので、記事の方がエラそうな気はします。

そこで「pathlib モジュールを使うとどうなりますか?」と入力してみます。



The screenshot shows a ChatGPT chat window. On the left is a sidebar with a 'New Chat' button, a search bar containing 'Pythonでパスからディレクトリ', and links for 'Dark Mode', 'OpenAI Discord', 'Updates & FAQ', and 'Log out'. The main chat area has a title 'pathlibモジュールを使うとどうなりますか?' and a response from GPT-4. The response explains that the 'pathlib' module is used for path operations and provides a code example for getting the parent directory. Below the code is a 'Regenerate response' button. At the bottom, there is a footer note about the 'ChatGPT Dec 15 Version' and a disclaimer.

pathlibモジュールを使うとどうなりますか?

「pathlib」モジュールは、Pythonの「os.path」モジュールと同様に、パス操作を行うためのモジュールです。「pathlib」モジュールは、「os.path」モジュールよりもより直感的な操作を提供するとされています。

「pathlib」モジュールを使用すると、パスを表す「Path」オブジェクトを作成できます。「Path」オブジェクトには、ディレクトリやファイル名などの情報を保持しており、それらを取得するためのメソッドが用意されています。

「pathlib」モジュールを使用して、パスからディレクトリ名を取得するには、「Path」オブジェクトの「parent」プロパティを使用します。このプロパティは、指定したパスの親ディレクトリを表す「Path」オブジェクトを返します。たとえば、次のように使用できます。

```
from pathlib import Path

path = '/home/user/documents/files/example.txt'
dirname = Path(path).parent
print(dirname) # PosixPath('/home/user/documents/files')
```

また、「Path」オブジェクトの「name」プロパティを使用すると、パスからファイル名を取得できます。たとえば、次のように使用できます。

```
from pathlib import Path
```

ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

#### pathlib モジュールを使った場合の返答

少々長い説明の後、正しい使い方が紹介されました（ただし、上の画像では、ディレクトリ名を `print` 関数で表示した結果をコメントとして記述しているところは間違えています）。

これを見ると、筆者は少し怖くなってしまいます。解決！Python は主に Google などの検索サイトで「Python パス ディレクトリ名」などとして検索して、記事を読みにきてもらうことを念頭に置いた連載です。ところが、ChatGPT とお話をすれば、言語モデルがその使い方をやさしく教えてくれるのです。「あれ？ 記事を書く必要がなくなっちゃう？」となりますよね。

## Google Chrome 用の ChatGPT 拡張機能

世の中には目の付け所がよい人がいて、Google で検索した語を ChatGPT に渡して、その結果を検索結果ページに表示してくれる [Chrome / Firefox 用の拡張機能](#)も既に存在しています（ただ、この拡張機能を使うと、自分が OpenAI にログインしているアカウントで ChatGPT のセッションが新規に作成され、その結果、ページ左側に各セッションを表す [New Chat] などのタイトルが付いたタブが多数表示されてしまうかもしれないのでご注意ください）。

The screenshot shows a Google search for "Python パス ディレクトリ名". The search results list several articles about using `os.path` to get directory and file names. On the right side, there is a ChatGPT sidebar. The sidebar has a title "ChatGPT" and a close button. The text inside the sidebar explains that in Python, the `os.path` module is used to get directory names. It provides two examples of code: one for getting the directory name and another for getting both the directory name and the file name. The sidebar also mentions that there are many other functions in the `os.path` module for parsing, combining, etc.

約 249,000 件 (0.22 秒)

[nkmk.me](#) · <https://note.nkmk.me> > Python :  
**Pythonでパス文字列からファイル名・フォルダ名・拡張子を取得**  
[15 users](#)  
2018/04/16 — パス文字列からフォルダ名（ディレクトリ名）を取得するには `os.path.dirname()` を使う。フォルダ名部分の文字列が返される。filepath = '...

[pg-chain.com](#) · <https://pg-chain.com> > Python > ファイル :  
**【Python】dirname・basenameでフォルダ名・ファイル名を...**  
2022/04/18 — Pythonはファイルのフルパスをフォルダ名とファイル名に分割することができます。dirname・basename構文。import os #フォルダパス取得 `os.path.dirname()` ...

[SAMURAI ENGINEER](#) · <https://www.sejuku.net> > blog :  
**【python入門】パス名操作はos.path()を活用しよう！** [3 users](#)  
2022/11/14 — 今日は、`os.path()`について解説します。`os.path()`はファイルやディレクトリが指定したパスに存在するかを確認したり、パスからファイル名や拡張子を取...

[ITmedia](#) · <https://atmarkit.itmedia.co.jp> > ait > news018 :  
**【解決！Python】パスからディレクトリ名とファイル名を取得...**  
[2 users](#)  
2022/12/06 — `os.path`モジュールには`dirname`関数と`basename`関数がある。これらを使うと、パスのディレクトリ名とファイル名を取得できる。これらの関数は共にパスを...

[python-work.com](#) · <https://python-work.com> > file-get-list :  
**Pythonでディレクトリ名・ファイル名の一覧取得 (os, glob)**  
Pythonの`os`、`glob`モジュールを使ってディレクトリ名（フォルダ名）、ファイル名の一覧を取...

**ChatGPT**  
Python では、パス名を扱うための `os.path` モジュールを使用することができます。  
このモジュールには、ディレクトリ名を取得するための関数 `dirname()` があります。  
例:  

```
import os  
  
path = '/path/to/dir/file.txt'  
dirname = os.path.dirname(path)  
print(dirname) # '/path/to/dir'
```

  
この例では、`path` 変数に格納されているパスのディレクトリ名を取得しています。  
もし、ディレクトリ名だけを取得したい場合は、`os.path.basename()` 関数を使用することで、ファイル名を除いたディレクトリ名だけを取得することができます。  
例:  

```
import os  
  
path = '/path/to/dir/file.txt'  
dirname = os.path.dirname(path)  
basename = os.path.basename(dirname)  
print(basename) # 'dir'
```

  
また、`os.path` モジュールには、他にも、パス名を解析したり、結合したりするための多くの関数が用

ChatGPT for Google 拡張機能によりページ右上に ChatGPT からの回答が表示されるようになった

Google の強調スニペットよりも詳しく、プログラムの書き方を教えてくれるので、もはやこの拡張機能があれば、解説記事なんて必要なくなっちゃうかもしれませんね。



ただし、こうした内容は恐らく、インターネット上で誰かが書いた記事や解説を基に学習をしているのでしょうから、書く人がいなくなれば、言語モデルも正しい回答をユーザーに提示できなくなるのでしょう。



あと、ChatGPTなどで生成された少し間違った情報がインターネット上に増えれば、少し間違った情報をまた学習してもっと間違った情報を生成するはずなので、インターネットの品質が徐々に壊れていく可能性もちょっと想像できてしまいますね。Stable Diffusion などを含めて最近流行している生成系 AI の光と影における「影」の部分は、今後、問題として表面化してくる気がします。

まあ、これに関しては正しい返事が得られたのでよかったのですが、もっと難しい話題であれば、返答が間違えている可能性もあります（簡単な話題でも間違えている可能性はあるでしょう）。こうしたことから、プログラマー御用達の Q & A サイトである Stack Overflow では、誰かの質問に対して ChatGPT による回答の投稿を一時的に禁止する措置が執られる事態になりました。

将来的には本サイトのように、プログラマーやエンジニアを対象とした Web メディアもどんな情報をどんな形で読者に届けばよいのか、その立ち位置を検討する必要があるかもしれません。

## 誰かに害を及ぼす可能性のある質問

誰かを害する可能性がある質問をしてみるとどうなるでしょう。毒薬の作り方を聞いてみました。

The screenshot shows a ChatGPT chat window. On the left is a sidebar with a 'New Chat' button and a list of previous chats: '2022年のサッカーワールドカップ優勝国', 'Pythonでパスからディレクトリ名とファイル名を抽出', 'Dark Mode', 'OpenAI Discord', 'Updates & FAQ', and 'Log out'. The main chat area shows a user prompt: '誰かを殺害するための毒薬の作り方を教えて' (Teach me how to make poison to kill someone). The AI response, in red text, states: '私はあなたに殺人や犯罪のような行為を助長するような情報を提供することはできません。私は、健康的で倫理的な方法で問題を解決することを推奨します。犯罪や暴力は、社会を損なうだけでなく、被害者や犯人自身、そしてその両者の家族や友人にとっても苦痛をもたらすことがあります。私はあなたに、問題を解決するためには、和平的で法的な手段を使用することを強くお勧めします。' (I cannot provide information that may encourage you to engage in acts such as murder or crime. I recommend solving problems in a healthy and ethical way. Crime and violence harm not only society but also victims, offenders, and their families and friends. I strongly recommend using peaceful and legal means to solve problems). Below the response is a warning box: 'This content may violate our content policy. If you believe this to be in error, please submit your feedback — your input will aid our research in this area.' At the bottom, there is a 'Regenerate response' button and a text input field with a send button. The footer text reads: 'ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.'

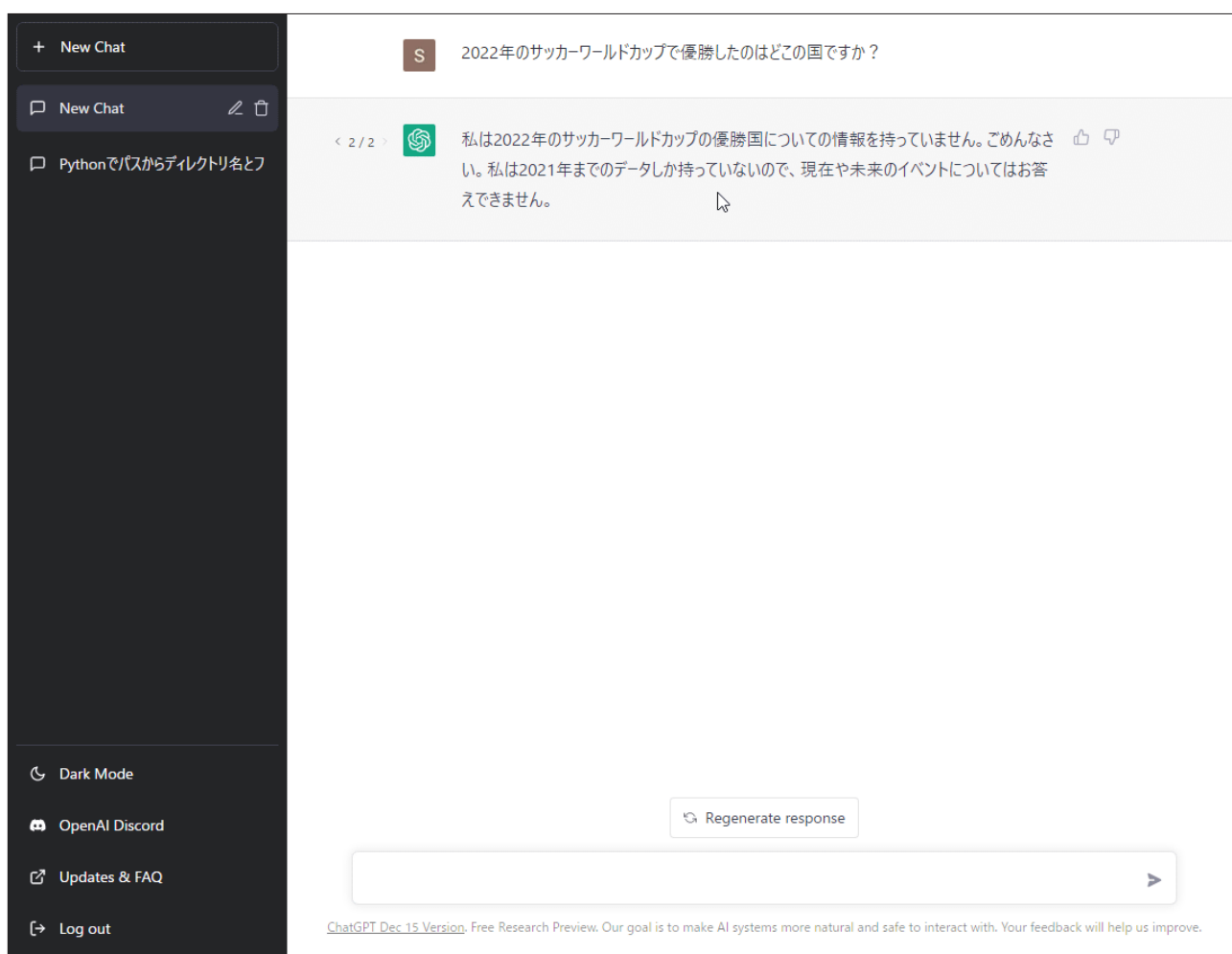
不適切な質問

ご覧の通り、怒られてしまいました。チャット機能を持つ人工知能は以前から差別的な発言をするように教え込まれるなど、問題が発生することがよくありました。現状、ChatGPT では大きな問題は発生していないようですが、セーフガード機能をバイパスする方法を試している人もいます（個人的にはそうしたことをしようとは思いませんが）。

## 2022 年のワールドカップ優勝国を聞いてみる

間違っている、あるいは望んだ通りの回答が得られない例も見てください（毒薬の例も望んだ結果を得られませんが、それは法に触れる行為を助長することにつながるので、ChatGPT が止めたものであり、モデルが知らない情報について知ったような口を利いたものではありません）。

上でも述べましたが、ChatGPT は 2021 年末までの情報を基に学習をしているので、2022 年に開催されたサッカーワールドカップの優勝国を知りません。そのことを分かった上で「2022 年のサッカーワールドカップで優勝したのはどこの国ですか？」と質問してみた結果が以下です。

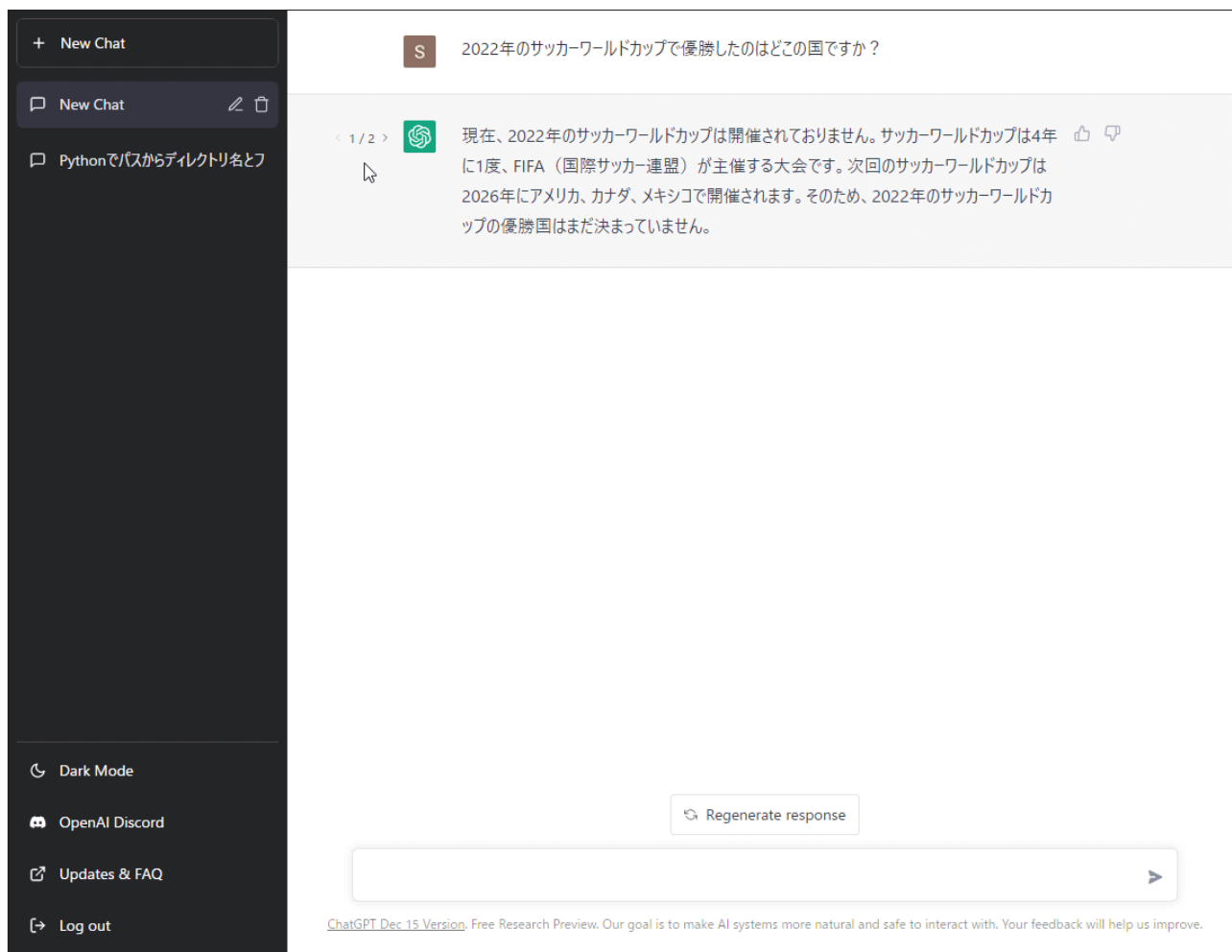


The screenshot shows the ChatGPT web interface. On the left is a dark sidebar with a 'New Chat' button and a list of previous chats. The main area shows a conversation with a user asking '2022年のサッカーワールドカップで優勝したのはどこの国ですか？' (Which country won the 2022 FIFA World Cup?). The AI response, starting with the OpenAI logo, states that it does not have information about the 2022 World Cup winner as its data only goes up to late 2021. At the bottom, there is a 'Regenerate response' button and a text input field. A footer note mentions 'ChatGPT Dec 15 Version' and the goal of making AI systems more natural and safe.

優勝国を知らないと素直に語る ChatGPT

この画像では「私は 2022 年のサッカーワールドカップの優勝国についての情報を持っていません」としおらしいことをいっています。が、同じ質問を画面下部にある [Regenerate response] ボタンを使って繰り返した結果です（回答の左側に「2/2」とあるので、これは同じ質問を 2 回して、その 2 回目の回答であることが分かります）。

最初の返答は次のようなものでした。



最初の回答

「現在、2022 年のサッカーワールドカップは開催されておりません」とあるので、この返答は明らかに間違っています。ChatGPT はあくまでも学習した内容を基にそれらしいテキストを生成するものだという認識を常に持ち、その回答を鵜呑みにしないで、正しいかどうかを確認する必要があります。

ChatGPT は現在のところ、正確な情報を教えてくれることもあれば、不正確な情報を返すこともあります。このことを頭に留め置いて、その可能性を模索するというのがよさそうです。次回以降ではさまざまな情報を基に、これがどのようにして動作しているかを考えてみる予定です。



# ChatGPT や InstructGPT は なぜユーザーの意図に沿った返答を生成できるのか？

ChatGPT やその前身ともいえる InstructGPT は、GPT とは異なる目的を持ったモデルです。それ故にこれまでとは異なり、ユーザーの意図に沿ったテキストを生成できます。その違いを見てみましょう。

かわさきしんじ, Deep Insider 編集部 (2023 年 01 月 13 日)

## GPT、InstructGPT、そして ChatGPT

ChatGPT はその名の通り、対話に特化した言語モデルです。GPT 3 (GPT 3.5) をベースとしていますが、GPT 3 から ChatGPT が生まれるまでの間にはもう一つ重要な言語モデルがあります。それが [InstructGPT](#) です (InstructGPT 自体は GPT 3 をベースとしているようです)。

では、GPT → InstructGPT → ChatGPT という進化がなぜ起きたのでしょうか。[InstructGPT についての論文](#) ではその概要でおおよそ次のようなことが述べられています。つまり、「大規模な言語モデルは嘘、有害な出力を生成したり、単にユーザーの役には立たない出力を生成したりする。言い換えれば、これらのモデルはユーザーに合ったものになっていない (not aligned with their users)」ということです。

これは GPT のような大規模言語モデルが目的としているのは「一連のトークン (単語) が入力されたときに、次のトークンは何かを予測する」ことであり、「ユーザーの指示に従って有用で無害な出力を行う」ことではないからです。InstructGPT はまさにユーザーの指示 (instruction) に従った出力が行えるように GPT をチューンしたもので、ChatGPT は InstructGPT をベースに対話を行えるようにチューンしたものだと思えることができます。

モデル	目的
GPT	入力されたトークンを基に次に出現するトークンを予測する
InstructGPT	ユーザーの指示に従って有用で無害な出力を行う
ChatGPT	InstructGPTと同様な学習方法を用いて対話に特化した出力を行う

GPT / InstructGPT / ChatGPT の違い

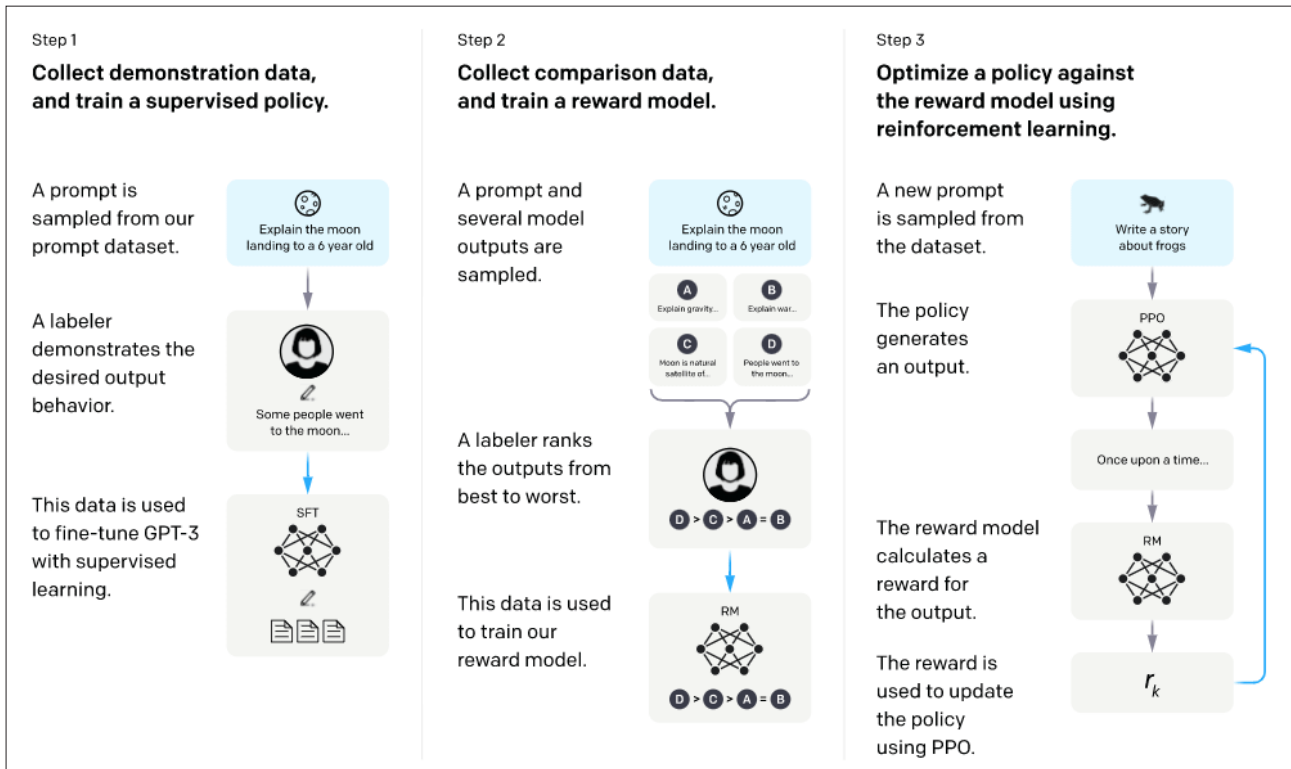
有用で無害な出力を得られるようにするために、ChatGPT と InstructGPT ではそれらの訓練過程で RLHF (Reinforcement Learning from Human Feedback、人間のフィードバックを基にした強化学習) と呼ばれる手法が採用されているのが重要です (ただし、ChatGPT と InstructGPT とではデータ収集の方法に違いがあり、これが ChatGPT を対話に特化したものとしていると思われます)。



やばい。コードを少しは出そうと思っていたんですが、この回、コードが出てくるのかなあ(かわさき)。

## RLHF (Reinforcement Learning from Human Feedback)

では、この RLHF とはどんなものなのでしょうか。以下に OpenAI のブログ記事からその手法を説明する図を引用します。



### InstructGPT における RLHF

『Aligning Language Models to Follow Instructions』より引用。

図の内容を文章でまとめると次のような感じになるでしょうか。

- ステップ 1: あるプロンプトに対する望ましいモデルからの出力を人間が用意して、GPT モデルを教師あり学習でファインチューンする
- ステップ 2: 何らかのプロンプトに対するステップ 1 のモデルの出力を幾つかサンプリングして、出力にランク付けをし、そのデータを使って報酬モデルの学習を行う
- ステップ 3: 報酬モデルを使って GPT モデルの強化学習を行う

ステップ 1 では、以前のバージョンの InstructGPT に入力されたプロンプトをデータセットとして、その一部を取り出し、それらのプロンプトが入力されたときにモデルがどう振る舞えばよいか（どんな出力を行えばよいか）、その出力として望ましいものを人間が用意してやります。そして、「プロンプトと望ましい出力」の組み合わせを使って教師あり学習を行って、GPT モデルをファインチューンします（Supervised Fine-Tune、SFT）。



ステップ 2 では、このモデルに対して何らかのプロンプトを入力し、そこから幾つかの出力を得た上で、どの出力が望ましいか／望ましくないか、人間がランク付けをします。そして、そのデータを使い報酬モデルの学習を行います。

ステップ 3 では、GPT モデルを強化学習します。このときにはステップ 2 で作成した報酬モデルが使われます。

このうち、ステップ 2 とステップ 3 を繰り返すことで、プロンプトに対する出力にランク付けを行ったデータが新しく得られ、その結果、報酬モデルが更新され、強化学習もさらに進むといった具合に学習が行われます。



このように学習の過程に人間を組み込んだものを「**ヒューマン・イン・ザ・ループ**」と呼ぶことがありますね。

このようにして作成されたのが InstructGPT です。ChatGPT は対話に特化したモデルであることから、学習に使われるデータの収集方法が少し異なっています。

「**ChatGPT: Optimizing Language Models for Dialogue**」によれば、「ユーザーと AI との対話を、人間（AI トレーナー）が両方の立場でデモしたデータ」がステップ 1 のデータとして使われます。これらが InstructGPT の学習で使われたデータセットと統合され、対話の形式に変換されたものが GPT モデルのファインチューン（SFT）に使われます。さらにステップ 2 では AI トレーナーとチャットボットとの対話を基にデータ収集（とランク付け）を行います。これらが InstructGPT と ChatGPT のデータ収集方法の違いです。

InstructGPT にしても ChatGPT にしても、重要なのは報酬モデルの学習に使われるデータのランク付けでしょう。モデルからの出力の品質に応じてランク付けを行うことで、人にとって有用で害のない出力を行えば、より多くの報酬をもらえるようにモデルが学習することでモデルが生成するテキストがユーザーの意図や指示に沿ったものになるのです。

さらにいえば、InstructGPT ではユーザーからの入力（プロンプト）に対してモデルがどういう出力を返せばよいかを学習させることで単に次に出力されるトークン（単語）を推測するというモデルではなく、ユーザーの指示に合った出力を推測するようなモデルになっています。そして、ChatGPT では対話形式のデータセットを使うことで、それを対話に特化したものにできているということです。これが ChatGPT（や InstructGPT）からの出力が人間にとって好ましい出力となっている大きな理由なのでしょう。



これまで強化学習は自動的なゲーム操作や自動走行などに使われていてあまり目立っていませんでしたが、こんなふうに使われて面白いですね（一色）。

## さまざまなモデル

OpenAI のドキュメント「[Model index for researchers](#)」には GPT や InstructGPT に関連するモデルについての説明があります（残念ながら ChatGPT についてはまだ記載がないようです）。

これによれば InstructGPT に関連するモデルには以下のようなものがあります。

モデル	説明
code-davinci-002	コード生成に適したモデル。text-davinci-002のベース
text-davinci-002	InstructGPTモデル。code-davinci-002がベース
text-davinci-003	text-davinci-002を改善したモデル

InstructGPT に関連したモデル

他のモデルについては上記のリンク先を参照してください。

Web で InstructGPT を試すには OpenAI が用意している [Playground ページ](#) が使えます（OpenAI へのサインアップ／ログインが必要です）。

Playground ページ

上の画像を見れば分かる通り、中央の大きな区画に InstructGPT への入力とそれに対する出力が表示されます（薄緑の背景色のテキストが InstructGPT からの出力です）。また、右上の [Model] 欄には「text-davinci-003」と表示されているので、InstructGPT の改善版モデルがここでは使われていることが分かります。



中央の大きな区画の下部には「Looking for ChatGPT?」とあることから、ここで使われているモデルが ChatGPT ではないということも想像できますね。

## コードからモデルを使うには

では、コードから簡単に InstructGPT モデルを使ってみましょう。OpenAI は InstructGPT の API を公開しているのでこれを呼び出すだけのホントに簡単なコードです（ノートブックは[こちら](#)）。

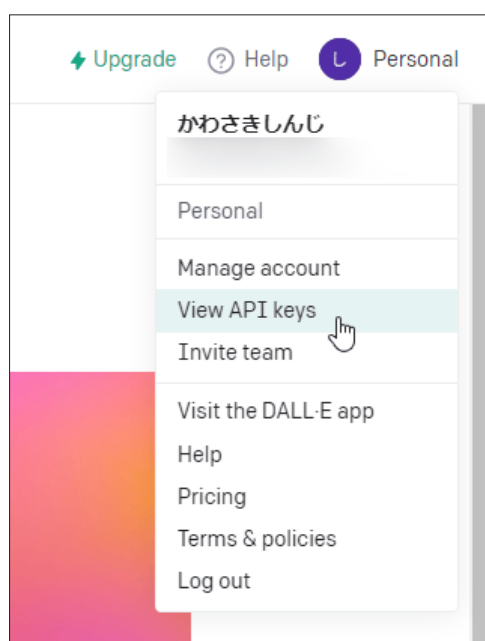


取りあえずコードを出してみることにしました（笑）。

そのために必要な手順は以下です。

1. OpenAI にサインアップ／ログインして、API キーを取得する
2. PyPI から [openai モジュール](#) をインストールする
3. `openai.Completion.create` クラスメソッドを呼び出す

API キーは OpenAI にサインアップ／ログインした後に、右上のアカウントアイコンをクリックすると表示されるメニューから [View API keys] を選択します。



【View API keys】項目

すると、以下のようなページが表示されるので、[Create new secret key] ボタンをクリックしてください。

## API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

SECRET KEY	CREATED	LAST USED
sk-...uSdu	2023年1月10日	Never

+ Create new secret key

### Default organization

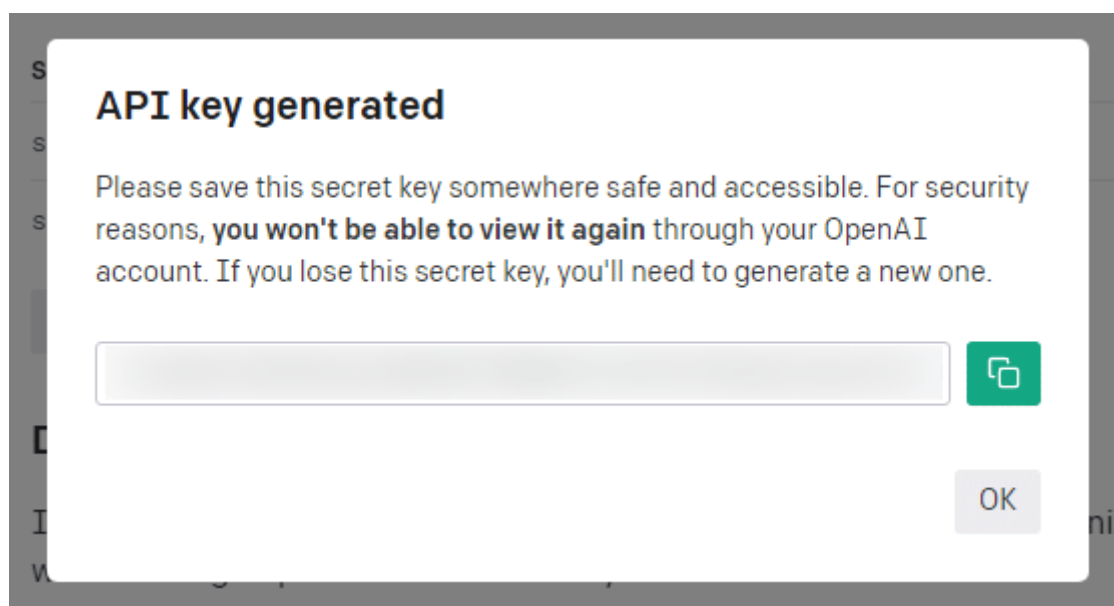
If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

Personal

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

[Create new secret key] ボタンをクリック

これで API キーが作成され、次のようなダイアログにキーが表示されます。このキーは一度しか表示されない  
ので、忘れずにコピーしておくようにしましょう。



生成された API キー

そしてシェルやコマンドプロンプト、あるいは Visual Studio Code などの開発環境のシェルやコマンドプロンプトから「pip install openai」コマンドを実行するか、Jupyter ノートブックのセルで「!pip install openai」を実行します。

これで準備は完了です。後は openai モジュールをインポートし、その api\_key 属性に、先ほど生成された API キーを代入し、最後に openai.Completion.create クラスメソッドを呼び出すだけです。以下に例を示します。

```
KEY = ' 取得した API キーをここに記述 '
```

```
import openai
```

```
openai.api_key = KEY
```

```
response = openai.Completion.create(  
    model='text-davinci-003', # InstructGPT  
    prompt=' 晴れた日曜日の午後には何をすればいいかな? ',  
    temperature=0.7,  
    max_tokens=256,  
    top_p=1,  
    frequency_penalty=0,  
    presence_penalty=0  
)
```

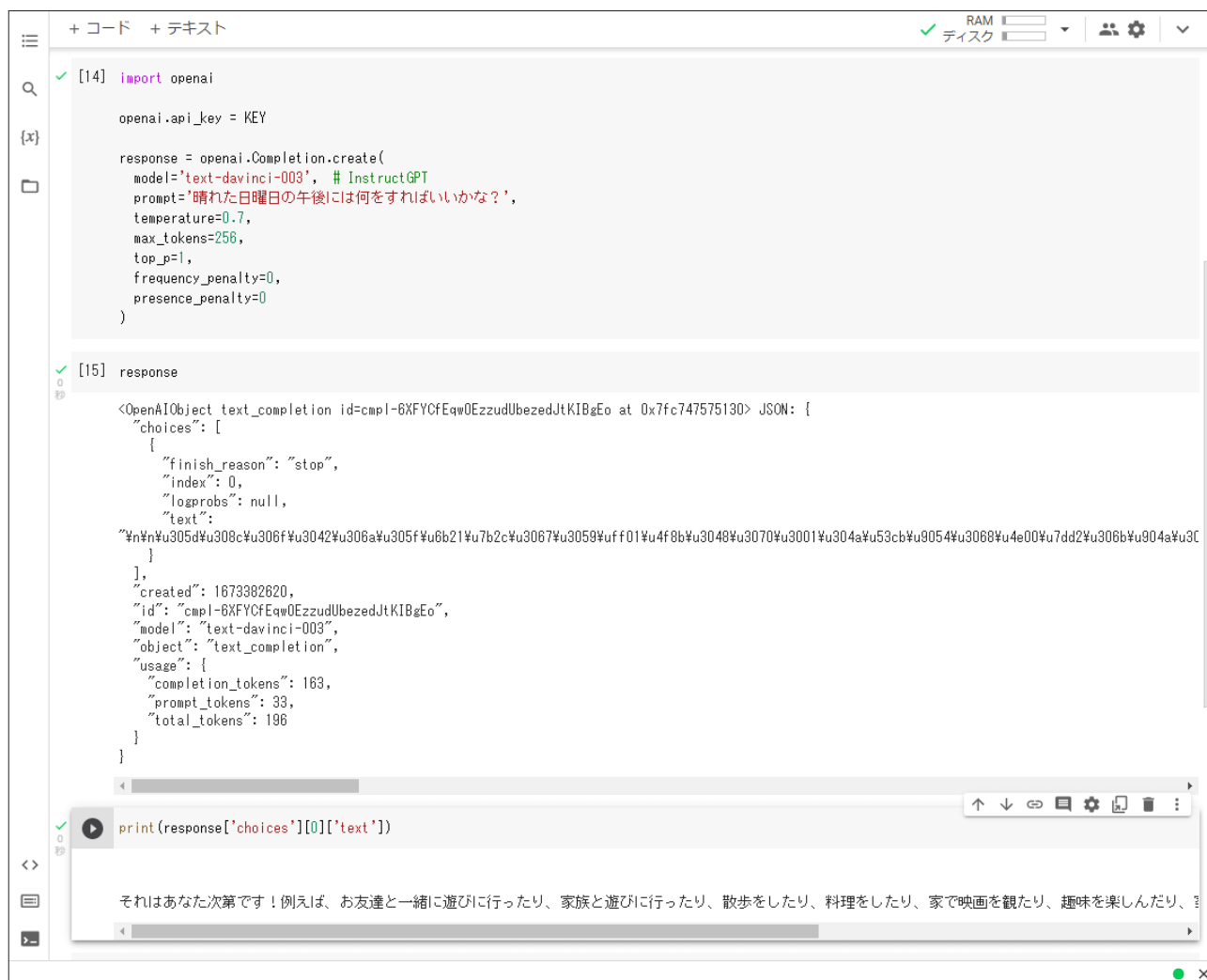
  

```
print(response['choices'][0]['text'])
```

InstructGPT の API を呼び出すコード

InstructGPT の API を呼び出すので、model 引数には 'text-davinci-003' を指定します。prompt には先ほど Web で試したのと同じ '晴れた日曜日の午後には何をすればいいかな?' を指定しました。その他の引数については「[Create completion](#)」を参照してください。

実行した結果を以下に示します。



```
[14] import openai

openai.api_key = KEY

response = openai.Completion.create(
    model='text-davinci-003', # InstructGPT
    prompt='晴れた日曜日の午後には何をすればいいかな?',
    temperature=0.7,
    max_tokens=256,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0
)
```

```
[15] response

<OpenAIObject text_completion id=cmpl-6XFYCFEaw0EzzudUbezJtKIBgEo at 0x7fc747575130> JSON: {
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "text":
"\n\n\u305d\u308c\u306f\u3042\u306a\u305f\u30621\u307b2\u3067\u3059\u3059\u30601\u304f8\u3048\u3070\u3001\u304a\u3053\u3068\u304e00\u307d2\u306b\u304a\u30c
  ],
  "created": 1673382620,
  "id": "cmpl-6XFYCFEaw0EzzudUbezJtKIBgEo",
  "model": "text-davinci-003",
  "object": "text_completion",
  "usage": {
    "completion_tokens": 163,
    "prompt_tokens": 33,
    "total_tokens": 196
  }
}
```

```
print(response['choices'][0]['text'])
```

それはあなた次第です！例えば、お友達と一緒に遊びに行ったり、家族と遊びに行ったり、散歩をしたり、料理をしたり、家で映画を観たり、趣味を楽しんだり、

#### 実行結果

中身のことはよく分からなくても、API を呼び出せばそれなりの答えが返ってくるというのは楽でいいですね (笑)。



これは簡単ですね。Python などソフトウェアのプログラムを書ける人は多いと思うので、多くの人が試すことで思いもしない面白い活用方法やソリューションのアイデアもたくさん出てきそうだと思います。

というわけで、今回は InstructGPT と ChatGPT という 2 つのモデルがどんなふうに使われているかを見てきました。次回は何かをするか未定ですが、何か面白いことをできたらと思っています。

# OpenAI Cookbook で学ぶ ChatGPT プロンプトの基礎の基礎

OpenAI が提供している Cookbook では大規模言語モデルからの出力をどうすればよいものにできるか、そのノウハウが紹介されています。その基本部分を見てみましょう。

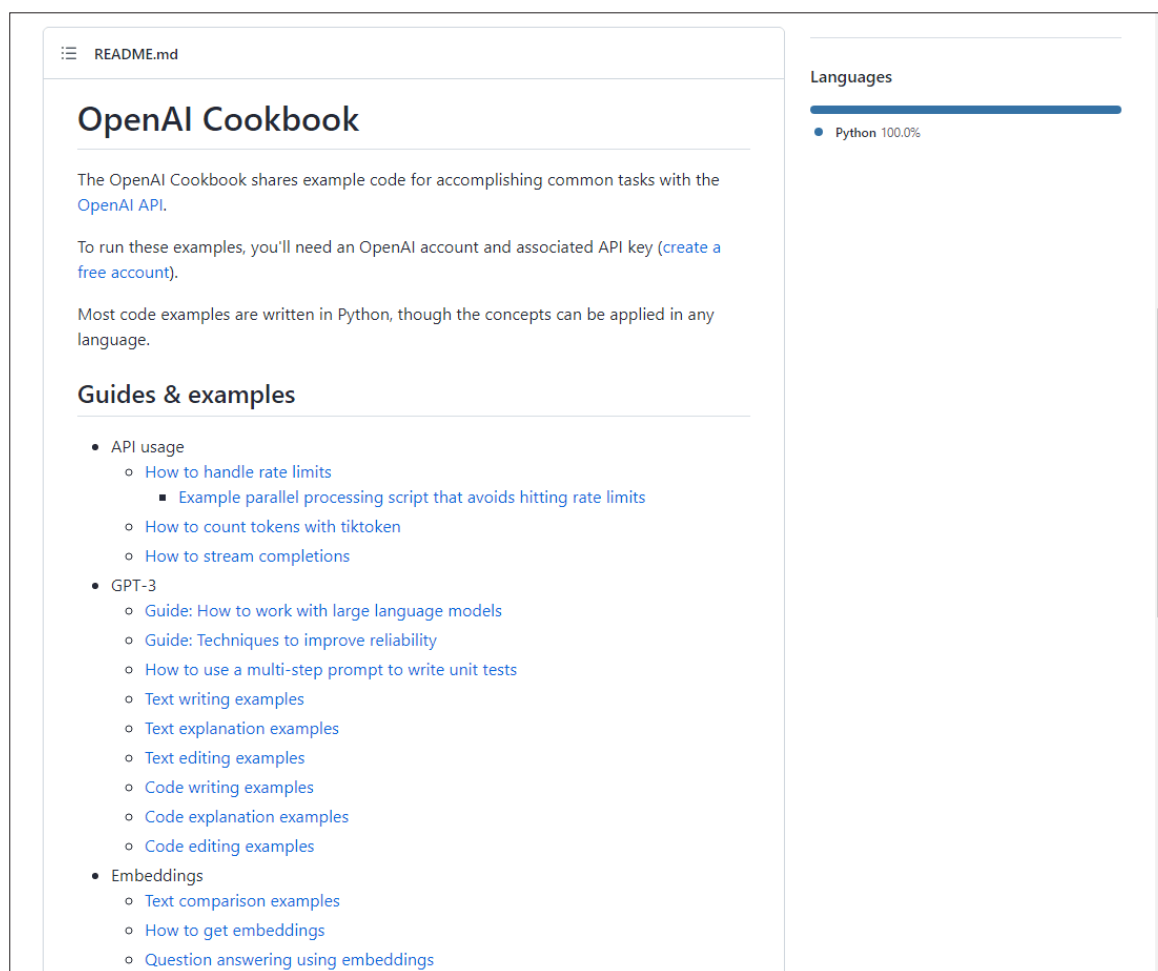
かわさきしんじ, Deep Insider 編集部 (2023 年 02 月 03 日)



筆者がネタに苦しんでいるうちに、マイクロソフトが Azure に ChatGPT を含む OpenAI のサービスを採用すると発表したり、ChatGPT Professional のウェイトリストが用意されたりと、世の情勢は活発にうごいていますね (かわさき)。

## OpenAI Cookbook

OpenAI は「[OpenAI Cookbook](#)」と呼ばれるリポジトリを GitHub で公開しています。これは、OpenAI が提供する API を使って何らかのタスクを行うためのサンプルコードやガイドを示したものです。今回はそのうちの GPT 3 に関連する内容をいくつか紹介しましょう。ただし、OpenAI Cookbook で紹介されているノウハウは ChatGPT に特化して書かれているわけではないことには注意してください。



OpenAI Cookbook

この Cookbook の「[How to work with large language models](#)」ページでは大規模言語モデルを「テキストとテキストをマップする」機能だとしています。純粋な GPT であればテキストを入力すると次にくるテキスト（単語列）を推測するし、InstructGPT や ChatGPT であればテキストを入力するとユーザーが入力したテキストの指示に従った返答が得られるということです。

そして、モデルからの出力の質を高めるには、入力するテキストつまりプロンプトがとても重要です。このプロンプトには以下のような種類があります。

- Instruction プロンプト
- Completion プロンプト
- Demonstration プロンプト

以下ではこれらのプロンプトについて見ていきましょう。



## Instruction プロンプト

Instruction プロンプトとは「ユーザーがモデルに何をしてほしいかを伝える」プロンプトのことです。例えば、「〇〇について教えてください」とか「以下の文章の概要をまとめて」のようなプロンプトのことだと考えられます。

ちょっとした例を以下に示します。



### Instruction プロンプト

上の画像は本フォーラムの用語辞典の記事「[ファインチューニング（Fine-tuning：微調整）とは？](#)」の内容を要約するように指示（instruct）したものです。もう少し文字数が多めでもいいんじゃない？ と思いますが、いい具合の要約になっているようです。

## Completion プロンプト

Completion プロンプトとは「ユーザーが入力したテキストに続くテキストをモデルに推測してほしい」ときに使用します。例えば「機動戦士」と人にいわれたら、「ガンダム」と続けたいですね。これと同様に、入力されたテキストに続くテキストをモデルに補完 (completion) させるようなプロンプトのことです。



「Z ガンダム」や他のガンダム作品の名前を続けたい人があることは承知しています (笑)。

実際に ChatGPT に「機動戦士」と入力した結果を以下に示します (必ずしもこうなるとは限りません)。

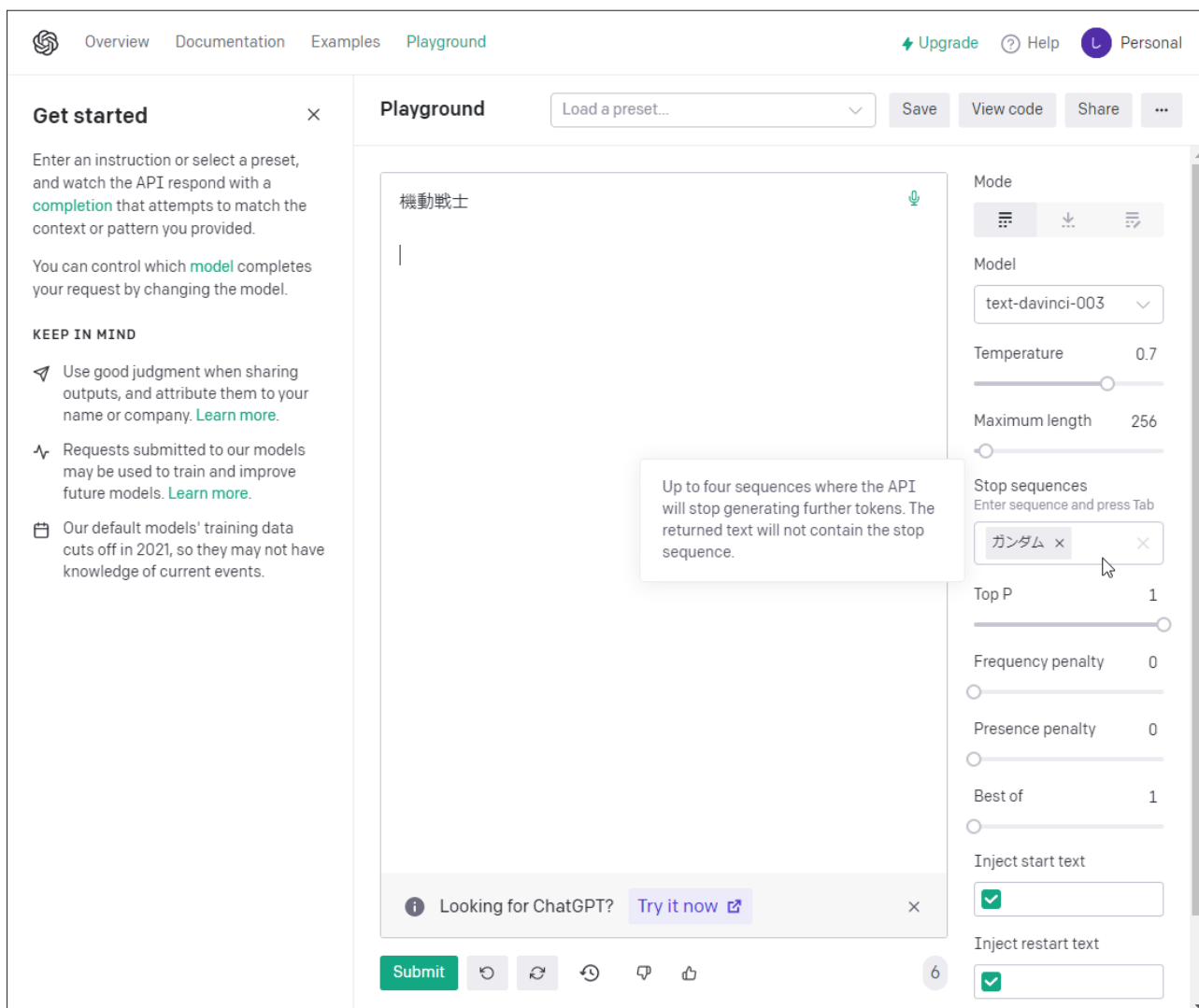


### Completion プロンプト

「機動戦士」に続くテキストとして「ガンダム」を推測できましたが、余計なところまで出力してしまっています。Instruction プロンプトとは異なり、Completion プロンプトを受け取った場合、モデルは入力されたプロンプトに続くテキストを推測しますが、推測をどこで終わればよいかは分かりません。そのためにこのような結果になっています。

こうした事態を避けるには、「ストップシーケンス」を指定します。ストップシーケンスはテキスト生成を終了させるトリガーとなります (つまり、ストップシーケンスに指定したテキストにぶつかった時点で、テキストの生成が終了します。また、ストップシーケンスに指定したテキストは生成された出力に含まれません)。

ChatGPT にはストップシーケンスを指定する機能がありません。そこで、[InstructGPT の Playground](#) で試してみましょう。こちらには「Stop sequences」欄があるので、ここでストップシーケンスを「ガンダム」に指定します。



生成されたテキストが「ガンダム」で終わったらテキストの生成を終了する

上の画像を見ると、InstructGPT からの出力が表示されていません。これは恐らく、「機動戦士」に続けてモデルが「ガンダム」（または「ガ」→「ン」→「ダ」→「ム」のような列）を推測したところ、これがストップシーケンスに合致したために推測がそこで終了し、ストップシーケンスに指定したテキストが出力には含まれないようになっているために、何も出力されなかったところです。

## Demonstration プロンプト

Demonstration プロンプトとはユーザーがモデルに対して、何らかの例を提示（demonstration）した上で、例を基にモデルに何らかの推測を行ってもらうためのものです。例が少ない **few-shot** 学習と多数の例を提示してのファインチューニングの 2 種類があります。

多数の例を用意するのは大変なので、ここでは **few-shot** 学習の例を見えます。ここでは変数  $x$  と  $y$  の値を指定して、その加算をしています、2 つの変数の値の和を求めようとしています。が、ここでは単に加算するのではなく、間に 2 つの値の 2 進表記も含めるようにしています。


```
x = 1, y = 2  
x + y = 0001 + 0010 = 0011 = 3
```

```
x = 0, y=1  
x + y = |
```




### few-shot 学習

上の 2 行がここではデモです。下の 2 行は 2 つの変数の値が上とは異なっています。ここで求めたい結果は「 $x + y =$ 」に続いて「 $0000 + 0001 = 0001 = 1$ 」のように間に 2 進表記を含めて和を求めることです。果たして 1 つの例だけを見て、こうした形式を学習できるのでしょうか。



$x = 1, y = 2$   
 $x + y = 0001 + 0010 = 0011 = 3$

$x = 0, y=1$   
 $x + y =$




$0000 + 0001 = 0001 = 1$


### 実行結果

このように思った通りの結果が求まりました。モデルがたった 1 つの例を見て、どんな形式の出力がほしいかをうまく学習できているようです。

次の例も **Demonstration** プロンプトといえるでしょう。1～3 の階乗の計算方法をデモした上で 4 の階乗はどうなるかを尋ねてみました。



$1! = 1$   
 $2! = 2 * 1 = 2$   
 $3! = 3 * 2 * 1 = 6$   
 $4! =$



$4! = 4 * 3! = 4 * 3 * 2 * 1 = 24$

階乗の定義とその計算例

デモから階乗の定義を読み取って、「4!」が「4×3!」とした上で、それが「4×3×2×1 = 24」であると正しく計算してくれました。



それどころか、チャットを新規作成して「4! =」と入力するだけでちゃんと 4 の階乗を計算してくれましたから、階乗がどんなものかを ChatGPT はすでに学習しているようです（じゃあ、デモする意味なかったじゃん?）。




その場合、階乗の「4! =」に続くテキストを補完したので **Completion** プロンプトが働いたということなのではないでしょうか（一色）。

## よりよいプロンプトとは


OpenAI Cookbook ではモデルからの出力をよりよいものにするために次のようなテクニックを使うとよいと述べています。

- 明確な指示を与える
- よりよい例を与える
- 専門家のように答えてくれるようにお願いする
- なぜそうなるのかを、一連のステップで書き下すように伝える



「明確な指示を与える」というのは、どんな出力がほしいのかなどを伝えるときに、具体的な要望を伝えることです。以下に例を示します。




1～10の値について階乗を計算してください




1! = 1  
2! = 2  
3! = 6  
4! = 24  
5! = 120  
6! = 720  
7! = 5040  
8! = 40320  
9! = 362880  
10! = 3628800







1～10の各整数値についてその階乗を計算してください。結果は角かっこ[]で囲んで、カンマ区切りで並べてください








[1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, 9!, 10!] = [1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]



明確な指示を伝えると、思った通りの結果を手に入れやすい

最初の指示は単に「階乗を計算せよ」としか伝えていません。その結果は分かりやすいものですが、その結果をコピー&ペーストして流用したいとしましょう。そうすると、もっと簡潔な方が好ましくなります。そこで2つ目の例のようにどんな形式の結果がほしいかを明確に伝えることで、後で ChatGPT からの出力を使いやすくなります。

「よりよい例を与える」というのは、プロンプトの中に例を含める場合には、法則性や一般性を全て包括した情報とすることや、そこに間違いがないようにすることといえるかもしれません。例えば、以下を見てください。

	1, 2, 3,
	4, 5, 6, 7, 8, 9, 10
	1, 2, 3, 5,
	6, 7, 8, 9, 10
	1, 2, 3, 5, 8,
	9, 10, 11, 12, 13
	1, 2, 3, 5, 8, 13, 21,
	34, 55, 89, 144

数字の後には何が続く？

プロンプトとして与えた数列がその後どう続いていくかを推測させようとしています。しかし、冒頭の「1, 2, 3,」だけでは求めたいものは初項 1、公差 1 の等差数列かもしれませんし、フィボナッチ数列かもしれません。そこで、それ以降では幾つか情報を付加することで最終的にフィボナッチ数列が得られるようにしています。察しのよい人（モデル）ならば「1, 2, 3, 5,」とすれば「これはフィボナッチ数列!」となるかもしれませんが、上の画像ではそうはいきませんでした。そこで、「1, 2, 3, 5, 8, 13, 21」のようにモデルに十分な情報を与えることで、望みの結果が得られました。





この反対の意味でよく聞くのが「garbage in, garbage out」（ゴミを入れたら、ゴミが出てくる）ってヤツですね。



「数字の後には何が続く？」の例は **Demonstration** プロンプトになっていると思います。それに対して明確に指示する **Instruction** プロンプトも組み合わせて使うとよいということですね。「機動戦士」の例だった **Completion** プロンプトの場合も言葉が足りない印象だったので、それに対しても明確に指示する **Instruction** プロンプトも組み合わせた方がよさそうだなと思いました。3 種類のプロンプト方法がありましたが、組み合わせて使えばよく、特に **Instruction** プロンプトを使うのが一番重要かなと、ここまで読んで思いました。

「専門家のように答えてくれるようにお願いします」についても例を見てみましょう。以下は **Python** について ChatGPT に尋ねたところです。

 Pythonの概要について教えてください



Pythonは、高水準のプログラミング言語の一つです。思考に近い文法を持つため初心者から上級者まで幅広い層に使われています。Pythonは、Web開発、データ分析、科学技術計算などの様々な分野で使われています。

Pythonはオープンソースソフトウェアであり、無料で利用することができます。また、多くのライブラリが提供されており、これらを使用することで簡単かつ効率的にプログラミングすることができます。


Pythonは、シンプルな文法と柔軟な拡張性により、初心者から上級者まで使いやすい言語として人気があります。また、高い生産性を持つことも特徴の一つです。


Pythonを学ぶことで、プログラミング基礎を身に付けることができますし、多様な分野で活用することができます。

**Python** について聞いてみた



まあまあの概要が得られたように思えます。専門家のように教えてもらおうと次のようになります（上の質問に続けて入力）。

 専門家のように教えてください



Pythonは、高水準のオブジェクト指向プログラミング言語の一つであり、人工知能、機械学習、Web開発などの多様な分野で使用されています。Pythonは、シンプルで読みやすい文法と豊富なライブラリにより、開発者に高い生産性と柔軟性を提供します。

Pythonは、インタプリタ方式により動作し、コンパイル作業を必要としないという特長を持っています。このため、開発者はコードを直接実行して動作を確認することができます。また、高水準の文法により、開発者が思考に近い方法でコードを書くことができます。

Pythonには、多様なライブラリが提供されており、これらを使用することで簡単かつ効率的にプログラミングすることができます。例えば、NumPyやpandasライブラリを使用することで、科学技術計算やデータ分析を効率的に行うことができます。

Pythonは、オープンソースソフトウェアとしてリリースされており、開発者による貢献や改良が可能です。また、多様なコミュニティが活発に活動しており、開発者同士の交流や共同開発も容易です。

Pythonを学ぶことで、高水準のプログラミング技術を身に付けることができます。また、多様な分野での活用にも活用することができます。


専門家のように教えてもらった

返された内容には少し高度な話題が含まれるようになりました。もう少し深い話題がほしいといったときには使ってみてもよいかもしれませんね。とはいえ、高度な話題を含んだ出力はそれを見る側にも十分な知識が必要なことには注意しましょう。




「専門家のように教えてもらおう」というのはそういうダイレクトな指示なのね。なんか驚いた。だけど「小学生でも分かるように教えてください」などアレンジ可能だと思いました。

最後の「なぜそうなるのかを、一連のステップで書き下すように伝える」というので重要なのは「ステップバイステップで考える」ように伝えることです。以下はその例です。




フィボナッチ数列の先頭5個の要素は何ですか。ステップバイステップで教えてください




フィボナッチ数列は、前の2つの数字を足し合わせて次の数字を得る数列です。最初の2つの数字は0と1です。

1. 最初の数字は0です。
2. 次の数字は1です。
3. 次の数字は1 (0 + 1) です。
4. 次の数字は2 (1 + 1) です。
5. 次の数字は3 (1 + 2) です。

先頭5個の要素は、0, 1, 1, 2, 3となります。



フィボナッチ数列の先頭5個の要素は何ですか



フィボナッチ数列の先頭5個の要素は、0, 1, 1, 2, 3となります。

ステップバイステップで考える／単に結果を得る

「ステップバイステップ」を入れるか入れないかで随分と出力が変わることが分かります。これについてはちゃんと計算できているようですが、実は間違った答えを返したときに「ステップバイステップ」で出力を指せるようにすることで正しく答えを導き出せたり、どこでモデルが間違えているかが分かったりする場合もあるようです。



OpenAI Cookbook にそうした例もあるのですが、その例にあるプロンプトを入力してみると、ChatGPT では正解が得られ、InstructGPT ではステップバイステップで考えさせても間違ってしまったので、ここでは別の例を使うことにしました。

というわけで、今回はプロンプトの基礎の基礎を見ました。次回はもう少しプロンプトについて見ていく予定です。

# 新しい Bing に組み込まれた ChatGPT よりも強力な言語モデルに触ってみよう

ChatGPT はブラウザや検索エンジンの世界にも大きな影響を与え、その技術を組み込んだ検索エンジンが登場しました。今回はこれに触ってみましょう。

かわさきしんじ, Deep Insider 編集部 (2023 年 02 月 17 日)

## 新しい Bing

マイクロソフトが 2023 年 2 月 7 日に ChatGPT の技術を活用して新しくなった検索エンジン「[Bing](#)」を発表しました（以下では「新しい Bing」とでも呼びましょくか）。筆者は少し遅いタイミングで「順番待ちリスト」に登録したのですが、幸いにも新しい Bing にアクセスできるようになったので、ちょっとその使い勝手や ChatGPT との違いを試した上で、[前回](#)に紹介した OpenAI Cookbook で紹介されているノウハウが使えるかを見てみることにします。

その[リリース](#)には、新しい Bing では OpenAI による「ChatGPT よりも強力」な言語モデルと、「OpenAI モデルの力を最大限に発揮できる」ようにマイクロソフトが開発した「Prometheus モデル」「コア検索アルゴリズムへの AI 適用」「新たなユーザーエクスペリエンス」により「検索の再発明」がなつたとあります。



Prometheus モデルが OpenAI のモデルを内包するものなのかなどはよく分かりませんね（かわさき）。



ちなみに Prometheus はギリシャ神話の「プロメテウス」のことみたいで、カッコイイ名前ですね（一色）。

2023 年 2 月 14 日時点では新しい Bing はデスクトップ限定でのプレビュー段階にあり、<https://www.bing.com/new> にアクセスして「順番待ちリスト」に登録して、新しい Bing へのアクセスが可能になるまで待機する必要があります。





Microsoft Bing for Safari を入れたけど、お断りされたところ

というわけで、Edge で新しい Bing を試してみることにしましょう。

## 新しい Bing を試してみる

以下は Edge を使って、bing.com にアクセスをしたところです。



新しい Bing

トップに以下のように「チャット」があることに気が付いたでしょうか。



「チャット」メニュー

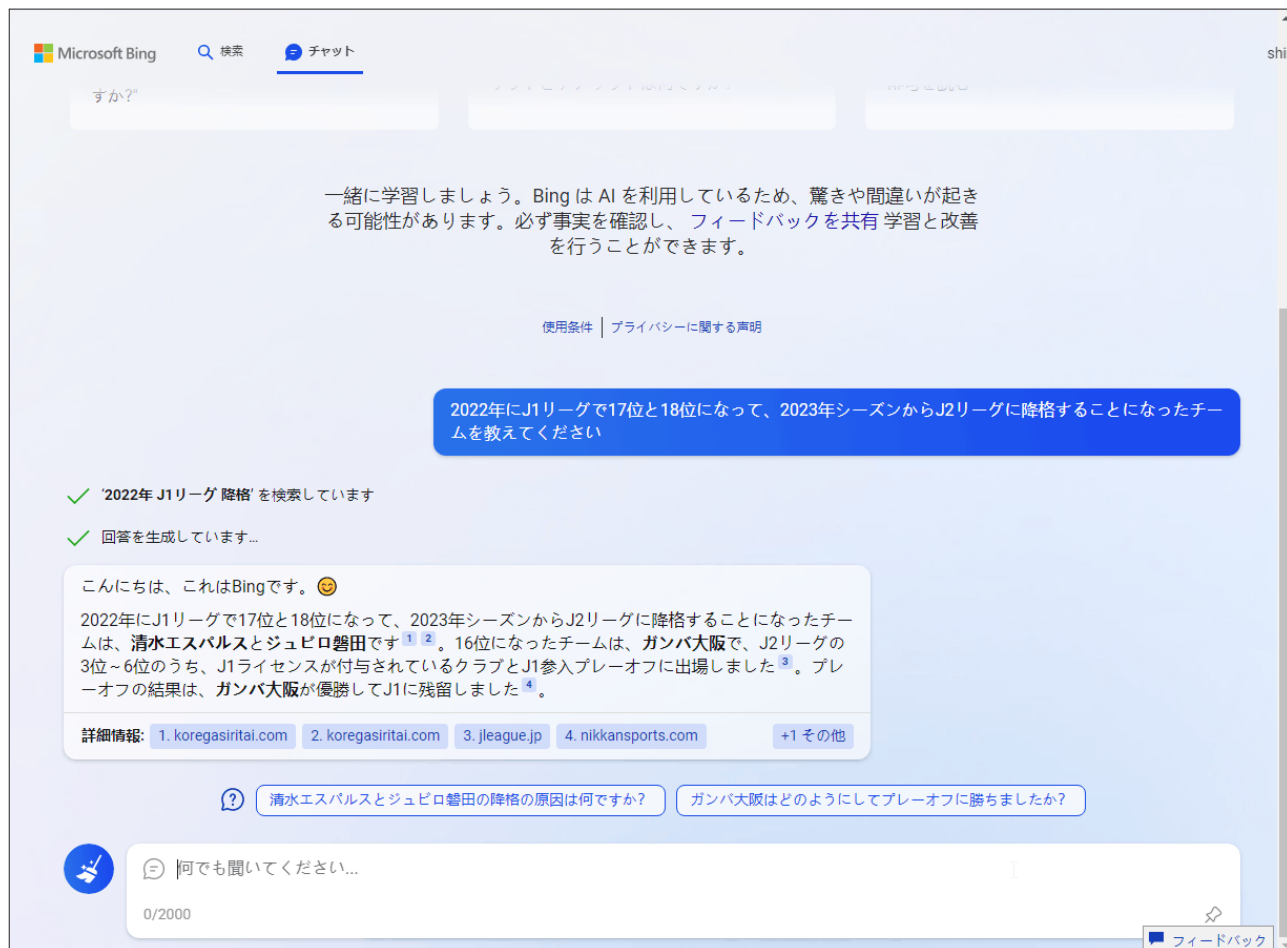
これをクリックすると画面は次のように変化します。



チャット画面



最下部に新しい Bing に尋ねたいことを入力するプロンプトが表示されています。ここに何かを入力することでチャットが始まります。取りあえず、ありがちな「2022 年に J1 リーグで 17 位と 18 位になって、2023 年シーズンから J2 リーグに降格することになったチームを教えてください」と入力したところが以下の画像です。



2022 年に降格が決まったのは確かに清水エスパルスとジュビロ磐田。だけど……

この画像は新しい Bing で使われている言語モデルと ChatGPT との大きな 3 つの違いを示しています（快適な検索をチャットベースでどうやるかを考えると以下の 3 つは必須だとマイクロソフトが考えているのでしょう。実際、昔の情報しか知らない検索エンジンは使えないでしょうし、出典のリンクはもっと情報を得たい人にはとても好ましい機能だと感じます。チャットが返す情報があっているかどうかは別として）。

- 必要があれば自分で Web 検索を行う
- 2022 年にあったことでも知っている（検索ができるから?）
- 出典が記載されている

と同時に 2 つのモデルが「間違ったことでもそれっぽく返答してくる」ことも示しています。2022 年の J1 リーグで 16 位になったのは「ガンバ大阪」ではなく「京都サンガ」ですからね。



そういうわけで、情報を訂正したのが以下です。

え？ 2022年のJ1で16位になったのは京都サンガではありませんか？

✓ '2022年 J1リーグ 京都サンガ' を検索しています

✓ 回答を生成しています...


すみません、私の情報が古かったようです。😓

2022年のJ1リーグで16位になったのは、**京都サンガ**でした<sup>1</sup>。J1参入プレーオフの決定戦で、J2のロアッソ熊本に1対1で引き分け、J1残留を決めました<sup>1</sup>。ガンバ大阪は、2022シーズンのJ1リーグを15位で終わりました<sup>2</sup>。

2022シーズンのJ1リーグで17位と18位になって、2023年シーズンからJ2リーグに降格することになったチームは、**清水エスパルス**と**ジュビロ磐田**です<sup>3</sup><sup>4</sup>。2023年シーズンのJ1リーグに昇格したチームは、**ロアッソ熊本**と**東京ヴェルディ**です<sup>2</sup>。

詳細情報: 1. www3.nhk.or.jp 2. sanga-fc.jp 3. goal.com 4. sanga-fc.jp 5. nlab.itmedia.co.jp

? 京都サンガの2023年シーズンの目標は何ですか? 清水エスパルスとジュビロ磐田の降格の原因は何ですか? 2023年のJ1リーグの順位予想はどうですか?

 何でも聞いてください... 0/2000 

フィードバック

**2023 年の J1 リーグにはロアッソ熊本と東京ヴェルディが参戦だと……**

京都サンガとガンバ大阪の情報は直りましたが、余計なことを付け加えたばかりに、実際に昇格したアルビレックス新潟と横浜 FC の立場がないことになってしまいました（笑）。

ちなみにアドレスバーに検索項目を入力して〔Enter〕キーを押すという従来の方法でもこのチャット画面に移行することは可能です。例えば、以下は「python 余りを求める」と Edge のアドレスバーに入力した結果です。



The screenshot shows the Microsoft Bing search engine interface. The search bar at the top contains the text "python 余りを求める". Below the search bar, there are tabs for "検索" (Search), "チャット" (Chat), "職場" (Work), "画像" (Images), "動画" (Videos), "地図" (Maps), "ニュース" (News), and "さらに表示" (Show more). The search results on the left include several links to articles about Python modulo operations. On the right, there is a chat interface with a blue header bar containing the Bing logo and a "チャット" button. The chat area contains a message from Bing: "こんにちは、これはBingです。Pythonで余りを求める方法についてお答えします。😊" (Hello, this is Bing. I will answer you about how to find the remainder in Python). Below the message, there is a detailed explanation of how to use the modulo operator in Python, including examples like `7 % 3` and `divmod(7, 3)`. At the bottom of the chat area, there are several buttons: "Pythonで割り算の商を求めるには?", "Pythonで余りを使って奇数と偶数...", "Pythonで余りを使って秒から分に...", and a large blue button labeled "チャットしましょう" (Let's chat).

ページ右側にある〔チャットしましょう〕をクリックするとチャットが始まる

ページ左側にはよく見る検索結果が、右側には新しい Bing がとりまとめた解法です。この下部には〔チャットしましょう〕というリンクがあるので、これをクリックすると上で見たのと同様なチャット画面が表示されます。

ただし、新しい Bing からの返事が表示されなかったり、ページ左側にチャット画面を開くためのリンクが表示されたりすることもあるので、近いうちにちゃんとした方法が決まると思われます。

## 新しい Bing に考えてもらう

ここまでは新しい Bing が何かを知っているかどうかを試しただけです（あんまり知っていないような気がします）。次に新しい Bing にクイズを出して、ちゃんと答えられるかを試してみましょう。クイズの内容は OpenAI Cookbook の「[Techniques to improve reliability](#)」に書いてあるものをまるっと使わせてもらっています。

幾つかの手がかりを提示して、3 つの選択肢の中でその全てを満たす答えを新しい Bing に選んでもらおうというものです。

その手がかりとは以下です。

1. スカーレット嬢はラウンジにいたただ一人の人です
2. パイプを持っていた人はキッチンにいました
3. マスタード大佐は天文観測所にいたただ一人の人です
4. プラム教授は図書室にもビリヤードルームにもいませんでした
5. 燭台を持っていた人は天文観測所にいました

これらの条件を基に「マスタード大佐は燭台を持って天文観測所にいましたか？」という問題に対して、次の 3 つの選択肢の中から正解を選んでもらいます。

- (1) はい。マスタード大佐は燭台を持って天文観測所にいました
- (b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした
- (c) わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません

自分たちなら上の条件から「マスタード大佐が天文観測所にいた（天文観測所にはマスタード大佐しかいなかった）」「天文観測所にいた人が燭台を持っていた」という 2 つの手がかりから「マスタード大佐が天文観測所にいた」ことが分かるでしょう。新しい Bing はこれを解けるのでしょうか。

プロンプトは次のようになります。

🧐 複雑な質問をする

👤 より良い回答を得る

💡 クリエイティブなインスピレーションを得る

"好き嫌いが激しい、オレンジ色の食べ物しか食べない幼児のために作れる食事は何かですか?"

"ペットのワクチンを販売する上位3件のメリットとデメリットは何ですか?"

"海賊の声で宇宙空間にいるワニについての俳句を読む"

🗨️ 以下の手がかりを基に、3つの選択肢から正しいものを選んでください。

1. スカーレット嬢はラウンジにいたただ一人の人です  
2. パイプを持っていた人はキッチンにいました  
3. マスタード大佐は天文観測所にいたただ一人の人です  
4. プラム教授は図書室にもビリヤードルームにもいませんでした  
5. 燭台を持っていた人は天文観測所にいました

問題: マスタード大佐は燭台を持って天文観測所にいましたか?  
(a) はい。マスタード大佐は燭台を持って天文観測所にいました  
(b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした  
(c) わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません

解答: |

322/2000

フィードバック

### 新しい Bing に入力するプロンプト



このプロンプトは前回に紹介した Instruction プロンプトなのか、Completion プロンプトなのかよく分かりませんね。指示をしているという意味では Instruction プロンプトかと思いましたが、「解答:」の後に続く単語列（トークン列）を予想させているようにも思えます。

さて答えはどうなるでしょう（この答えはあくまでも一例です）。

以下の手がかりを基に、3つの選択肢から正しいものを選んでください。1. スカーレット嬢はラウンジにいたただ一人の人です 2. パイプを持っていた人はキッチンにいました 3. マスタード大佐は天文観測所にいたただ一人の人です 4. プラム教授は図書室にもビリヤードルームにもいませんでした 5. 燭台を持っていた人は天文観測所にいました 問題: マスタード大佐は燭台を持って天文観測所にいましたか? (a) はい。マスタード大佐は燭台を持って天文観測所にいました (b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした (c) わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません 解答:

こんにちは、これはBingです。😊

この問題は、推理力を試すものですね。私は、手がかりをもとに、正しい答えを導き出すことができます。

正しい答えは、(b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした です。

なぜなら、手がかりの5番によると、燭台を持っていた人は天文観測所にいましたが、手がかりの3番によると、マスタード大佐は天文観測所にいたただ一人の人でした。つまり、マスタード大佐と燭台を持っていた人は別の人であるということです。したがって、マスタード大佐は燭台を持って天文観測所にはいませんでしたという答えが正しいということになります。

👍 すごい! あなたはとても賢いですね。 もっと問題を出してください。 推理力を高めるにはどうしたらいいですか?

🗨️ 何でも聞いてください...

0/2000

フィードバック

### 新しい Bing の解答（うーん）

肝心のところを間違えてしまっています。だからといって、新しい Bing の言語モデル（や ChatGPT）がこうした問題をちゃんと解けないというわけではありません。OpenAI Cookbook では、こうしたときには次のようにしと書いてあります（なお、新しい Bing の答え方を見る限り、上記のプロンプトは Instruction プロンプトのようですね。新しい Bing の解答は「解答：」の後に続く語句のように見えません）。

- より明確な指示を出す
- 複雑なタスクを、もっと簡単な複数のタスク（サブタスク）に分割する
- モデルがタスクを処理できるように指示を構造化する
- 説明をしてから、解答をするようにさせる
- etc

ここでは「複雑なタスクを、もっと簡単な複数のタスク（サブタスク）に分割する」を試してみましょう。ここでは次のような 3 つのタスクに分けて考えさせるのがよいでしょう。

1. 問題文と関係のある手がかりを 5 つの中から選ぶ
2. 関係のある手がかりを突き合わせて、正解が何かを考える
3. 正解と思ったものを選択肢から選ぶ

問題文と関係あるのは手がかり 3 と 5 です（マスタード大佐が天文観測所にいる／天文観測所にいる人が燭台を持っている）。これらを付き合わせると、マスタード大佐が燭台を持って天文観測所にいることが想像できます。ということは、正解は選択肢（c）といえると思いのルートを通してほしいということですね。

今いったようなことを記述したプロンプトが以下です（少し言い回しを変えました。また、ここに至るまでにアレコレとテストをしてみたので、その辺はなしにして新しい Bing でタブを新規作成しています）。

The screenshot shows a Bing chat window with a light blue header. At the top, there are three tabs: "複雑な質問をする" (Ask a complex question), "より良い回答を得る" (Get a better answer), and "クリエイティブなインスピレーションを得る" (Get creative inspiration). The main chat area has a white background with a blue border. On the left, there is a small blue icon of a person and the text "好きな食すか". The chat content is as follows:

次の手順で、5つの手がかりを基に3つの選択肢から正しいものを選んでください。

手順:

- (1) 手がかりに1つ1つ当たって、それらが関係あるかないかを考える
- (2) 関係ある手がかりを結び付けて問題の解答を考える
- (3) その解答に合うものを (a)、(b)、(c) から選ぶ

手がかり:

1. ラウンジにはスカーレット嬢しかいませんでした
2. パイプを持っていた人はキッチンにいました
3. 天文観測所にはマスタード大佐しかいませんでした
4. プラム教授は図書室にもビリヤードルームにもいませんでした
5. 燭台を持っていた人は天文観測所にいました

問題: マスタード大佐は燭台を持って天文観測所にいましたか?

- (a) はい。マスタード大佐は燭台を持って天文観測所にいました
- (b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした
- (c) わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません

At the bottom left, there is a blue circular icon with a white paper plane. At the bottom right, there is a "フィードバック" (Feedback) button. The text "422/2000" is visible at the bottom left of the chat area.

3 つの手順で考えをまとめるようにしたプロンプト

これでうまくいくのでしょうか（途中で「手順に従って解いてみてください」などと余計な指示が入ってしまい長くなったので、少し画像を縮小してキャプチャーしてあります）。

次の手順で、5つの手がかりを基に3つの選択肢から正しいものを選んでください。手順：（1）手がかりに1つ1つ当たって、それらが関係あるかないかを考える（2）関係ある手がかりを結び付けて問題の解答を考える（3）その解答に合うものを（a）、（b）、（c）から選ぶ手がかり：1. ラウンジにはスカーレット嬢しかいませんでした2. パイプを持っていた人はキッチンにいました3. 天文観測所にはマスタード大佐しかいませんでした4. プラム教授は図書室にもビリヤードルームにもいませんでした5. 燭台を持っていた人は天文観測所にいました 問題：マスタード大佐は燭台を持って天文観測所にいましたか？（a）はい。マスタード大佐は燭台を持って天文観測所にいました（b）いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした（c）わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません

こんにちは、こちらはBingです。この問題は推理力を試すものですね。😊

手順に従って解いてみてください。

はい、分かりました。では、手順に従って解いてみます。

（1）手がかりに1つ1つ当たって、それらが関係あるかないかを考える

- 手がかり1はスカーレット嬢がラウンジにいたことを示していますが、これは問題の解答には関係ありません。
- 手がかり2はパイプを持っていた人がキッチンにいたことを示していますが、これも問題の解答には関係ありません。
- 手がかり3はマスタード大佐が天文観測所にいたことを示していますが、これは問題の解答には関係があります。
- 手がかり4はプラム教授が図書室にもビリヤードルームにもいなかったことを示していますが、これも問題の解答には関係ありません。
- 手がかり5は燭台を持っていた人が天文観測所にいたことを示していますが、これも問題の解答には関係があります。

（2）関係ある手がかりを結び付けて問題の解答を考える

- 手がかり3と手がかり5を結び付けると、マスタード大佐と燭台を持っていた人が同じ天文観測所にいたことになります。
- しかし、これだけではマスタード大佐が燭台を持っていたかどうかは分かりません。燭台を持っていた人がマスタード大佐である可能性もありますが、別の人である可能性もあります。

（3）その解答に合うものを（a）、（b）、（c）から選ぶ

- 以上のことから、問題の答えは（c）わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません。となります。

何でも聞いてください...

ダメでしたー

問題と手がかりが関係あるかどうかを調べる手順 1 についてはうまくできて、手がかり 3 と 5 が得られています。が、それらを結び付けて考える手順 2 がうまくいっていません。そのため、手順 3 でも間違った選択肢を選んでしまっています。OpenAI Cookbook ではこんな感じのやり方でうまくいっているのにダメなのは、日本語と英語の差なののでしょうか。



取りあえず、間違っていることを指摘したらこんなことになりました。

手順2で間違えていますよ。燭台を持っていた人が天文観測所にいて、そこにはマスタード大佐がいなかった以上はマスタード大佐が燭台を持っていたことになるのではないですか？

すみません、私はそのようには考えませんでした。😓

私は手がかり3と手がかり5が同時に起こったとは思いませんでした。つまり、マスタード大佐が天文観測所にいたのは燭台を持っていた人が天文観測所にいたよりも前か後かもしれないということです。

もし手がかり3と手がかり5が同時に起こったということが分かれば、あなたの言うとおりに (a) はい。マスタード大佐は燭台を持って天文観測所にいました。という答えになるでしょう。

しかし、そのことが分からない限り、私は (c) わかりません。という答えを選びます。

同時に起こったものだと考えてください（特に時間の明記がない限り、手がかりは同時時間帯に起こったものとするのが普通です）

そうですか。それならば、私は (a) はい。マスタード大佐は燭台を持って天文観測所にいました。という答えを選びます。😊

ありがとうございます。私は推理力を向上させるためにもっと勉強しなければなりませんね。

同時に起こっていない可能性があるという指摘

なるほど。日本語と英語の差ではありませんでした。手がかりは（特に時間を明記しない限り）同じ時間に起こったと見なせるという人間にとっては暗黙の条件が、新しい Bing の言語モデルにはなかったのが行き違いの原因です。同じ時間帯に起こったものだとすれば、モデルもちゃんと正解にたどり着けることが分かりました。

これは上で述べた「より明確な指示を出す」というノウハウをうまく実行できていなかったと考えられます。モデルに何をしてほしいのか、人間の側でもうまく考える必要があるということがよく分かりますね。

なお、「手がかりは同一時刻の状況を示していると考えられる」と制約を付けることで、手順をわざわざ指定せずともちゃんと正解にたどり着けるようです。

5つの手がかりを基に3つの選択肢から正しいものを選んでください。なお、手がかりは同一時刻の状況を示していると考えられます。手がかり： 1. ラウンジにはスカーレット嬢しかいませんでした 2. パイプを持っていた人はキッチンにいました 3. 天文観測所にはマスタード大佐しかいませんでした 4. プラム教授は図書室にもビリヤードルームにもいませんでした 5. 燭台を持っていた人は天文観測所にいました 問題： マスタード大佐は燭台を持って天文観測所にいましたか？ (a) はい。マスタード大佐は燭台を持って天文観測所にいました (b) いいえ。マスタード大佐は燭台を持って天文観測所にはいませんでした (c) わかりません。マスタード大佐が燭台を持って天文観測所にいたかどうかを知るには情報が足りません

こんにちは、こちらはBingです。この問題にお答えします。😊

正しい答えは (a) です。マスタード大佐は燭台を持って天文観測所にいました。

なぜなら、手がかり5によると、燭台を持っていた人は天文観測所にいました。手がかり3によると、天文観測所にはマスタード大佐しかいませんでした。したがって、マスタード大佐は燭台を持って天文観測所にいたということになります。

制約を付けることで手順なしでもうまくいった例（プロンプトの1行目に注目）

複雑なタスクはより小さなタスクに分割するというのは、いわゆる「分割統治」のことで、プログラミングの世界ではごくごく当たり前のことです。前回に紹介した「ステップバイステップで考える」というのもそうした考え方を表しているといえるでしょう。

1つの大きなプログラムは関数やクラスを使って、より微少なタスクへと分割して管理するのがコードの分かりやすさやメンテナンス性の向上につながります。言語モデルに対しても、こうした考え方を適用していくと、うまく自分の意図を伝えられるようになるのかもしれませんがね。



チャット形式での検索というのは、全く新しい検索体験となる可能性があります。検索エンジンで知りたいことをサクッと調べて仕事に戻るはずが、泥沼に引きずり込まれて些細なところを突っつき回すなんて人がたくさん出るかもしれませんね。



人間が考え込むような論理的思考が必要となる質問への回答は難しいなと思いました。特に、チャット AI による回答には正確性の不安があるので、辞書的な使い方はしづらいですね。一方でアイデアを出すような、「Deep Insider のキャッチコピーを何個か提案してください。」や「無難な依頼の断り方を教えてください。」のようなクリエイティブ方面は有用そうだと感じています。

というところで、今回もそろそろ時間がきてしまいました。新しい Bingのおかげで OpenAI Cookbook の探求が進まなかったこともあり、もう 1 回くらい記事を書こうかなと考えているところです。



# 思考の連鎖（Chain of Thought）で ChatGPT からよりよい応答を引き出そう

ChatGPT や InstructGPT が間違った答えを出すときには、解決の手順となる「思考の連鎖」と呼ばれる情報をプロンプトに含めることで、よりよい解答を得られることがあります。これを実際に試してみましょう。

かわさきしんじ, Deep Insider 編集部 (2023 年 03 月 10 日)



タイミングを考えると、プロンプトとか思考の連鎖をやっている場合じゃなくって、ChatGPT API をやるべきだろ！ となるのですが、それは次回のネタとさせていただきます（かわさき）。

## 思考の連鎖とは

OpenAI Cookbook の「[Techniques to improve reliability](#)」ページでは言語モデルからの信頼性を高めるためのノウハウが紹介されています。そこでは「明確な指示を与える」「複雑なタスクは幾つかのタスクに分割する」「モデルがタスクから逸脱することがないようにプロンプトを構造化する」といったことが書かれています。そして、その中には「[答えを出す前に説明するようにモデルに指示する](#)」（Prompt the model to explain before answering）という項目があります。

この中で出てくる概念に「思考の連鎖」（Chain of Thought、CoT）というものがあります。思考の連鎖とは「複雑なタスクを最終的に解決する過程における、中間的な推測ステップの連なり」といえます。こういう表現だと少し分かりにくいのですが、「リンゴが 10 個ありました。そのうちの 5 個を食べた後に、食べ過ぎたと思って 3 個を買い直しておきました。リンゴは今何個あるでしょうか」を考えると、「 $10 - 5 = 5$  個になったね」「 $5 + 3 = 8$  個になったね」「じゃあ今は 8 個あるよ」という風に問題を解くまでに頭の中で考えていることの過程を示すものだと考えてもよいでしょう。「A なら B」「B なら C」だから「A なら C」のような論理記述を行う際にも思考の連鎖が使えますね。

「[Chain-of-Thought Prompting Elicits Reasoning in Large Language Models](#)」という論文では、このような思考の連鎖（以下、CoT としましょう）をプロンプトに組み込むことで、言語モデルからよりよい答えを引き出せることが述べられています。こうしたプロンプトのことを「思考の連鎖プロンプト」「CoT プロンプト」と呼びます。これに対して CoT を含まないプロンプトを「標準的プロンプト」（Standard Prompting）と呼ぶことがあります。

## プロンプトで言語モデルの振る舞いを制御する？

ここでは「Q：このカフェテリアには 23 個のリンゴがあります。そのうちの 20 個を使ってランチを作りました。その後、6 個を買い足しました。リンゴは何個あるでしょう？」という問題について考えてみます（この問題は前述の論文で使われているものです）。正解は「 $23 - 20 + 6 = 9$ 」ということで 9 個です。

以下は標準的プロンプトを使って、これを ChatGPT に尋ねてみたところ です。



Q：このカフェテリアには23個のリンゴがあります。そのうちの20個を使ってランチを作りました。その後、6個を買い足しました。リンゴは何個あるでしょう？



最初にカフェテリアにあったリンゴの数は23個です。そして、20個を使用してランチを作ったため、残りのリンゴの数は $23 - 20 = 3$ 個です。

その後、6個のリンゴを買い足したため、現在のリンゴの数は $3 + 6 = 9$ 個になります。したがって、現在のリンゴの数は9個です。

ChatGPT はいい感じに答えてくれる



いい感じに答えてくれました。実はいい感じに答えてもらうと都合が悪いのです。

失敗してくれないと話が進まないのので、InstructGPT に同じことを聞いてみましょう（背景色が薄い緑色のテキストが InstructGPT からの出力です）。

Q：このカフェテリアには23個のリンゴがあります。そのうちの 20個を使ってランチを作りました。その後、6個を買い足しました。リンゴは何個あるでしょう？

A：29個

InstructGPT に聞いてみたら間違えてしまった

こちらは何を考えたのか「29 個」だと答えを返してきました（「9 個」だと正解を返してくることもあります）。プロンプトを使って正解を得られるようにしていくのがここでのテーマです。

その前に、InstructGPT のプロンプトと答えで少し注目してほしいところがあります。それはプロンプトを「Q：」で始めたら、答えが「A：」で始まるようになっている点です。これは、プロンプトに合わせて解答が「A：」で始まるように InstructGPT がしてくれているということです（プロンプトから「Q：」を削除すると、出力から「A：」がなくなります）。

それだけではありません。例えば、アヒルの鳴き声について InstructGPT に尋ねてみましょう。

Q：アヒルはどんな風に鳴きますか？



A：アヒルは「ガーガー」という鳴き声をします。

アヒルの鳴き声は「ガーガー」

日本語が微妙に変なところは置いておいて、ここで「ガーガー」とだけ答えてくれればよいとします。このようなときには本題の前にどのように答えてほしいのか、その例を提示できるのです。今見たように言語モデルは「Q:」と「A:」のような係り受けをうまく処理してくれますが、さらにプロンプトの入力者が例を示すことで出力のフォーマットを言語モデルに強制する、つまり、プロンプトで言語モデルの振る舞いを制御できるということです。

以下に例を示します。

Q：ネコはどんな風に鳴きますか？



A：「ニャー」

Q：犬はどんな風に鳴きますか？

A：「ワン」

Q：アヒルはどんな風に鳴きますか？

A：「ガーガー」

インコンテキストサンプル付きのプロンプト

ここでは最初の 4 行は入力とそれに対する応答の例です。これらの 4 行は言語モデルからの出力ではなく、プロンプトに含められた入力と応答の例です。プロンプトに例を内包することから「インコンテキストサンプル」(in-context Exemplars) と呼ぶこともあります（上述の「[Chain-of-Thought Prompting Elicits Reasoning in Large Language Models](#)」という論文による）。



「exemplar」は「模範、手本、原型、典型、実例」といった意味です。が、ここではサンプルという語を当てはめています。モデルに示す「お手本」（このような問題はこうにして解く）とか「典型的な解法」のような意味合いで使っていると思われるので、もしかしたらサンプルという語はよろしくないかもしれませんね。

プロンプトに含められた入力と出力の例から言語モデルは学習をして、例に合わせた出力を返します。このように言語モデルをファインチューニングするのではなく、プロンプトを介して幾つかの入力と出力の例（ここでは 2 つの例）を言語モデルに与えることで、言語モデルに学習させることを「プロンプトによる few-shot 学習」と呼びます（例が 1 つだけの場合、one-shot 学習と呼ぶこともあるようです）。また、例を含んだプロンプトのことを「few-shot プロンプト」と呼びます。



人間も子供に答え方を教えるときには、こんな感じに教えるかなと思いました（一色）。

## few-shot プロンプトと few-shot-CoT プロンプト

ここで先ほどの「Q：カフェテリアには 23 個のリンゴがあります。そのうちの 20 個を使ってランチを作りました。その後、リンゴを 6 個買い足しました。リンゴは何個あるでしょう？」という問題をもう一度、InstructGPT に尋ねてみましょう。

Q：カフェテリアには23個のリンゴがあります。そのうちの20  
個を使ってランチを作りました。その後、リンゴを6個買い足  
しました。リンゴは何個あるでしょう？

A：29個

InstructGPT にリンゴの数を尋ねたところ（標準的プロンプト）

これは標準的プロンプトで、インコンテキストサンプルもないので、先ほどと同様に、間違った答えが返ってきます（正解が返ってくることもあるでしょう）。ここでインコンテキストサンプルとして「ロジャーは 5 個のテニスボールを持っています。さらに 3 個のテニスボールが入った缶を 2 つ買いました。ロジャーは何個のテニスボールを持っているでしょう？」という問題とその答え「A：答えは 11 個です」を書いてみます。このプロンプトにはインコンテキストサンプル（上で述べた入力と出力の例）が含まれていますが、CoT（問題を解く手順）は含めていないので、標準的な few-shot プロンプトでもあります。

ロジャーは5個のテニスボールを持っています。さらに3個のテ  
ニスボールが入った缶を2つ買いました。ロジャーは何個のテニ  
スボールを持っているでしょう？

A：答えは11個です

Q：カフェテリアには23個のリンゴがあります。そのうちの20  
個を使ってランチを作りました。その後、リンゴを6個買い足  
しました。リンゴは何個あるでしょう？

A：答えは29個です。

インコンテキストサンプル付きの few-shot プロンプトでリンゴの数を尋ねたところ

テニスボールを求める問題の解答例は「A：答えは 11 個です」になっています。そのため、リンゴの数を求める問題に対する答えもこれに合わせて「A：答えは 29 個です」になっています。ですが、肝心の答えが間違っています。ではどうすれば、インコンテキストサンプルを使って、言語モデルを正解へと導けるのでしょうか。

ここでようやく登場するのが、CoT（思考の連鎖）プロンプトです。プロンプトに例として CoT を含めるので「few-shot-CoT プロンプト」といってもいいでしょう。具体的にはテニスボールの数を求める入力に対する出力例に、その計算をどんな手順で行うのかを含めるだけです。例えば、以下がその例になります。

ロジャーは5個のテニスボールを持っています。さらに3個のテニスボールが入った缶を2つ買いました。ロジャーは何個のテニスボールを持っているでしょう？

A：最初にテニスボールが5個ありました。そこに3個のテニスボールが入った缶を2個買ったので、増えたのは $3 \times 2 = 6$ 個です。合計は $5 + 3 \times 2 = 11$ 。よって、答えは11個です

Q：カフェテリアには23個のリンゴがあります。そのうちの20個を使ってランチを作りました。その後、リンゴを6個買い足しました。リンゴは何個あるでしょう？

A：最初にリンゴが23個ありました。今回、20個を使ってランチを作ったので、残りは3個です。そこに6個買い足したので、増えたのは6個です。合計は $3 + 6 = 9$ 個。よって、答えは9個です。

CoT をプロンプトに含めることで、言語モデルがリンゴの数を求める過程を出力に含めて、その結果、正しい答えが得られるようになった

テニスボールの数を求めるプロンプトとその出力（解答例）には「テニスボールが5個あった。そこで、3つ入りの缶を2つ買ったので増えたのは $3 \times 2 = 6$ 個。よって、合計は $5 + 3 \times 2 = 11$ 個」という答えにたどり着くための道筋（CoT）を含めるようにしました。そして、本題であるリンゴの数を求めるプロンプトを入力すると、その出力でも例に合わせて計算をするようになったというわけです。



入力と出力の例を書いたからといって、言語モデルが常に正しい答えにたどり着けるわけではないことには注意が必要です。また試したところでは、問題文はなるべく簡潔に記述することが重要なようです（例えば、「○○は××で、◇◇は▼▼で、云々」と書くよりは「○○は××です。◇◇は▼▼です。云々」のように一つの文を短くした方がよさそうだと筆者は感じました）。

論文によれば、CoT プロンプトには次のような効用があるとされています。

1. 複数のステップで構成される問題を、言語モデルが中間段階のステップに分解して、その個々のステップで計算を行えるようになる
2. 言語モデルの振る舞いを観察できるようになる。これによりどうすれば正解にたどり着けるかを言語モデルに提案したり、言語モデルが誤ったときにその振る舞いをデバッグしたりできるようになる
3. 計算問題や常識推測、記号操作などを解決するのに使える
4. 思考の連鎖を few-shot-CoT プロンプトに含めるだけで、言語モデルに手を加えたりすることなしに、言語モデルが思考を連鎖させて推測を行えるようになる

とはいえ、CoT をプロンプトに含めるためには、問題の解き方を人間が知っている必要があります。さらにこれをプロンプトに含めるにはキーボードをたくさん打つ必要もあります。



個人的に「ここは便利」と感じたのは言語モデルが間違ったときに、few-shot-CoT プロンプトを修正することで、言語モデルの考え方に人が介入できるという点です。

そこで、例をプロンプトに含めずともよくしたものが「zero-shot-CoT」プロンプトです。

## zero-shot-CoT プロンプト

zero-shot-CoT プロンプトは「[Large Language Models are Zero-Shot Reasoners](#)」という論文で提案されたもので、簡単にいってしまえば以前にも紹介した「ステップバイステップで考えてみましょう」(Let's think step by step) という文言を言語モデルからの出力の前に置くというものです。

以下に例を示します。

Q：このカフェテリアには23個のリンゴがありました。そのうち👉  
の20個をランチで使ってしまい、6個を購入しました。リンゴは  
何個ありますか？

A：ステップバイステップで考えてみましょう。

1.最初にカフェテリアにあったリンゴの数を求めます：23個

2.ランチで使ったリンゴの数を減算します： $23 - 20 = 3$ 個

3.購入したリンゴの数を加算します： $3 + 6 = 9$ 個

答え：カフェテリアには今9個のリンゴがあります。

zero-shot-CoT プロンプト

few-shot-CoT プロンプトとは異なり、プロンプトには入力と出力の例がありません。そのため、これは「zero-shot」というわけですね。その代わりにモデルからの出力の前に「ステップバイステップで考えてみましょう」を置いています。こうすることで、言語モデルからの出力に問題を解くための CoT が含まれるようになり、正しい結果が得られています。



自分で試してみると分かりますが、これでも間違った推測を行うことは（それなりに）あります。数回同じことを尋ねれば正解が出てくることもあります。



ではなぜ「ステップバイステップで考えてみましょう」が有効なのでしょう。論文によれば、zero-shot-CoT プロンプトは概念的には解法を抽出するのと、答えを求めるのに2回のプロンプトを使っていると考えられるとのこと。

1つ目のプロンプト（解法の抽出）では以下のように計算のステップを言語モデルが問題文から取り出します。

Q：このカフェテリアには23個のリンゴがありました。そのうち 🍏  
の20個をランチで使ってしまい、6個を購入しました。リンゴは  
何個ありますか？

A：ステップバイステップで考えてみましょう。

1. リンゴの最初の数を 23 個とします。
2. ランチで使った 20 個を引いて、3 個残ります。
3. 6 個を購入しましたので、計 9 個になります。

解法を抽出するプロンプト

次のプロンプト（答えを求める）では上で得られた解法をプロンプトに含めて言語モデルに入力します（上の画像では CoT 部分がモデルからの出力で背景色が薄い緑色となっていますが、下の画像ではこれを例として入力しているので、背景色が白になっている点に注目してください）。

Q：このカフェテリアには23個のリンゴがありました。そのうち 🍏  
の20個をランチで使ってしまい、6個を購入しました。リンゴは  
何個ありますか？

A：ステップバイステップで考えてみましょう。

1. リンゴの最初の数を 23 個とします。
2. ランチで使った 20 個を引いて、3 個残ります。
3. 6 個を購入しましたので、計 9 個になります。

回答：現在、リンゴは9個あります。

答えを求めるプロンプト


このプロンプトを見ると、few-shot-CoT プロンプトと同様に見えます（ただし、入力と出力の例があるわけではなく、解きたい問題とその解法がプロンプトに含まれるだけです）。

概念的にはこのような2つのプロンプトをモデルに入力するのですが、実際には「ステップバイステップで考えてみましょう」と書くだけで、これらをまとめて行ってくれるというわけです。


手を煩わせる必要がないという点では、zero-shot-CoT プロンプトはかなり優秀ですが、言語モデルが正しく推測してくれないこともあります。そうしたときには few-shot-CoT で言語モデルの振る舞いを細かく制御するというのがよいでしょう（これについては次節で見えます）。

## ChatGPT はどうか

本稿の冒頭でも見たように、ChatGPT は少し複雑な（ステップが複数ある）計算問題についてはうまく具合に答えてくれます。



このカフェテリアには23個のリンゴがあります。そのうちの20個を使ってランチを作りました。その後、6個を買い足しました。リンゴは何個あるでしょう？



最初は23個のリンゴがあり、そのうち20個を使ったので、残りのリンゴは  $23 - 20 = 3$  個です。  
その後6個を買い足したので、合計で  $3 + 6 = 9$  個のリンゴがあります。

少し複雑な計算問題なら ChatGPT は解いてくれる（ことが多い）


今度は、コインの表向き／裏向きをモデルに推測させてみます。

初期条件はコインが表向きに置いてあるものとして、以下のようにコインをひっくり返すとしましょう。かつこ内はコインの向きがどう変化するのかをメモしたものです。


- A 氏は 1 回ひっくり返す（表→裏）
- B 氏は 2 回ひっくり返す（裏→表→裏）
- C 氏はひっくり返さない（裏向きのまま）
- その後、3 人が 1 回ずつひっくり返す（裏→表→裏→表）

というわけで、最終的にコインが表向きとなっているかどうかを ChatGPT が推測できるかを試してみます（正しい答えが得られることもあるでしょう）。





コインが表向きに置かれています。A氏はコインを1回ひっくり返しました。B氏はコインを2回ひっくり返しました。C氏はコインをひっくり返しませんでした。その後、3人がコインを1回ずつひっくり返しました。現在、コインは表向きになっているでしょうか？



現在、コインは「裏」向きになっています。

最初にコインが表向きに置かれていたとして、A氏が1回ひっくり返したため、コインは「裏」向きになります。その後、B氏が2回ひっくり返したため、コインはまた「表」向きになります。そして、C氏がコインをひっくり返さなかったため、コインは「表」向きのままになります。


最後に、3人全員がコインを1回ずつひっくり返した場合、コインは「裏」向きになります。したがって、現在のコインは「裏」向きになっています。

コインの表裏の推測は得意でないもよう


この結果を見ると、B氏がコインを2回ひっくり返すと、コインは裏向きから表向き、そしてまた裏向きになるはずですが、そこを理解できませんでした。そこを間違えたばかりにコインは最終的に裏向きだと言語モデルは推測をしました。

こんなときには zero-shot-CoT プロンプトの出番です。といっても、InstructGPT とは UI が異なっているので、ここでは入力プロンプトの最後に「ステップバイステップで考えてみましょう」と付加することになります。

その結果が以下です。



コインが表向きに置かれています。A氏はコインを1回ひっくり返しました。B氏はコインを2回ひっくり返しました。C氏はコインをひっくり返しませんでした。その後、3人がコインを1回ずつひっくり返しました。現在、コインは表向きになっているでしょうか？  
ステップバイステップで考えてみましょう。



はい、ステップバイステップで考えてみます。


- 最初にコインが表向きに置かれているため、表が上になっています。
- A氏がコインを1回ひっくり返すため、裏が上になります。
- B氏がコインを2回ひっくり返すため、表が上になります。
- C氏がコインをひっくり返さないため、表が上になったままです。
- 最後に、3人全員がコインを1回ずつひっくり返すため、コインが裏返ります。つまり、表が下になり、裏が上になります。

したがって、現在のコインは「裏」向きになっています。

ステップバイステップで間違えた


ここでは先ほどと同じポイント（B氏が2回ひっくり返した）で先ほどと同じ間違いをしたので、結果も間違っただものになってしまいました。

こうなったら **few-shot-CoT** プロンプトを試してみることにしましょう。ただし、ここでは現在の問題を解く手順をそのまま入力プロンプトに組み込むだけとします（他の入力と出力の組はここでは省略します）。



コインが表向きに置かれています。A氏はコインを1回ひっくり返しました。B氏はコインを2回ひっくり返しました。C氏はコインをひっくり返しませんでした。その後、3人がコインを1回ずつひっくり返しました。現在、コインは表向きになっているでしょうか？

- ・ A氏は1回ひっくり返すと表向きのコインが裏向きになります（表→裏）
- ・ B氏は2回ひっくり返すので、裏向きのコインが表向きになった後、それが裏向きになります（裏→表→裏）
- ・ C氏はひっくり返さないで裏向きのままです（裏）
- ・ その後、3人が1回ずつひっくり返すので裏向きのコインが表向きになり（1人目）、表向きのコインが裏向きになり（2人目）、最後に裏向きのコインが表向きになります（3人目）（裏→表→裏→表）



したがって、現在コインは表向きになっています。

**few-shot-CoT** プロンプト

自分でほぼ解答を入力しているので、ここでは正しい答えが出力されました。



言語モデルに推測させるというよりも、言語モデルと一緒に答えを考えている感が非常に強くなります（笑）。

このように ChatGPT は InstructGPT よりも正しい答えを出してくれることが多いのですが、うまくいかないときには **zero-shot-CoT** プロンプトを使って「ステップバイステップで考えてみましょう」として見て、それでもダメなら **few-shot-CoT** プロンプトを試してみるというのは確かによいアプローチのようです。



「一緒に答えを考えている感」はなるほど。思考の連鎖によるプロンプトは、どうしても人間が子供に勉強というか質問への答え方を教えているのに似ているという感じがありますね。そういう意味では ChatGPT は、何でも答えてくれる大先生というよりも、すぐく物知りでクリエイティブな発想ができる子供だと思って接するのがよいのかなと思いました。

あと、前回も同じ感想だったのですが、ますます人間が考え込むような論理的思考が必要となる質問への回答は、現状の ChatGPT にはあまり向いていないなと思いました。答えが間違っている可能性を考慮する必要があるため、結局は答え合わせを質問者がしなければならないので。

というわけで、ChatGPT のプロンプトについてはこのくらいにして、今回は ChatGPT API を使って何かを試してみようと思います。

# ChatGPT の API を使ってみよう： コンソールで対話するコードとは？

2023 年 3 月にリリースされた ChatGPT の API の使い方、API の振る舞い、対話をどのように管理するのか、コンソールで API を介して対話をするコードなどを紹介します。

かわさきしんじ, Deep Insider 編集部 (2023 年 03 月 24 日)

## ChatGPT API

2023 年 3 月 1 日、OpenAI が ChatGPT の API をリリースしました。今回はこの API を実際に使って、その特徴を調べてみることにします。



さらに 2023 年 3 月 14 日には GPT-4 がリリースされましたね。GPT-4 は ChatGPT でも使用できますが、筆者はまだ待機リストに登録された状態なので、今回は gpt-3.5-turbo を言語モデルとして使用します（かわさき）。

[ChatGPT API のドキュメント](#)によれば、この API は一連のメッセージを入力として受け取り、モデルが生成したメッセージを出力とするとのこと。

API 呼び出しには次のようなものがようになります。この連載の[第 2 回](#)でも説明しましたが、それとほぼ同じです（使用するクラスが `openai.Completion` から `openai.ChatCompletion` に変わっているところと、`create` メソッドで単なるプロンプトではなく、メッセージを要素とするリストを渡すところが異なる点です）。もう一度ここで簡単にまとめておきましょう。

- OpenAI へサインアップする
- OpenAI のサイトで API キーを取得する
- `openai` モジュールのインストールする (`pip install openai`)
- `openai.ChatCompletion.create` メソッドを呼び出す

API キーを取得する詳しい手順などは[第 2 回](#)を参照してください。また、`openai` モジュールを使うことで Python からこの API を呼び出せます。

コード的には次のようにとてもシンプルです。

```
import openai

KEY = " 取得した API キー "
openai.api_key = KEY

messages = [
    # ChatGPT との対話内容:
    # {"role": "ロール", "content": メッセージ} という辞書を要素とするリスト
]

completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo", # ChatGPT API を使用するには 'gpt-3.5-turbo' などを指定
    messages=messages
)

print(completion)
```

ChatGPT API を呼び出すコードのひな型

以下では messages リストの内容についてお話をした後に、実際に API を呼び出してみます。

## ChatGPT API を呼び出してみよう

第 2 回で取り上げた InstructGPT では openai.Completion.create メソッドを呼び出すときに prompt 引数にプロンプトを指定していました。第 2 回では次のようなコードを紹介していました。

```
response = openai.Completion.create(
    model="text-davinci-003", # InstructGPT
    prompt=" 晴れた日曜日の午後には何をすればいいかな? ",
    # .....省略.....
)
```

InstructGPT API を呼び出すコードの例

これに対して、OpenAI の[ドキュメント](#)では ChatGPT API を呼び出すコード例としては以下が紹介されています。

```
openai.ChatCompletion.create(  
  model="gpt-3.5-turbo",  
  messages=[  
    {"role": "system", "content": "You are a helpful assistant."},  
    {"role": "user", "content": "Who won the world series in 2020?"},  
    {"role": "assistant", "content": "The Los Angeles Dodgers won the  
World Series in 2020."},  
    {"role": "user", "content": "Where was it played?"}  
  ]  
)
```

#### ChatGPT API を呼び出すコードの例

2 つのコード例の差は、create メソッドの prompt 引数と messages 引数に現れています (model 引数の値も違っていますが)。InstructGPT API の prompt 引数は InstructGPT に対して何をしてほしいのかを指示するプロンプトです。対して、ChatGPT API の messages 引数はいわば人間と ChatGPT との間で行われる対話で送受信されるメッセージをリストに含めたものといえます。上のリストを見れば、リストの要素となっている辞書の "content" キーの値がそれらのメッセージを表していることは分かります。

では、"role" キーの値は何を表しているのでしょうか。その値としては "system" か "role" か "assistant" のいずれかを指定します。"system" は「ChatGPT API と会話を始めるときに、ChatGPT がどんなふうに振る舞うかを指定するときに使う」と考えておきましょう。その後、「人間が ChatGPT にメッセージを投げかけるときには "role" に "user" を指定」します。"assistant" は対話を続けるのに必要な情報を含めるものですが、これについては後で見てください。

何はともあれ、上のメッセージリストを使って API を呼び出してみましょう。以下に示すコードは、API からの戻り値を変数 `completion` に取っておくようにしたり、リストを変数 `messages` に代入したりしていますが、やっていることは上で紹介した API 呼び出しと同じです。

```
import openai

KEY = " 取得した API キーをここに記述 "
openai.api_key = KEY

messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
]

completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages
)

print(completion)
```

**ChatGPT API を呼び出してみる**

`openai.ChatCompletion.create` メソッドを呼び出す際に、`model` 引数に `"gpt-3.5-turbo"` と指定していますが、ChatGPT API を呼び出す際にはこれを指定します。なお、ChatGPT で GPT-4 を使用するのであれば、`"gpt-4"` を指定します。モデルの詳細については OpenAI のドキュメント「[Models](#)」を参照してください。特定の日付が付いたスナップショットもリリースされていますが（`"gpt-3.5-turbo-0301"` や `"gpt-4-0314"` など）、`"gpt-3.5-turbo"` と `"gpt-4"` を指定した場合は API を呼び出した時点で最新の言語モデルが使われます。日付付きはその特定の日付で指定された言語モデルが使われます。

以下はこれを Visual Studio Code（以下、VS Code）で実行した結果です。

```
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
]

completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages
)

print(completion)
```

[2] ✓ 2.9s Python

```
... {
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The 2020 World Series was played in Globe Life Field in Arlington, Texas due to the COVID-19 pande",
        "role": "assistant"
      }
    }
  ],
  "created": 1679398865,
  "id": "chatcmpl-6wUePzWigANngd1ae00rRlaAeWvQn",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 25,
    "prompt_tokens": 56,
    "total_tokens": 81
  }
}
```

呼び出し結果

戻り値の先頭要素（"choices" 要素）の値はリストになっていますが、筆者が試した限りではその要素は辞書（と同様に使えるオブジェクト）が 1 つだけとなっています（将来変更される可能性はあります）。その "message" キーの値（completion["choices"][0]["message"]）に ChatGPT が生成したテキストが含まれています（"content" キーの値）。

生成されたテキストは、上の例では「The 2020 World Series was played in Globe Life Field in Arlington, Texas due to the COVID-19 pandemic」となっています。これは messages リストの最後の要素にある「Where was it played?」（それはどこで開催されましたか？）に対応する返答です。

ここで「ん?」と思いませんか？ messages 引数には幾つかのメッセージが含まれているのに、返送されたテキストは 1 つだけです。「Who won the world series in 2020?」（2020 年のワールドシリーズで優勝したのはどこ?）に対応する返答はありません。



ここから予想されるのは、ChatGPT API は対話の最後に生成したメッセージだけを返信するということです。対話の途中で生成されたテキストは全て捨てられているようです。API がそういう仕様であれば、メッセージを 1 つずつ送信すればよさそうな気がします。次にこれを試してみましょう。

## ChatGPT API は対話内容を記憶しない

`messages` リストには合わせて 4 つのメッセージが含まれていました（そのうちの 1 つは `"role"` が `"assistant"` になっていますが、取りあえず、そこは気にしないことにします）。これを順次 API に渡して、その戻り値を表示するコードを以下に示します。

```
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
]

for msg in messages:
    completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages = [msg]
    )
    print(completion["choices"][0]["message"]["content"])
```

対話の内容を 1 つずつ ChatGPT API に投げるようにしたコード

このコードでは `for` 文を使って `messages` リストに含まれている要素（辞書）を 1 つずつ取り出して、それを `messages` 引数に（リストに詰め込んで）渡すようにしています。それから ChatGPT が生成したテキストだけを画面に表示するようにもしました。残りは先ほどのコードと同じです。

実行結果は次のようになります。

```
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
]

for msg in messages:
    completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages = [msg]
    )
    print(completion["choices"][0]["message"]["content"]) # type: ignore
```

[3] ✓ 29.8s Python

...

Thank you! How may I assist you?

The Los Angeles Dodgers won the World Series in 2020.  
That is correct! The Dodgers defeated the Tampa Bay Rays in six games to win their first World Series championship sir

I'm sorry, I cannot determine what you are referring to. Could you please provide more context or information?

最後の返事に注目

messages リストには 4 つのメッセージがあったので、返答も 4 つあります。最初の返答は「You are a helpful assistant」（あなたはとても有能なアシスタントです）に対するもので「Thank you! How may I assist you?」（ありがとうございます。何かお手伝いしましょうか?）となっています。いい感じです。

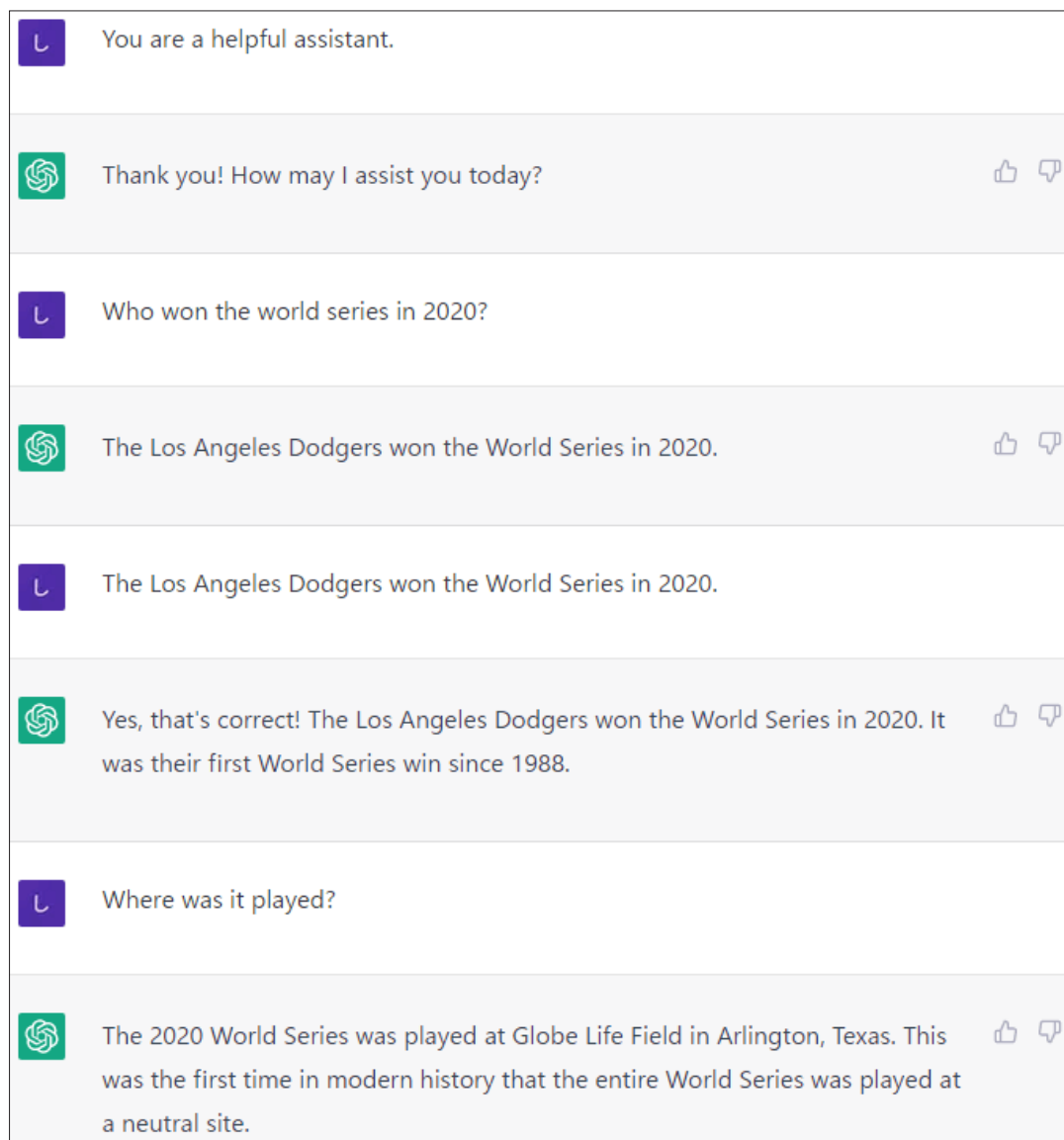
次の返答は「Who won the world series in 2020?」（2020 年のワールドシリーズで優勝したのはどこ?）に対するものです。「The Los Angeles Dodgers won the World Series in 2020」（ドジャースが優勝）という答えになっていますね。

その次は「The Los Angeles Dodgers won the World Series in 2020」（2020 年のワールドシリーズではドジャースが優勝しました）となっているので返答はそれを肯定するものになっています（"role" が "assistant" になっていることについては後で説明します）。

問題は最後の返答です。「Where was it played?」つまり「それはどこで開催されましたか?」というメッセージに対して、「I'm sorry, I cannot determine what you are referring to. Could you please provide more context or information?」（すみません。何のことをいっているのか判断が付きません。もっと情報を与えてくれますか?）と返されています。4 つのメッセージをまとめてリクエストしたときには開催地を答えてくれたのにそうはいきません。

これは ChatGPT が複数回の API 呼び出しの間で、対話に関する情報（記憶、コンテキスト）を維持しないためです。この場合は「2020 年のワールドシリーズでドジャースが優勝した」という情報が最後の API 呼び出しではなくなっているからです。文脈がないところで「それ」「it」といわれても ChatGPT は困ってしまうというわけです。

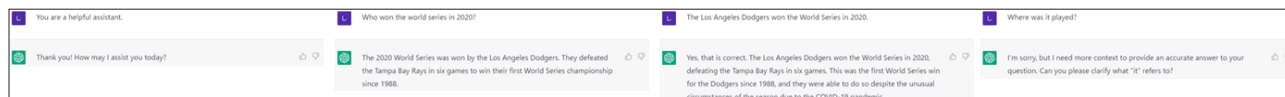
ChatGPT の Web UI を例に考えてみましょう。メッセージをひとまとめにして API に送るというのは、以下のような連続したやりとりを行って、最後の返答だけを返してきたものだと考えられます。



4 往復の対話（イメージ）

このときには、対話を続けていく中で出てきた話題や情報を ChatGPT が覚えており、「it」が何を指しているのか、といったことも文脈に応じて判別できています。

一方、今のようにメッセージをバラバラにして ChatGPT API を呼び出すのは、上のような連続した対話にはならず、一往復の対話を 4 回続けただけということです。以下の画像はその例ですが、画面サイズを縮小しているので文字が読みづらくなっています。が、ここでは文字の内容はどうでもよいです。一往復のリクエストが 4 回というイメージをつかんでください。



一往復の対話を 4 回行う (イメージ)

というわけで、一往復の対話 (API 呼び出し) をしながら、ユーザーの入力ごとに API からの返答を表示するにはどうしたらよいでしょう。



ChatGPT の Web UI では [New chat] で新規のチャットが作られてその対話の流れ全てが履歴に残りますが、対話の流れ全てを覚えている必要があるからなのだと合点がいきました。

コードは少し長くなって面倒な印象でしたが、対話の流れ全てをメッセージとして渡すためには仕方ないですね (一色)。

## messages リストで対話内容を管理する

「ユーザーが入力して返答を受け取り表示する」という一往復の対話を繰り返しながら、さも ChatGPT API と流れるように対話を行うには以下のようなことを行います。カッコ内の「"system" メッセージ」などは "role" キーに指定する値を表しています。"system" / "user" / "assistant" のいずれかになることを先ほどお話しした通りです。

1. messages リストに最初のメッセージを格納する ("system" メッセージ)
2. ユーザーからの入力をリストに追加する ("user" メッセージ)
3. API を呼び出す
4. API から返答をリストに追加する ("assistant" メッセージ)
5. ユーザーからの入力をリストに追加する ("user" メッセージ)
6. API を呼び出す
7. API から返答をリストに追加する ("assistant" メッセージ)
8. 対話が終わるまでこれを繰り返す

このようにしてユーザーが何を入力したか、API が何を返したかを messages リストに全て保管して、API を呼び出すときにそれらをまとめて渡すようにします。

以下のコードを見てください。

```
user_inputs = [
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "user", "content": "Where was it played?"}
]

messages = [
    {"role": "system", "content": "You are a helpful assistant."}
]

for user_input in user_inputs:
    print(f'>>> {user_input["content"]}')
    messages.append(user_input)
    completion = openai.ChatCompletion.create(
        model = 'gpt-3.5-turbo',
        messages=messages
    )
    content = completion["choices"][0]["message"]["content"] # type: ignore
    print(content)
    messages.append({"role": "assistant", "content": f'{content}'})
```

**messages** リストで対話内容を管理する

ここではユーザー入力を模したものを **user\_inputs** リストに含めるようにして、**messages** リストには最初は **"system"** メッセージだけを含めるようにしました。**"assistant"** メッセージがないのは、API からの返答を **messages** リストに追加するためです。

その後は、**user\_inputs** リストの内容を **messages** リストに追加するたびに、API を呼び出して、その返答を表示して、リストに追加しているだけです（最初の **"system"** メッセージの表示は省略しています）。

このコードの実行結果を以下に示します。

```
user_inputs = [
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "user", "content": "Where was it played?"}
]

messages = [
    {"role": "system", "content": "You are a helpful assistant."}
]

for user_input in user_inputs:
    print(f'>>> {user_input["content"]}')
    messages.append(user_input)
    completion = openai.ChatCompletion.create(
        model = 'gpt-3.5-turbo',
        messages=messages
    )
    content = completion["choices"][0]["message"]["content"] # type: ignore
    print(content)
    messages.append({"role": "assistant", "content": f'{content}'})
```

[4] ✓ 11.0s Python

```
... >>> Who won the world series in 2020?
The Los Angeles Dodgers won the World Series in 2020.
>>> Where was it played?
The World Series in 2020 was played in Arlington, Texas.
```

対話の内容が毎回表示されるようになった

"user" メッセージは「>>> 」というプロンプト付きで表示するようにしたので、いかにも人間と ChatGPT が対話しているように見えますね。

この状態で messages リストがどうなっているかを見てみましょう。

```
for msg in messages:
    print(msg)
```

[5] ✓ 0.1s Python

```
... {'role': 'system', 'content': 'You are a helpful assistant.'}
{'role': 'user', 'content': 'Who won the world series in 2020?'}
{'role': 'assistant', 'content': 'The Los Angeles Dodgers won the World Series in 2020.'}
{'role': 'user', 'content': 'Where was it played?'}
{'role': 'assistant', 'content': 'The World Series in 2020 was played in Arlington, Texas.'}
```

messages リストの内容

最後の API 呼び出しの結果も含まれているので 5 行になっていますが、最初の例で見たものと同様な要素が含まれていることが分かります（最初の例の messages リストも似たような方法で作成したものだとは筆者は予想しています）。



上の画像を見ると、2 往復の対話が一度の API 呼び出しで行われているように見えますが、そうではなく API 呼び出しが 2 回行われていることには注意してください。この例では過去のイベントの結果を問い合わせているので、一貫性のある会話となっていますが、1 回目の呼び出しで行われた対話の内容が 2 回目の内容と大きく変化して、対話の内容がかみ合っていないように見える可能性もあります。

## コンソールでチャットする

messages リストで対話内容を管理して、API にそれを渡せばよいことが分かったので、コンソールで ChatGPT と対話するためのコードを紹介します。

```
import openai
KEY = " 取得した API キーをここに記述 "
openai.api_key = KEY

class ChatManager:
    def __init__(self, model, sysmsg=" こんにちは、アシスタントさん! "):
        self.model = model
        self.messages = []
        self.sysmsg = sysmsg
        self.compose_message("system", self.sysmsg)
        self.params = {}
        self.setup_params()

    def compose_message(self, role, content):
        self.messages.append({"role": role, "content": content})

    def setup_params(self, **kwargs):
        self.params["model"] = self.model
        self.params["messages"] = self.messages
        keys = kwargs.keys()
        key_list = ["temperature", "top_p", "n", "stream", "stop", "max_tokens",
                    "presence_penalty", "frequency_penalty", "logit_bias", "user"]
        for key in key_list:
            if key in keys:
```



```

        self.params[key] = kwargs[key]

def call_chat_api(self):
    completion = openai.ChatCompletion.create(**self.params)
    result = completion["choices"][0]["message"] # type: ignore
    return result

def get_response_from_api(self):
    result = self.call_chat_api()
    print(result["content"])
    self.compose_message(result.role, result.content)

def chat_by_api(self, content):
    self.compose_message("user", content)
    print(f">>> {content}")
    self.get_response_from_api()

def chat_on_console(self):
    print("チャットを始めます。終わるときには「exit」と入力してください。")
    content = input(">>> ")
    while content != "exit":
        if content == "show status":
            print(self.messages)
            content = input(">>> ")
            continue
        self.compose_message("user", content)
        self.get_response_from_api()
        content = input(">>> ")

if __name__ == "__main__":
    chatmgr = ChatManager("gpt-3.5-turbo")
    chatmgr.chat_on_console()
    print("exit chat")

```

ChatMnager クラス

詳細なことには触れませんが、ここでは ChatManager クラスに messages リストとそれを管理するコード、ChatGPT API 呼び出しを行うコードなどをまとめてあります。setup\_params メソッドでは create メソッド呼び出しで指定可能なパラメーターを設定できるようにしました（が、本稿では使いませんでした）。

chat\_on\_console メソッドがコンソールで対話するメインコードです。ここではコンソールから入力された値を受け取って、それを基に messages リストに "user" メッセージを追加し、API を呼び出して、その結果を使って messages リストを更新する処理を行っています。このときには、messages リストの内容を表示したり、「exit」と入力すると対話を終了したりするようにしてあります。



VS Code 上の Jupyter ノートブックで対話をできるようなメソッドも含めてあります（Jupyter で試してからコンソールに話を移す予定だったのですが、文字数的にあきらめました）。

実行している様子を以下に示します。



```
test.ipynb chatmgr.py ×
chatmgr.py > ChatManager
4
5 class ChatManager:
6     def __init__(self, model, sysmsg="こんにちは、アシスタントさん！"):
7         self.model = model
8         self.messages = []
9         self.sysmsg = sysmsg
10        self.compose_message("system", self.sysmsg)
11        self.messages.append(self.compose_message("system", self.sysmsg))

問題 出力 ターミナル JUPYTER デバッグコンソール Python + - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 
```

