Quickstart

Looking for ChatGPT? Head to chat.openai.com.

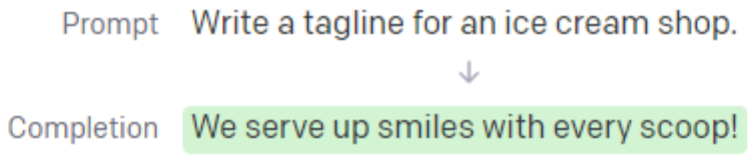
OpenAI has trained cutting-edge language models that are very good at understanding and generating text. Our API provides access to these models and can be used to solve virtually any task that involves processing language.

In this quickstart tutorial, you'll build a simple sample application. Along the way, you'll learn key concepts and techniques that are fundamental to using the API for any task, including:

- Content generation
- Summarization
- Classification, categorization, and sentiment analysis
- Data extraction
- Translation
- Many more!

Introduction

The completions endpoint is the core of our API and provides a simple interface that's extremely flexible and powerful. You input some text as a **prompt**, and the API will return a text **completion** that attempts to match whatever instructions or context you gave it.



You can think of this as a very advanced autocomplete — the model processes your text prompt and tries to predict what's most likely to come next.

1 Start with an instruction

Imagine you want to create a pet name generator. Coming up with names from scratch is hard!

First, you'll need a prompt that makes it clear what you want. Let's start with an instruction. **Submit this prompt** to generate your first completion.

```
Suggest one name for a horse.
```

Not bad! Now, try making your instruction more specific.

```
Suggest one name for a black horse.
```

As you can see, adding a simple adjective to our prompt changes the resulting completion. Designing your prompt is essentially how you "program" the model.

2 Add some examples

Crafting good instructions is important for getting good results, but sometimes they aren't enough. Let's try making your instruction even more complex.

```
Suggest three names for a horse that is a superhero.
```

This completion isn't quite what we want. These names are pretty generic, and it seems like the model didn't pick up on the horse part of our instruction. Let's see if we can get it to come up with some more relevant suggestions.

In many cases, it's helpful to both show *and* tell the model what you want. Adding examples to your prompt can help communicate patterns or nuances. Try submitting this prompt which includes a couple examples.

```
Suggest three names for an animal that is a superhero.

Animal: Cat
Names: Captain Sharpclaw, Agent Fluffball, The Incredible Feline
Animal: Dog
Names: Ruff the Protector, Wonder Canine, Sir Barks-a-Lot
Animal: Horse
Names:
```

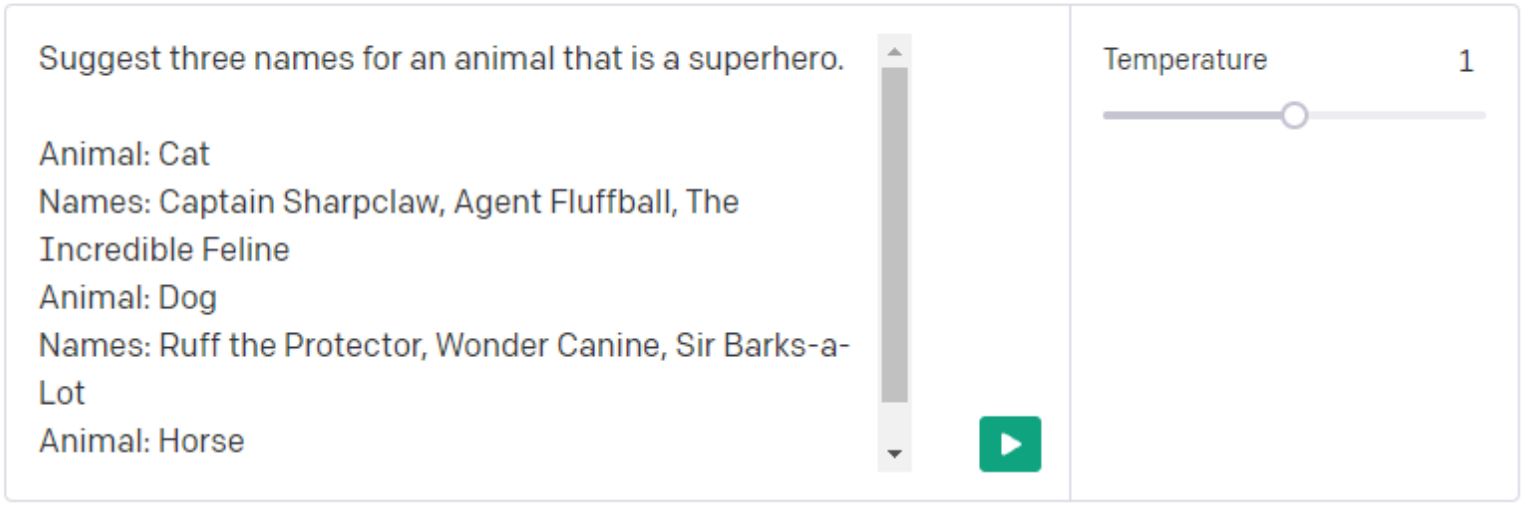
Nice! Adding examples of the output we'd expect for a given input helped the model provide the types of names we were looking for.

3 Adjust your settings

Prompt design isn't the only tool you have at your disposal. You can also control completions by adjusting your settings. One of the most important settings is called **temperature**.

You may have noticed that if you submitted the same prompt multiple times in the examples above, the model would always return identical or very similar completions. This is because your temperature was set to **0**.

Try re-submitting the same prompt a few times with temperature set to **1**.



See what happened? When temperature is above 0, submitting the same prompt results in different completions each time.

Remember that the model predicts which text is most likely to follow the text preceding it. Temperature is a value between 0 and 1 that essentially lets you control how confident the model should be when making these predictions. Lowering temperature means it will take fewer risks, and completions will be more accurate and deterministic. Increasing temperature will result in more diverse completions.

DEEP DIVE

Understanding tokens and probabilities

For your pet name generator, you probably want to be able to generate a lot of name ideas. A moderate temperature of 0.6 should work well.

4 Build your application

NODE.JS PYTHON (FLASK)

Now that you've found a good prompt and settings, you're ready to build your pet name generator! We've written some code to get you started — follow the instructions below to download the code and run the app.

Setup

If you don't have Node.js installed, install it from here. Then download the code by cloning this repository.

```
git clone https://github.com/openai/openai-quickstart-node.git
```

If you prefer not to use git, you can alternatively download the code using this zip file.

Add your API key

To get the app working, you'll need an API key. You can get one by signing up for an account and returning to this page.

Run the app

Run the following commands in the project directory to install the dependencies and run the app.

```
npm install
npm run dev
```

Open http://localhost:3000 in your browser and you should see the pet name generator!

Understand the code

Open up `generate.js` in the `openai-quickstart-node/pages/api` folder. At the bottom, you'll see the function that generates the prompt that we were using above. Since users will be entering the type of animal their pet is, it dynamically swaps out the part of the prompt that specifies the animal.

```
function generatePrompt(animal) {
  const capitalizedAnimal = animal[0].toUpperCase() + animal.slice(1).toLowerCa
  return `Suggest three names for an animal that is a superhero.

Animal: Cat
Names: Captain Sharpclaw, Agent Fluffball, The Incredible Feline
Animal: Dog
Names: Ruff the Protector, Wonder Canine, Sir Barks-a-Lot
Animal: ${capitalizedAnimal}
Names:`;
}
```

On line 9 in `generate.js`, you'll see the code that sends the actual API request. As mentioned above, it uses the completions endpoint with a temperature of 0.6.

```
const completion = await openai.createCompletion({
  model: "text-davinci-003",
  prompt: generatePrompt(req.body.animal),
  temperature: 0.6,
});
```

And that's it! You should now have a full understanding of how your (superhero) pet name generator uses the OpenAI API!

Pricing

We offer a spectrum of models with different capabilities and price points. In this tutorial, we used `text-davinci-003`. We recommend using this model or `gpt-3.5-turbo` while experimenting since they will yield the best results. Once you've got things working, you can see if the other models can produce the same results with lower latency and costs. Or if you might need to move to a more powerful model like `gpt-4`.

The total number of tokens processed in a single request (both prompt and completion) can't exceed the model's maximum context length. For most models, this is 4,096 tokens or about 3,000 words. As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text.

Pricing is pay-as-you-go per 1,000 tokens, with $5 in free credit that can be used during your first 3 months. Learn more.

Closing

These concepts and techniques will go a long way in helping you build your own application. That said, this simple example demonstrates just a sliver of what's possible! The completions endpoint is flexible enough to solve virtually any language processing task, including content generation, summarization, semantic search, topic tagging, sentiment analysis, and so much more.

One limitation to keep in mind is that, for most models, a single API request can only process up to 4,096 tokens between your prompt and completion.

For more advanced tasks, you might find yourself wishing you could provide more examples or context than you can fit in a single prompt. The fine-tuning API is a great option for more advanced tasks like this. Fine-tuning allows you to provide hundreds or even thousands of examples to customize a model for your specific use case.