

# Learning Objectives

---

## Learners will be able to...

- Generate a response using `openai.Completion.create`
- Define NLP and Language Model
- Learn about OpenAI
- Compare GPT-2 to GPT-3
- Run different prompts with OpenAI

info

## Make Sure You Know

You are familiar with Python.

## Limitations

This is a gentle introduction. So there is a little bit of Python programming. Using a “playground” (no code, just natural language) is used to interact with GPT-3.

# NLP

## Definition: Natural Language Processing

Natural Language Processing (NLP) lies at the intersection of linguistics, computer science, and artificial intelligence. Its focus is to give computers the power to read written text and interpret spoken words just as humans can. This is done through the use of computational modeling of human language, which allows real-time analysis of data.

Some common forms of NLP that you may use on a regular basis:

- **Chatbots** - chatbots are used for customer service or even to help you learn a new language.
- **Virtual assistants** - assistants from Amazon, Apple, and Google allow you to interact with computing devices in a natural manner.
- **Online translation** - computers can properly translate text by understanding the larger context through NLP.
- **Spam checkers** - NLP can identify words and phrases that frequently suggest spam or a phishing attempt.

# Language Model

One way to analyze natural language is to use a **language model**.

## What is a Language Model?

A language model is a model which understands language – more precisely how words occur together in natural language. A language model is used to predict what word comes next.

There are a few different types of language models, including **probabilistic language models** and **machine learning language models**. Within each type of language model, there are a number of design decisions in the creation of the model. This includes the mechanics of the model creation (e.g. unigram vs bigram for probabilistic, Neural Network setup for machine learning).

Another design decision for a language model aside from model type is the text it is built from or trained on. Language data can come from a wide range of sources:

- \* chat platforms
- \* text repositories
- \* websites
- \* news articles
- \* books

Ideally, you would create or train your language model on text from the same context it will be deployed in. For example, a model trained on social media sites would be more informal and use different words than a model trained on research articles. For more general purposes, there are general purpose language models.

# Large Language Models

**Large language models** (LLMs) are machine learning algorithms that can recognize, summarize, translate, predict, and generate human languages on the basis of very large text-based datasets.

## Pre-Trained Models

The building and training of models are both complex and resource intensive. Luckily, there are **pre-trained language models**.

A couple of factors to consider when choosing a pre-trained language model:

1. What task are you using it for?
1. What are the technical requirements to use the model?

### What task are you using it for?

The best place to start is to find a purpose-specific model. Pre-trained models often have descriptions which include what the pre-trained models are best for. If you cannot find a model that is specific to your task or your task is ill-defined you can use a more general purpose model.

### What are the technical requirements to use the model?

While some technical requirements are easier to meet such as libraries like PyTorch or TensorFlow, using even a pre-trained model can be resource intensive. In some cases, a minimum RAM is specified or even the use of a GPU, however even meeting the minimum hardware requirements could result in very slow results.

## Popular Pre-Trained Models

There are hundreds of pre-trained language models that can be used. This course will focus on a well-known and very powerful model **GPT-3**.

### OpenAI's GPT-3

GPT-3 is a transformer-based NLP model that performs a range of tasks such as translation, question-answering, and tasks that require reasoning such as unscrambling words.

It is trained on over 175 billion parameters on 45 TB of text from all over the internet, making it one of the biggest pre-trained NLP models available. What differentiates GPT-3 from other language models is it does not require fine-tuning to perform downstream tasks, developers are allowed to reprogram the model using instructions.

# OpenAI

OpenAI is on the cutting edge of AI capabilities. OpenAI's mission is to ensure that artificial general intelligence (AGI)—by which we mean highly autonomous systems that outperform humans at most economically valuable work—benefits all of humanity.

---

**Generative pre-training (GPT)** can acquire knowledge and process long-range dependencies by being trained on a diverse corpus with long stretches of text. Generative in the sense that it can generate text.

## Definition

- **Corpus** refers to one collection of texts.
- **Corpora** refer to multiple collections of texts.

OpenAI released the complete version of the GPT-2(Generative Pre-trained Transformer) as a successor to GPT with 1.5 billion *parameters* in November 2019.

In order to understand the transformer model, we must know a bit about neural networks. A **neural network** refers to a system of neurons working in tandem. Using a neural network we can have a set of connected input/output units where each connection has a weight associated with it.

A **transformer model** is a machine learning method where a sequence of text is processed all at once instead of a word at a time. This allows the connection between words to be more evident.

OpenAI defines **parameters** as the variables that define the behavior of a machine learning model. In other words, parameters are the settings that determine how a model will learn from data and make predictions.

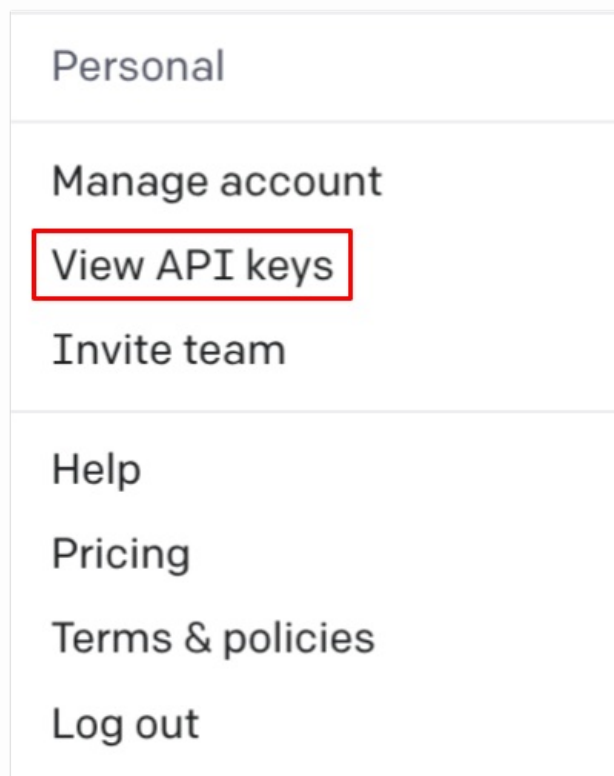
# Setting Up

## OpenAI

Before we can get started using the GPT-3 model, we need to have an account with OpenAI. Use [this link](#) to create an account. OpenAI will ask you a few questions along the way.

Once your account has been created, you need to find your API key. This allows you to connect to the OpenAI API and make use of the GPT-3 model. Click on your profile picture in the top-right corner. Then click on “View API keys”.

)



The image depicts the list of options once you click on your profile picture in OpenAI. There is a red box around the choice “View API keys”.

Next, click the link that reveals your API key if you want to see it. Otherwise, copy the API key for use later. **Important** you should not share your API key with anyone.

# API keys

Your secret API keys are shown in your browser or other client-side application. We automatically rotate any API key that is compromised.

## SECRET KEY

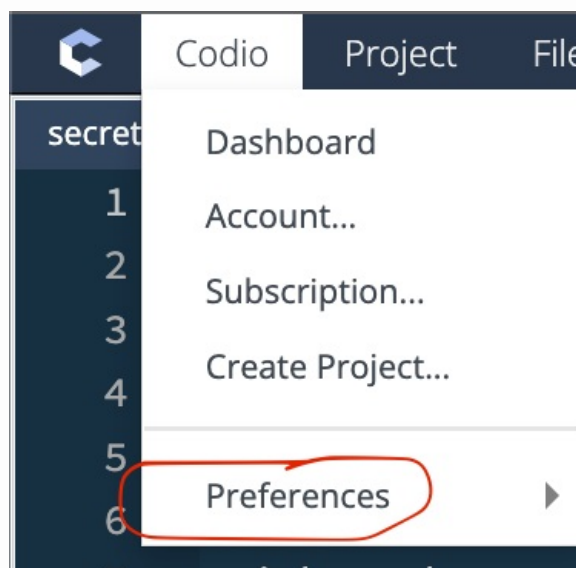
sk-...TtvS [Reveal](#) [Copy](#)

The image depicts a portion of the webpage that has your API key. There is a red box around the link to reveal the API key.

## CODIO

Now that our OpenAI account is set up and we have copied our API key, we are ready to prepare our coding environment. We are going to store our API key as an environment variable. This is going to allow us not to have to copy and paste our variable everywhere. Environment variables are useful when you want to avoid hard-coding access credentials or other variables into code.

The code on the left is going to be how we pull our environmental variables. In order to create our environmental variable we are going to click on the Codio tab on the top left. Inside the Preferences tab, click on the Environment Variables button.

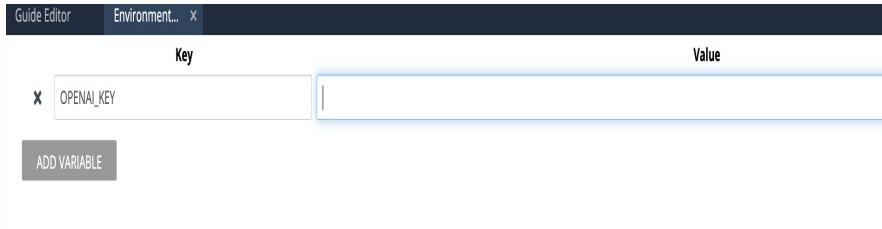


The image depicts a portion of the webpage. It shows the content



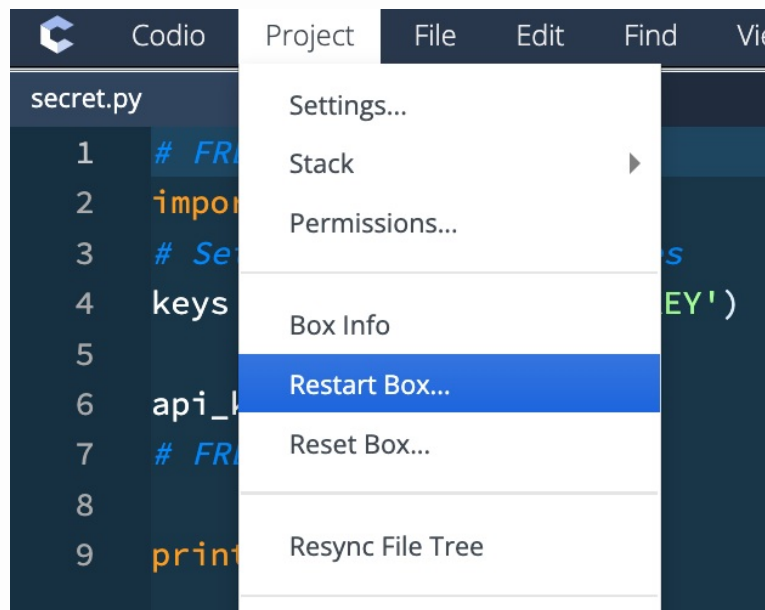
of the Codio tab. There is a red box around the link preferences at the bottom of the Codio tab.

You should get to a page where similar to a dictionary you can create key-value pair. In the key area, name your key `OPENAI_KEY` and paste your OpenAI key in the value section.



The screenshot shows the 'Environment' tab in Codio. It features a table with two columns: 'Key' and 'Value'. The 'Key' column contains the text 'OPENAI\_KEY'. The 'Value' column is empty. Below the table is a button labeled 'ADD VARIABLE'.

From there we can press the **SAVE** button on the bottom left. Then we can restart our Codio box, to make sure the environmental is saved and stored. To restart, press the Project tab next to the Codio drop-down then click on the Restart box.



The image depicts a portion of the webpage. In the drop down "Restart Box" is highlighted.

Note that using `getenv()` or the `get()` method on a dictionary key will return `None` if the key does not exist.

Try to print the key.

```
print(api_key)
```

To execute commands at different checkpoints, a Try It button will be provided. In order to run the print statement from earlier click the try it button below, which will run the code for you and display the response.

If the following prints your key without quotation you are clear. If it is printed with quotations, please go back to the Environment Variable tabs and remove them. If you get None, try checking if you named the API key the right way. Delete the print statement when you are done. We don't want it to keep printing when we use other commands.

## Setting Up 2

### Python

Now that our OpenAI account is set up and we have copied our API key, we are ready to prepare our coding environment. The first thing we need to do is install the `openai` package. In the terminal (bottom-left panel), copy and paste the command below. Press ENTER or RETURN on your keyboard to install the package.

```
python3 -m pip install openai
```

Next, go to the Python file (top-left panel) and import the `os` module and the newly downloaded `openai` package.

```
import os
import openai
```

important

Now we need to put in our environment variable which is stored in our `secret.py` file.

```
import secret
openai.api_key=secret.api_key
```

Create the variable `prompts` which we will use to store the prompt we are going to pass to the GPT-3 model. The variable can be an empty string for now.

```
prompts = ''
```

To query the OpenAI API, we are going to call the `Completion.create` method and store its return value in the variable `response`. We need to specify the model we are using as well as the prompt. This is done with the `model` and `prompt` keyword arguments.

```
response = openai.Completion.create(model="text-davinci-002",  
    prompt=prompts)
```

Finally, we want to print the output from the model. OpenAI returns a complex dictionary. The response we want is buried inside the dictionary. Use the following code to print your results.

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

Because we did not send a prompt to OpenAI, the response is random. Run your code a few more times to see different outputs.

challenge

## **## Try this variation:**

Delete the previous `print` statement. Copy and paste the code below in order to print the entire response from the OpenAI API.

```
print(response)
```

# GPT-3

**GPT-3**(Generative Pre-trained Transformer 3) launched in 2020 is the successor to GPT-2. GPT-3 is trained on over 175 billion *parameters* on 45 TB of text from all over the internet. One of the datasets used for example is *Wikipedia*. The Wikipedia corpus has nearly 1 trillion words altogether.

GPT-3 is the third-generation language prediction model in the GPT-n series. This course will focus on a wide variety of tasks that we can perform with the GPT-3 model.

How **OpenAI's** GPT-3 works by giving an initial text as a prompt, then the program will produce text that continues the prompt.

For example, write a prompt on the text editor on the left, then click the **TRY IT** button below. An example prompt could be how are you.

If you want to try other prompts, you can use the **Reset** button below to clear the text inside the file on the left. Then use the **TRY IT** button to generate another response.

The quality of the text generated by GPT-3 is very high. In other words, its ability to mimic human speech would make it hard to think of the response generated as simply a trained model.

# Models

To query the OpenAI API, we call the `Completion.create` function and store its return value in the variable `response`.

```
response = openai.Completion.create(model="text-davinci-002",
prompt=prompts)
```

We can see the `Completion.create` function takes a `model` keyword argument and a `prompt` keyword argument. The `prompt` keyword argument is the variable which we use to store the prompt we are going to pass to the GPT-3 model.

Now let's tackle, the `model` keyword argument. When OpenAI defines GPT-3, it defines it as a set of models that can understand and generate natural language. GPT-3 offers four main models suitable for different tasks. For example, the `ada` model is the fastest, whereas the `davinci` model we used earlier is the most capable.

Model	Description	Max Request	Training Data
<b>text-davinci-002</b>	Most capable GPT-3 model. Can do the task of the other models, often with less context. Contains the most recent training data. Additionally, this model is capable of inserting completions within a text.	4000 tokens	up to June 2021
<b>text-curie-001</b>	Faster and lower cost than the <code>davinci</code> model, but it is less capable.	2048 tokens	Up to Oct 2019
<b>text-babbage-001</b>	Works best with straightforward tasks only. It is fast and low cost when compared to the previous two models.	2048 tokens	Up to Oct 2019
<b>text-ada-001</b>	Lowest cost and fastest model. You get the best results with <code>ada</code> when the tasks are straightforward.	2048 tokens	Up to Oct 2019

OpenAI plans to continuously improve their models over time. `davinci` is generally the most capable model. `curie` can perform many of the same tasks as `davinci` but faster with a reduced cost. As we continue to experiment with GPT-3, we will use the `davinci` model since it yields the best results. Once things are working, we can try different models to see how the results differ.

Add the following code to the IDE. Make sure to delete the previous prompt,response and print statements. We should only have our libraries and our key left. We are going to have different variables representing the responses from each of the models in the table above. The code then prints each response so we can compare and contrast how each model responds to the same prompt.

```
prompts = 'tagline for ice cream shop'
response1 = openai.Completion.create(model="text-davinci-002",
    prompt=prompts)
response2 = openai.Completion.create(model="text-curie-001",
    prompt=prompts)
response3 = openai.Completion.create(model="text-babbage-001",
    prompt=prompts)
response4 = openai.Completion.create(model="text-ada-001",
    prompt=prompts)

print('davinci-002')
print('-----')
print(response1['choices'][0]['text'].strip(), '\n')

print('curie-001')
print('-----')
print(response2['choices'][0]['text'].strip(), '\n')

print('babbage-001')
print('-----')
print(response3['choices'][0]['text'].strip(), '\n')

print('ada-001')
print('-----')
print(response4['choices'][0]['text'].strip(), '\n')
```

Whereas responses between curie and davinci are pretty close, we can see a clear difference between the ada model and the davinci model. Use this [tool](#) to generate more comparisons if you want to experiment further.

# Testing GPT-3

**GPT-3** is a transformer-based NLP model that performs a range of tasks such as translation, question-answering, and tasks that require reasoning.

Now let us try using GPT-3 in order to try out different prompts with our Python code. We are going to put our different prompts inside the string prompts.

```
prompts = ""
## FREEZE CODE BEGIN
# This code queries the OpenAI API using the
# provided prompt and prints the result
response = openai.Completion.create(model="text-davinci-002",
                                     prompt=prompts)

print(response['choices'][0]['text'].strip())
## FREEZE CODE END
```

Let's first look at how GPT-3 handles prompts in which the model has to translate a word or words. Copy and paste the new value of prompts into the IDE and click the TRY IT button to see the result.

challenge

## Try these variations:

- Change the value of prompts to:

```
prompts = 'Translate "my name is Jeff" in french'
```

- Change the value of prompts to:

```
prompts = 'how do you say car in lebanese?'
```

- Change the value of prompts to:

```
prompts = 'jueves in spanish means'
```



Now, we are going to look at how GPT-3 answers questions. Notice how the sentences do not have to be grammatically correct in order for the model to understand them. Copy and paste the new value of `prompts` into the IDE and click the TRY IT button to see the result.

challenge

### Try these variations:

- Change the value of `prompts` to:

```
prompts = 'what is the capital of canada ?'
```

- Change the value of `prompts` to:

```
prompts = 'closest planet to the sun?'
```

- Change the value of `prompts` to:

```
prompts = 'who is the president of the us'
```

On the one hand, the above questions are not that difficult. You could argue that all the model has to do is “look up” the correct answer. Let’s see how GPT-3 handles questions that require a bit of reasoning. Copy and paste the new value of `prompts` into the IDE and click the TRY IT button to see the result.

challenge

## Try these variations:

- Change the value of prompts to:

```
prompts = 'Correct this to standard English: She no went to  
the movies'
```

- Change the value of prompts to:

```
prompts = 'Create an analogy for this phrase: life is like a  
highway'
```

- Change the value of prompts to:

```
prompts = 'Write a tagline for an ice cream shop'
```

# Coding Exercise

Coding Exercise Requirement :

- \* Use the lowest in cost model
- \* Print only the string response