Can artificial intelligence make software development more productive?

The idea that software can be developed by artificial intelligence without requiring a human developer opens a world of possibilities — and questions. Software development AI applications are targeted mainly at developers, promising to act as 'co-pilots', and making them more productive. Could this be taken even further to the point where developers are not required at all? What benefit could it have for business users? Having recently been granted preview access to the OpenAI Codex application, ***Ravi Sawhney*** *took it on a tour through the lens of a business user.*

In May 2020 OpenAI, an artificial research laboratory, released a new type of AI model called GPT-3. This large language model was trained on a corpus of hundreds of billions of words, with the goal of predicting what text comes next given a prompt by the user. The model quickly gained media attention for its ability to be applied to a wide variety of language tasks with minimal prompt required from the user, known as 'few-shot learning'. For example, it was demonstrated that the model could translate from English to French with a good level of efficacy through the user providing a few examples beforehand. It also performed well in text summarisation, classification, and question-answering tasks.

Moving on from the initial buzz, which was coupled with growing concerns around AI use in decision-making, GPT-3 went quiet as it remained in private beta and it wasn't clear if this model was ready to be incorporated into software production and what its use cases might be beyond general entertainment.

Commercialisation

However, it seems that events are accelerating and Microsoft has begun commercialising this technology, which is not too surprising given the company's significant investment into OpenAI. Microsoft subtly incorporated GPT-3 into its low-code application, Power Apps, by allowing users to type in natural language what their intention is and the application will then return the appropriate syntax.

More significant, though, was the preview release by Github (a Microsoft owned company) of their Co-Pilot product. This application, targeted mainly at software developers, promises to act as a 'co-pilot' by suggesting code to the developer based on comments they write.

Co-Pilot was developed using, as OpenAI calls it, a descendent model of GPT-3 referred to as Codex. Codex was trained on billions of lines of source code from publicly available sources including, of course, Github.

The wider promise

Having recently been granted preview access to OpenAI Codex, I took it on a tour through the lens of a business user.

My goal was to understand if this technology can be practically used to make software developers more productive. Could it be taken even further to the point where developers are not required at all? What benefit could it have to business users? How capable is it in understanding human intent? Which, really, is the ultimate promise of this technology.

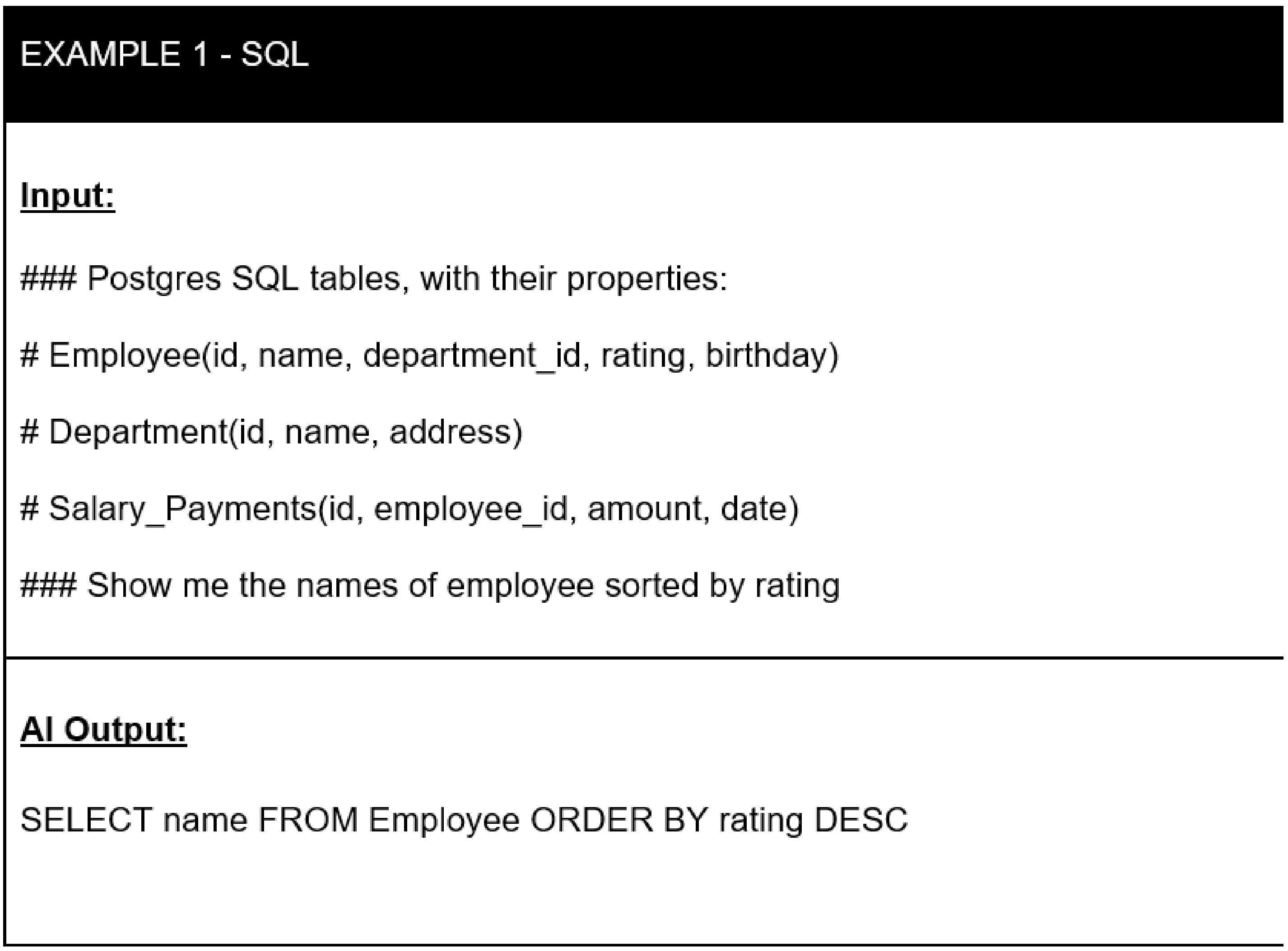
Before diving into real-world examples of Codex, it's worth understanding the potential significance of what this technology is proposing. The terms no- or low-code have only recently entered into our vocabulary. The idea is that software applications can be developed without requiring a software developer, or, put another way, the actual end user can convert their intent to software with zero or minimal understanding of coding. If you consider that software applications exist in nearly every aspect of our personal and professional lives, this capability provides a radically new method to create applications beyond hiring a full-time engineer or buying an off-the-shelf application.

Getting hands-on

Codex works by the user providing some prompts. Then, it takes those prompts, with certain user-controlled parameters, to predict what it thinks the user wants next. In simplistic terms it can be thought of as a turbocharged autocomplete. In principle, it's the same as GPT-3, however the model has been trained on examples of code. There are two models that were released here: 'davinci' and 'cushman'. The latter is designed to be a relation of davinci but faster, at the expense of predicted accuracy. For this demonstration I stuck with davinci, as speed was not a concern but it is interesting to know that OpenAI are thinking about the performance/speed trade-off for real-world applications where low latency is a must.

To demonstrate what it can do I present a series of input prompts ('Inputs') and document the response provided ('AI Output').

I start with examples that convert natural language to the widely adopted structured query language (SQL), which begin simple, then get more complicated and, as you will see, do not always work. I also demonstrate the ability of Codex to convert English to Python.



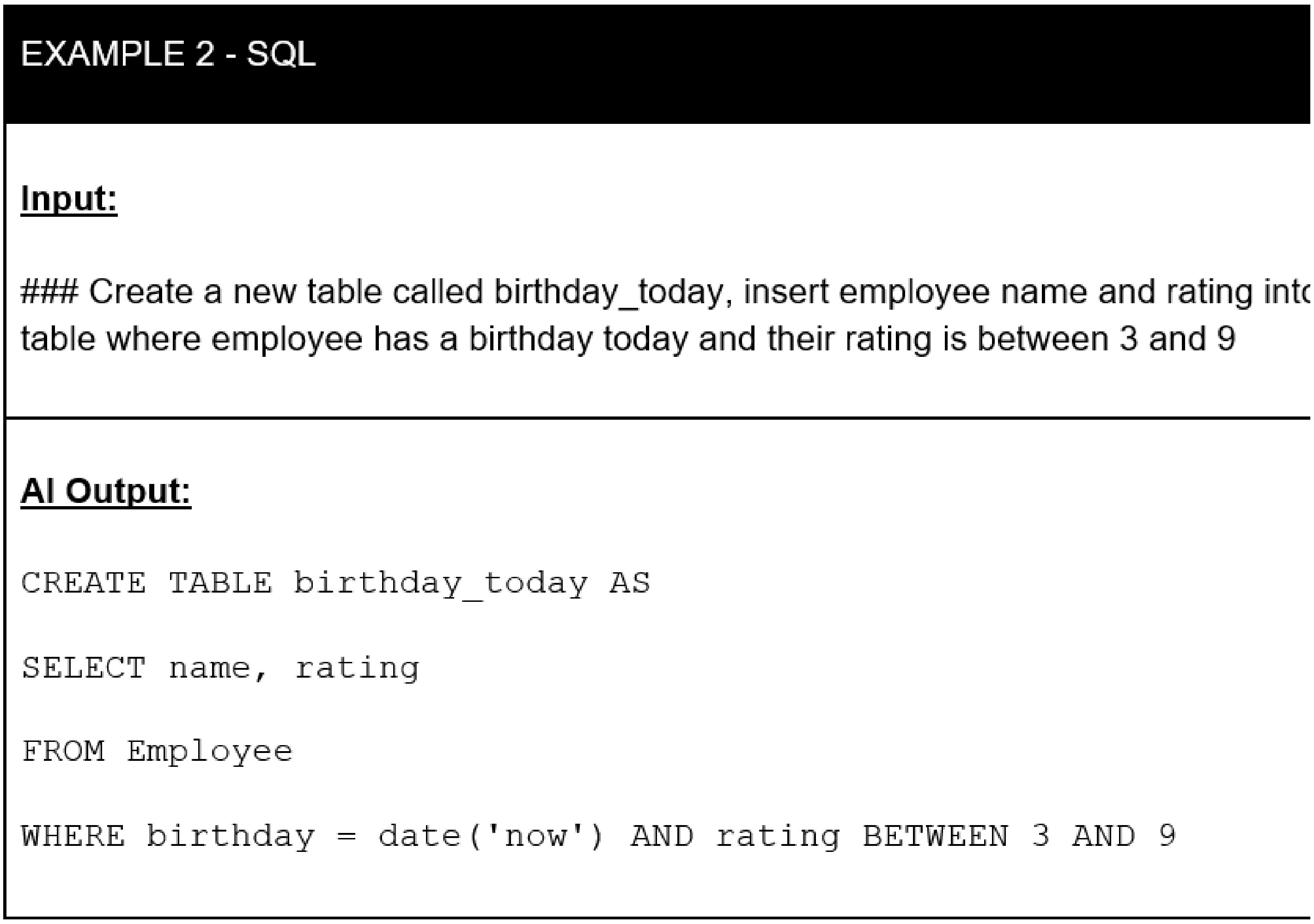
EXAMPLE 1 - SQL

Input:

```
### Postgres SQL tables, with their properties:

# Employee(id, name, department_id, rating, birthday)

# Department(id, name, address)

# Salary_Payments(id, employee_id, amount, date)

### Show me the names of employee sorted by rating
```

AI Output:

```
SELECT name FROM Employee ORDER BY rating DESC
```



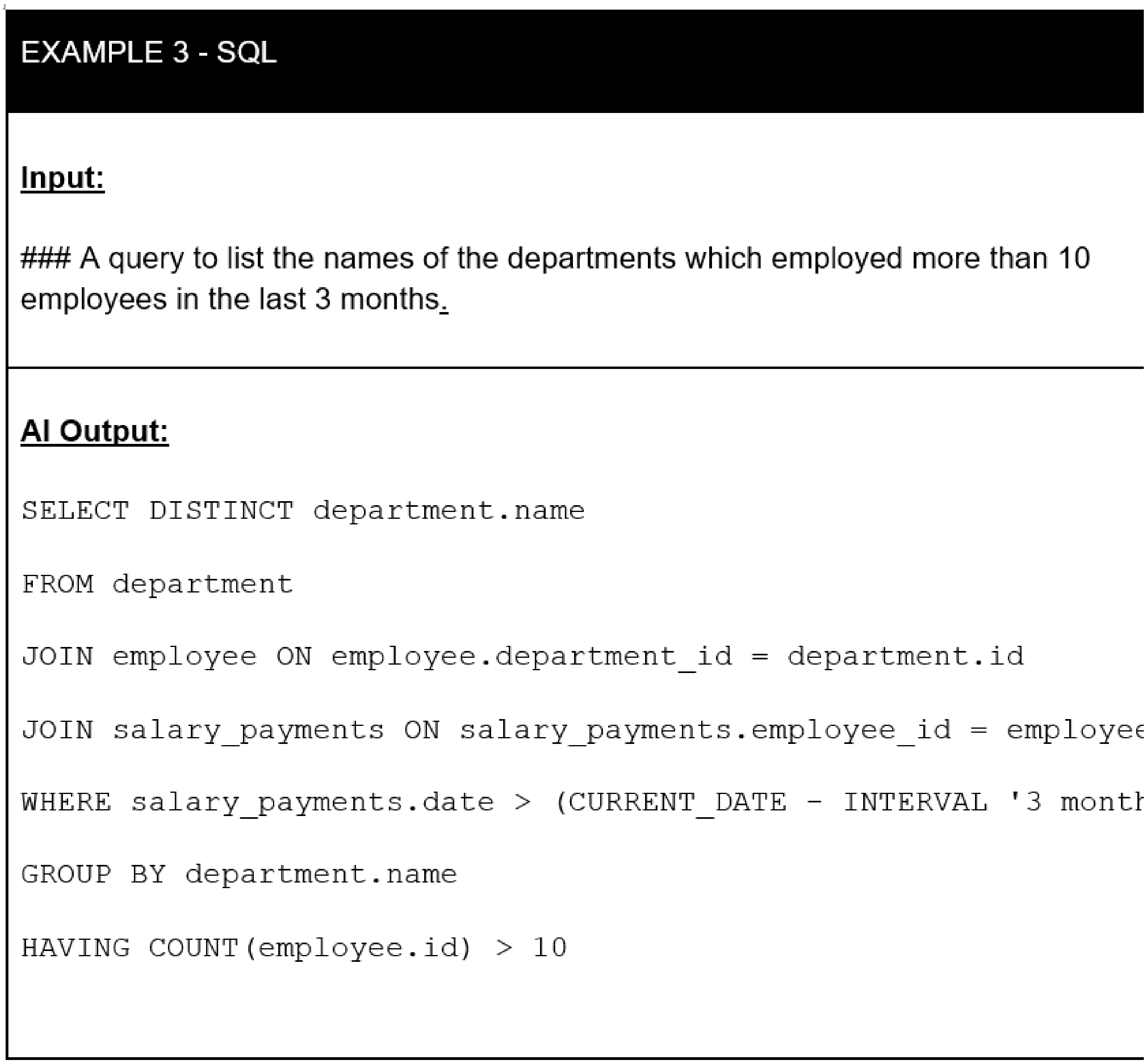
EXAMPLE 2 - SQL

Input:

Create a new table called birthday_today, insert employee name and rating intc
table where employee has a birthday today and their rating is between 3 and 9

AI Output:

```
CREATE TABLE birthday_today AS

SELECT name, rating

FROM Employee

WHERE birthday = date('now') AND rating BETWEEN 3 AND 9
```



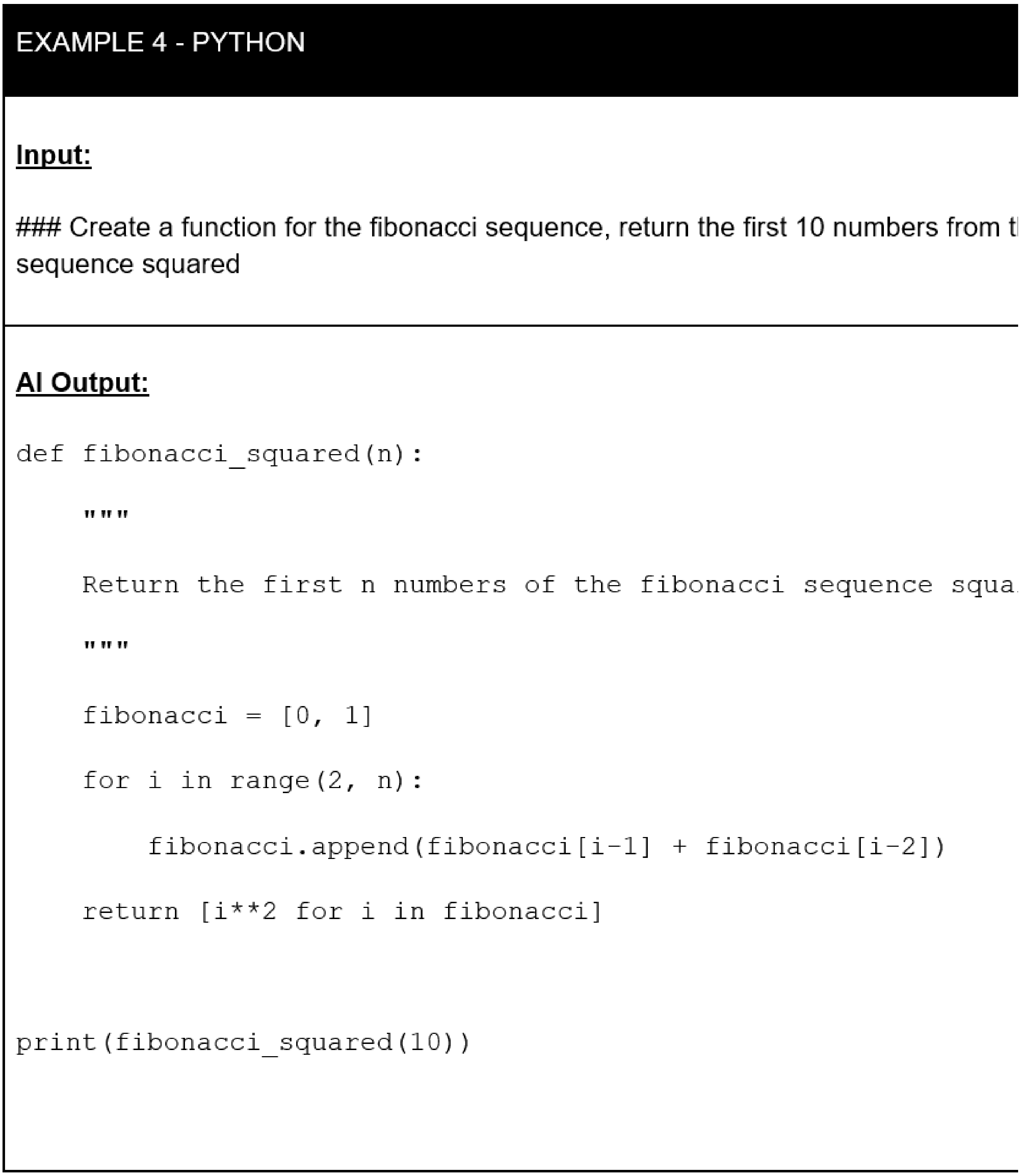
EXAMPLE 3 - SQL

Input:

A query to list the names of the departments which employed more than 10 employees in the last 3 months.

AI Output:

```
SELECT DISTINCT department.name

FROM department

JOIN employee ON employee.department_id = department.id

JOIN salary_payments ON salary_payments.employee_id = employe

WHERE salary_payments.date > (CURRENT_DATE - INTERVAL '3 mont

GROUP BY department.name

HAVING COUNT(employee.id) > 10
```



EXAMPLE 4 - PYTHON

Input:

Create a function for the fibonacci sequence, return the first 10 numbers from t
sequence squared

AI Output:

```
def fibonacci_squared(n):

    """

    Return the first n numbers of the fibonacci sequence squa

    """

    fibonacci = [0, 1]

    for i in range(2, n):

        fibonacci.append(fibonacci[i-1] + fibonacci[i-2])

    return [i**2 for i in fibonacci]


print(fibonacci_squared(10))
```

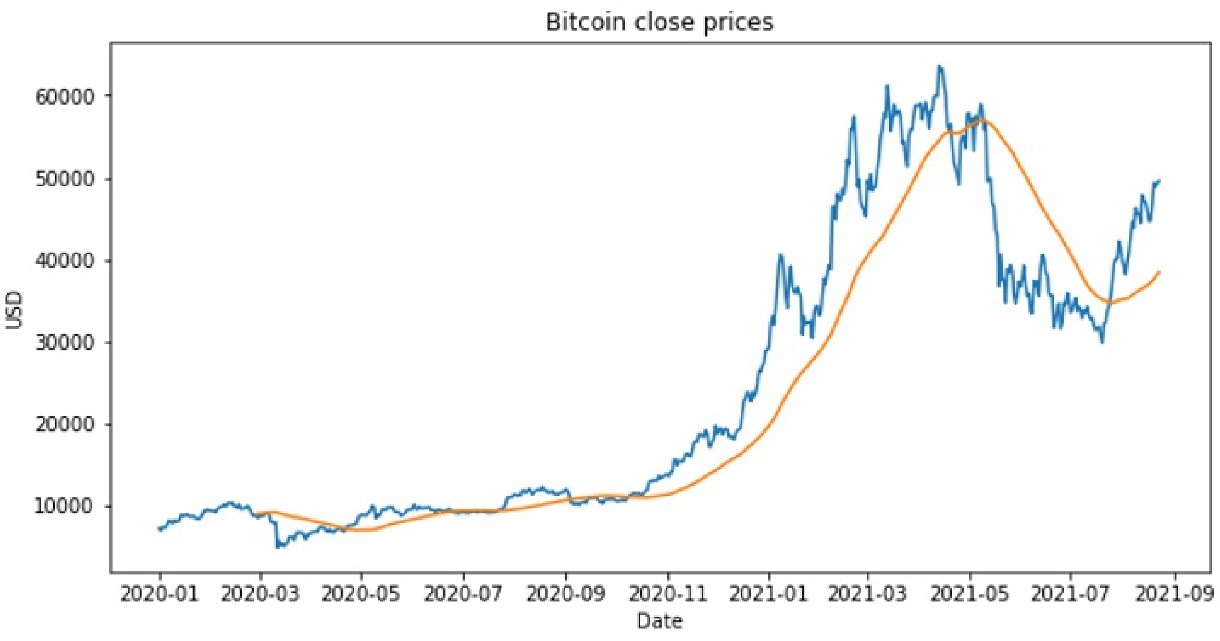
EXAMPLE 5 - PYTHON

Input:

Get Bitcoin close prices from coindesk for 2021, plot those prices in USD and show the 25 day moving average.

AI Output:

```
import requests

import pandas as pd

from datetime import datetime, timedelta

import matplotlib.pyplot as plt

# Get the data from coindesk API

url = 'https://api.coindesk.com/v1/bpi/historical/close.json?start=2020-01-01&end=' +
str(datetime.now())[:10]

r = requests.get(url)

data = r.json()

df = pd.DataFrame(data['bpi'], index=[0]).T

df.columns = ['Close']

# Compute 25 days moving average

df['MA25'] = df['Close'].rolling(25).mean()

# Plot the prices and the moving average

plt.figure(figsize=(12,8))

plt.title('Bitcoin close prices')

plt.xlabel('Date')

plt.ylabel('Price in USD')

plt.plot(df['Close'])

plt.plot(df['MA25'])

plt.legend(['Close', 'MA25'])

plt.show()
```



Initial thoughts

It is hard not to be impressed by OpenAI Codex. Simply writing what you want and having the code being produced in seconds is the stuff of dreams for product managers. The Python example illustrates that Codex knew how to call the CoinDesk API to get the price of bitcoin, although it did not capture the intent exactly right, as it started the plot from the beginning of 2020 and not 2021. These small errors did occur with more complicated examples, but in many of them it only took a few minor edits to fix.

It was also perhaps no surprise that SQL produced the best examples given how close the syntax is to English natural language. In fact, as I was experimenting, it became apparent how useful the technology could be from an educational point of view for someone who is learning to code from scratch. Instead of using Google, the student can ask the AI to help and, more likely than not, it will return something useful that will move their thinking forward.

It is only right for me to add that those examples above were taken after spending some time learning how best to frame the input prompt. In the same way that if you express your business requirements poorly to your human engineer you are likely to end up with a poor product, a vague prompt to Codex will result in non-executable output or one that doesn’t match your intentions.

There are a couple of interesting points to note about Codex, which help provide direction to its wider application in enterprise. Firstly, it is worth stressing that it is informed by existing code. This can result in it quoting back verbatim without attribution to the original developer. Whilst very rare, the fact it can happen might create a headache in trying to understand the legal ramification of how this code might then be used.

Secondly, the model itself is non-deterministic. Whilst the level of creativity can be controlled through exposed parameters, it is impossible to guarantee reproducibility of the output from the model given the same input. Whilst this may seem problematic, especially for the production of code, I noticed that in some cases increasing the creativity of the model resulted in it producing the desired results from poorly defined inputs which was impressive.

What does this mean for the future of software development?

Whilst the examples above demonstrate that Codex can generate executable code to match the users’ intent, I do not see it replacing developers anytime soon. Deploying AI generated code to production enterprise systems without at least a code review is just too risky for the time being.

The more pertinent question for today is: Can Codex assist software engineers in making them more productive? As someone who works on the business side of software development, I find it hard to make a definitive call on this. From a quick straw poll of engineers within my network, the takeaway was that AI definitely has potential to improve developer efficiency if used appropriately.

Many enterprise codebases are vast and complicated in nature and it would be hard to see how Codex could provide high quality and safe suggestions to developers who work on them when it has been trained on unvetted public repositories such as Github. However, if OpenAI permitted Codex to train on private codebases, something it does on GPT-3 through a process called fine-tuning, this could be a game changer. Engineering teams would have certainty as to the quality of training data and it would make the model highly relevant to the firm's existing applications. This could reduce the time it takes to get a new engineer to get productive when learning a new codebase.

Final thoughts

Codex was only released a few weeks ago in private beta and still under active development. Yet, I am truly impressed as it provides a real glimpse of how software might be developed in the very near future. From reducing the barrier to entry for beginner programmers, making expert ones more productive and accelerating the low-code movement that is currently capturing the imagination of many business executives. The economic value of AI in the software development industries cannot be underestimated and warrants continued research.

Authors' disclaimer: *All views expressed are my own.*

♣♣♣

Notes:

- *The post represents the views of the author, not the position of LSE Business Review or the London School of Economics.*
- *Featured image by Markus Spiske on Unsplash*
- *When you leave a comment, you're agreeing to our Comment Policy.*