An experiment of having ChatGPT write my Python code.

Finally, I gave in! I wanted to check the basic Python writing capabilities of ChatGPT. I decided to use a simple command-line application to keep things simple.

I wanted to write a tool that manages text replacement in a group of files (like a subset of the SED command in Linux)

In three small steps, I ran into potential problems. You might argue that the problem is due to unclear specifications. I will let you be the judge!!!

[To keep this article small, I decided to only include the problem code snippet. For detailed output, you can refer to my GitHub ‘Greplace’ repository (https://github.com/vandevelde/Greplace/).]

## I started as follows with the first prompt to ChatGPT:

|  |
| --- |
| write a Python program that takes 3 arguments: directory, search-text, and replace-text. The program should inspect all files in the directory, and replace all occurrences of search-text into replace-text |

ChatGPT gave me back a beautiful piece of software that worked flawlessly! Exactly what I wanted!

|  |
| --- |
| Certainly! Below is a simple Python program that takes three arguments - directory, search-text, and replace-text. It iterates through all the files in the specified directory, reads each file, replaces all occurrences of the search-text with the replace-text, and then writes the modified content back to the file.  python replace\_text.py /path/to/directory "search\_text" "replace\_text"  Replace **/path/to/directory**, **"search\_text"**, and **"replace\_text"** with your specific values.  Note that this script performs a simple text replacement and may not handle complex cases (e.g., multiline replacements or binary files). Adjustments may be needed based on the specific requirements of your use case. |

## I then asked it to modify the code as follows:

|  |
| --- |
| change the directory argument into an argument that can hold a directory path and filename (that includes wildcards) |

Still wonderful. ChatGPT cranked out beautiful code with an extra comment about the use of ‘glob’.

|  |
| --- |
| Certainly! You can use the **glob** module to handle wildcard characters in the filename.  python replace\_text.py /path/to/files/\*.txt "search\_text" "replace\_text"  This will replace occurrences of "search\_text" with "replace\_text" in all **.txt** files in the specified directory. Adjust the file pattern according to your needs.  Here's the modified script: |

## I then asked it to add an optional command-line parameter.

|  |
| --- |
| add an optional first argument "-s". if that is specified, the program should also inspect all subdirectories |

ChatGPT gave me the extra code with an extra comment about the use of os.walk.

|  |
| --- |
| Certainly! You can use the **os.walk** function to recursively iterate through all files in subdirectories. Here's the modified script with an optional **-s** flag: |

And here is where a problem occurred. As you can see in the code below, if include\_subdirectories is True, wildcard file naming is ignored and the program reverts to scan all files in the subdirectory instead of filtering out files that comply with the wildcards.

|  |
| --- |
| # Determine whether to include subdirectories      if include\_subdirectories:          files = [os.path.join(root, file) for root, \_, files in os.walk(os.path.dirname(file\_pattern)) for file in files]      else:          files = glob.glob(file\_pattern)      # Iterate through all files      for file\_path in files:    # etc |

As a result, if the user requests to scan sub-directories, the software ignores the wildcards and considers all the files in the folder(s).

**From my point of view, it created code with a bug!!**

Now, you could argue that I did not specifically request that, and you could be right 😊

## I asked it specifically to take care of that condition.

|  |
| --- |
| when scanning through the subdirectories, also maintain the file-pattern |

ChatGPT updated the code and gracefully acknowledged the new/changed requirements.

|  |
| --- |
| Certainly! I've modified the script to include the **file-pattern** when scanning through subdirectories. The **file-pattern** will be used to filter files in both the current directory and its subdirectories. |

The updated code for your reference, is as follows:

|  |
| --- |
| # Determine whether to include subdirectories      if include\_subdirectories:          files = [file for file in glob.glob(file\_pattern)]          for root, \_, subfiles in os.walk(os.path.dirname(file\_pattern)):              for subfile in subfiles:                  if glob.fnmatch.fnmatch(os.path.join(root, subfile), file\_pattern):                      files.append(os.path.join(root, subfile))      else:          files = glob.glob(file\_pattern)      # Iterate through all files      for file\_path in files:  # etc |

All is well that ends well.

## Conclusion based on this quite simple test.

ChatGPT is currently one of our most advanced and popular AI tools. It writes flawless code, that is clear, easy to read, and efficiently utilizes built-in features and libraries. However, during my test, it was not able to be intuitive enough in step 3 to anticipate what I wanted to accomplish.

Based on this limited experiment, my conclusion is that, at the current state of AI, I would be very hesitant to let AI generate the code for anything else but simple tools. I think AI code generation could be used as a code-buddy, to review the code that an engineer writes, but I would not advocate doing the opposite and let AI do a first draft.