



## PROMPTS FOR PROGRAMMING IN CHATGPT

## Prompt Template

When building a prompt for ChatGPT to generate code, make sure you consider including the following elements:

- What specialized knowledge currently exists? Is there existing code to start from? Do you already have the steps needed for a particular calculation or procedure?
- What are the inputs to the program? If the user provides inputs, what are they and what format will they be in? If there are data files, what formats will they be in? If the program is to retrieve information from the internet, what site(s) should it use?
- What should the program do? How should it handle errors that occur due to bad inputs or faulty code?
- What should the output look like? Should it be graphical, text-based, or a combination? If there are files to be output, what format should they be in?
- What constraints should the program follow? Should it have a graphical user interface (GUI)? Should it be accessible as a web-based application or a desktop app? What programming language should it be in?
- What other instructions should ChatGPT follow? Should it explain the code it generates? Should it provide alternative approaches?

Write a Python program to convert pounds to long tons. There are 2,240 pounds per long ton. The user will enter pounds into a graphical user interface, and long tons should appear to two decimal places of precision. Explain the code you write.

Some prompts might not require all of the elements, and there's no requirement to put the elements in any particular order.

## General Prompting Strategies

**Specific and detailed prompts** are more likely to deliver correct code, while **general and open prompts** generate capabilities and features you might not have thought of.

Use ChatGPT's **Custom Instructions** to give it information that applies to all of your prompts (e.g., "Write all code in Python," "Comment all code," etc.)

Use **delimiters** to help ChatGPT recognize separate elements of your prompt. For example, if you include code in a prompt, put a line of three asterisks before and after it, and tell ChatGPT that's what you've done.

**Reference existing apps** that do something similar to what you want to help ChatGPT understand the task.

**Provide sample inputs** to help ChatGPT understand their format and content.

Use **programming terminology** to get ChatGPT to write code using specific constructs. For example, if you want your code to be in a reusable function, tell ChatGPT to "write a function that..." instead of "write a program that..."

**Describe the user** to help ChatGPT better understand the problem. For example, "Write a program for elementary school students that tests them on simple addition problems" helps ChatGPT build a user interface suitable for children.

Specify how the code should **handle errors**. This includes erroneous inputs, unavailable resources (e.g., a file is missing or the network is down), or runtime errors. For example, while you're in the debugging process you might consider including "Let all unanticipated runtime errors generate a traceback."

Custom instructions only apply to new conversations. Your existing conversations will not change when you create or change custom instructions.

## Strategies for Complex Tasks

Use the **least to most strategy**. Start with the basic core of the application, then once it's working prompt ChatGPT to modify the code to add a new feature or capability. Get that working, then add the next feature or capability. Repeat this process until you have the app you want. This strategy gives ChatGPT smaller problems to work on, which helps it succeed.

Use **question refinement** to understand what ChatGPT needs in a prompt. This helps you understand what ChatGPT considers critical information for the task at hand.

Use the **generated knowledge strategy**. Prompt ChatGPT to explain its approach to the problem before generating any code. Review that approach, and prompt ChatGPT to make changes where you see flaws in its logic. Then prompt ChatGPT to follow its approach, either all at once or in small steps as suggested by the least to most strategy.

**Give ChatGPT the steps** it needs to follow to complete the task. This helps ChatGPT by eliminating one potential source of error (i.e., a flawed approach) and gives you more control over the operation of the resulting program.

ChatGPT maintains a running log of prompts and responses in a conversation, stored as tokens that represent parts of words. If that log becomes too long, ChatGPT can get distracted by old information and generate errors. In long conversations, you may need to periodically **reset the context** by summarizing where you are and including the most current version of the code.

Some of ChatGPT's underlying models can handle over 1,000,000 tokens (on average, every two words of conversation take up three tokens). That said, even conversations with much fewer tokens can cause problems.

## Strategies for Debugging

**Regenerate the prompt** to allow ChatGPT's randomness injection to work in your favor by getting a different and hopefully correct response. Use this strategy for simple prompts only. If ChatGPT has already generated three or more incorrect responses you'll need to take another approach.

**Paste the traceback** you get from Python in the event of a runtime error. Since the traceback provides the specific error and where in the code it occurred, ChatGPT has lots of information to put the error in context. This strategy should always be your first choice when you have a traceback.

**Explain the problem** your code is having in plain language. Include the inputs that are causing the problem and what the expected performance is. If you can anticipate ChatGPT giving you an incorrect "fix" to the solution (e.g. "you should change the format of your input"), tell ChatGPT that's not a valid solution. Use this strategy early in the debugging process and with logic errors (runtime errors with tracebacks are better attacked with the "paste the traceback" strategy).

Prompt ChatGPT to **explain the approach** it's taking to the problem. This allows you to identify any fundamental flaws in ChatGPT's logic. If you can identify where ChatGPT is going wrong, prompt it to change that part of its approach. Alternatively, you can ask ChatGPT to explain a specific step that's not succeeding. Use this strategy when it becomes clear that earlier strategies are not getting you closer to correct code.

Prompt ChatGPT to **change the approach** it's taking when it's clear that other debugging techniques are not getting it closer to correct code and you can't identify what part of the approach is flawed. Be sure to include in your prompt that the current approach isn't working and to try a different approach. That tells ChatGPT what to avoid and also makes it less likely that minor bugs that were fixed earlier in the debugging process won't reappear.

If you can isolate the problem to a specific line of code and ChatGPT can't seem to fix it, **search the internet** for examples of correct application of that code. This is especially useful for libraries whose interfaces may have changed after ChatGPT was trained. Be sure to prompt ChatGPT to make the change so that it has the correct version in its context for the remainder of the conversation.