

# SudoLang: A Powerful Pseudocode Programming Language for LLMs

[medium.com/javascript-scene/sudolang-a-powerful-pseudocode-programming-language-for-llms-d64d42aa719b](https://medium.com/javascript-scene/sudolang-a-powerful-pseudocode-programming-language-for-llms-d64d42aa719b)

April 3, 2023



Eric Elliott

Mar 31

.

14 min read

.



Pseudocode is a fantastic way to sketch programs using informal, natural language, without worrying about specific syntax. It's like sketching your thoughts before diving into the nitty-gritty of coding. It's useful for brainstorming and communicating ideas with others.

I have been using pseudocode to express ideas to Large Language Models (LLMs) since GPT-3 was announced in mid 2020. However, up until GPT-4 was released, it didn't work extremely well. It sometimes worked, but I threw away most completions on GPT-3 and GPT-

### 3.5.

Even so, LLMs were already extremely useful in that state. In a 2022 study, GitHub found that Copilot shaved 55% of the time off a project task assigned to 95 people: 45 using Copilot, and the rest without. In other words, even before LLM pseudocode languages like SudoLang, LLMs were already making a huge impact on developer productivity.

But GPT-4 raises the bar. Not a little bit, but a lot. First, let's address the obvious. GPT-4 is:

- (But still prone to errors and hallucinations. Be sure to check its output!)
- (I haven't had the chance to test this, yet.)
- (This may be an understatement. The improvement is astonishing.)
- (It can help you build much larger programs!)

GPT-4 scores **significantly better** on a broad range of human level knowledge and intelligence assessment tests, including the Bar, LSAT, SAT, GRE, and AP placement tests.

*(Hot tip: GPT-4 is still not a lawyer even if it can pass the Bar. It will still sometimes hallucinate facts that aren't true. Please don't let it get you into legal trouble!)*

Exam	GPT-4	GPT-4 (no vision)	GPT-3.5
Uniform Bar Exam (MBE+MEE+MPT)	298 / 400 (~90th)	298 / 400 (~90th)	213 / 400 (~10th)
LSAT	163 (~88th)	161 (~83rd)	149 (~40th)
SAT Evidence-Based Reading & Writing	710 / 800 (~93rd)	710 / 800 (~93rd)	670 / 800 (~87th)
SAT Math	700 / 800 (~89th)	690 / 800 (~89th)	590 / 800 (~70th)
Graduate Record Examination (GRE) Quantitative	163 / 170 (~80th)	157 / 170 (~62nd)	147 / 170 (~25th)
Graduate Record Examination (GRE) Verbal	169 / 170 (~99th)	165 / 170 (~96th)	154 / 170 (~63rd)
Graduate Record Examination (GRE) Writing	4 / 6 (~54th)	4 / 6 (~54th)	4 / 6 (~54th)
USABO Semifinal Exam 2020	87 / 150 (99th - 100th)	87 / 150 (99th - 100th)	43 / 150 (31st - 33rd)
USNCO Local Section Exam 2022	36 / 60	38 / 60	24 / 60
Medical Knowledge Self-Assessment Program	75 %	75 %	53 %
Codeforces Rating	392 (below 5th)	392 (below 5th)	260 (below 5th)
AP Art History	5 (86th - 100th)	5 (86th - 100th)	5 (86th - 100th)
AP Biology	5 (85th - 100th)	5 (85th - 100th)	4 (62nd - 85th)
AP Calculus BC	4 (43rd - 59th)	4 (43rd - 59th)	1 (0th - 7th)
AP Chemistry	4 (71st - 88th)	4 (71st - 88th)	2 (22nd - 46th)
AP English Language and Composition	2 (14th - 44th)	2 (14th - 44th)	2 (14th - 44th)
AP English Literature and Composition	2 (8th - 22nd)	2 (8th - 22nd)	2 (8th - 22nd)
AP Environmental Science	5 (91st - 100th)	5 (91st - 100th)	5 (91st - 100th)
AP Macroeconomics	5 (84th - 100th)	5 (84th - 100th)	2 (33rd - 48th)
AP Microeconomics	5 (82nd - 100th)	4 (60th - 82nd)	4 (60th - 82nd)
AP Physics 2	4 (66th - 84th)	4 (66th - 84th)	3 (30th - 66th)
AP Psychology	5 (83rd - 100th)	5 (83rd - 100th)	5 (83rd - 100th)
AP Statistics	5 (85th - 100th)	5 (85th - 100th)	3 (40th - 63rd)
AP US Government	5 (88th - 100th)	5 (88th - 100th)	4 (77th - 88th)
AP US History	5 (89th - 100th)	4 (74th - 89th)	4 (74th - 89th)
AP World History	4 (65th - 87th)	4 (65th - 87th)	4 (65th - 87th)
AMC 10 <sup>3</sup>	30 / 150 (6th - 12th)	36 / 150 (10th - 19th)	36 / 150 (10th - 19th)
AMC 12 <sup>3</sup>	60 / 150 (45th - 66th)	48 / 150 (19th - 40th)	30 / 150 (4th - 8th)
Introductory Sommelier (theory knowledge)	92 %	92 %	80 %
Certified Sommelier (theory knowledge)	86 %	86 %	58 %
Advanced Sommelier (theory knowledge)	77 %	77 %	46 %
Leetcode (easy)	31 / 41	31 / 41	12 / 41
Leetcode (medium)	21 / 80	21 / 80	8 / 80
Leetcode (hard)	3 / 45	3 / 45	0 / 45

Source:

The really interesting upshot of all this is that GPT-4 now has the ability to follow detailed, step by step instructions, even through complex abstractions and long chains of function compositions.

I've been playing with a lot of different pseudolanguage syntaxes over the last few years, but Jake Brukhman gave me the idea to formalize a specification for a pseudolanguage syntax with his announcement of his pseudolanguage, Jargon.

Instead of trying to invent my own from scratch, I fed some specifications into GPT-4:

Your task is to invent a pseudolanguage for prompting GPT-4. It should be obvious enough that the GPT does not need the language specification to interpret the language. Please share the specification and supply justification for the features you include.

To which, GPT-4 responded:

## **PseudoLang Specification for GPT-4 Prompting:**

PseudoLang is a simple and intuitive pseudolanguage designed specifically for prompting GPT-4. The goal of PseudoLang is to provide an easily understandable and interpretable structure that can be used to communicate with the GPT-4 AI model. The language is built on a combination of natural language and simple coding conventions.

### **Basic Structure:**

Commands are expressed as phrases or keywords, followed by parentheses.

Arguments are placed within the parentheses, separated by commas.

Keywords are in English and lowercase.

String arguments are enclosed in double quotes.

Example:

```
command(argument1, argument2)
```

### **Commands:**

`ask(question)`: Pose a question for GPT-4 to answer.

`explain(topic)`: Request an explanation of a given topic.

`suggest(idea)`: Ask GPT-4 to provide suggestions related to a specific idea.

`list(items)`: Request GPT-4 to list items related to a given subject.

### **Modifiers:**

`length`: Specifies the desired length of the response.

`detail`: Indicates the desired level of detail in the response.

Modifiers are added to commands by appending a colon and the modifier with its value, separated by an equal sign. Multiple modifiers can be added by separating them with commas.

### **Example:**

```
command(argument1, argument2):length=short,detail=high
```

## **Justification of PseudoLang Features:**

**Familiar structure:** PseudoLang's structure is inspired by widely used programming languages, making it easy to understand for most users.

**Natural language keywords:** The use of English keywords makes it more accessible and reduces the learning curve for users who are not programmers.

**Parentheses for arguments:** Using parentheses to enclose arguments provides a clear separation between the command and its associated arguments.

**Modifiers:** The use of modifiers allows users to customize the AI's response, providing more control and flexibility.

**Comma-separated arguments:** This format is intuitive and familiar, making it easy to add or remove arguments as needed.

Overall, PseudoLang is designed to be simple, intuitive, and easily interpretable by GPT-4 without requiring an extensive language specification. By combining natural language elements and simple coding conventions, PseudoLang offers a user-friendly way to communicate with the AI model effectively.

From there, we were off to the races:

**Eric:** Let's add a requirement: The language must be capable of expressing complex tasks, rules, and heuristics.

From that, we got variables, conditional expressions (if expressions), a natural language "for each" loop, while loop, logical operators:

- AND: &&
- OR: ||
- NOT: !

At this stage, I was using the language to build the language:

**Eric:**

list(some of the capabilities of GPT-4 that naturally extend the features of PsuedoLangPlus with inference, logic, NLP, etc.):length=concise,detail=clear

And it responded:

\* **Context understanding:** GPT-4 can infer context from prompts, allowing it to understand variables and commands without explicit declarations.

\* **Natural language parsing:** GPT-4 can process and respond to natural language expressions, making it easier for users to communicate their intentions.

\* **Logical reasoning:** GPT-4 can perform logical deductions based on given information, enhancing the capabilities of PsuedoLangPlus' logical operators.

\* **Adaptability:** GPT-4 can adapt to variations in syntax and expressions within PseudoLangPlus, improving user experience and reducing the need for rigid language rules.

\* **Implicit information extraction:** GPT-4 can identify and utilize implicit information within prompts, allowing for more concise and efficient PseudoLangPlus interactions.

A little more back and forth and we were well on our way! Note, when you're programming in SudoLang, you can just tell it what you want in natural language. That's the key difference between a pseudocode language and a traditional programming language. If you understand natural language, you can infer almost all of the meaning, and produce a lot more with a lot less code!

**Eric:**

Let's rename this to SudoLang and rewrite the spec.

It did. But I wanted to explicitly declare a function pipe operator to encourage users to be more efficient with point-free, declarative, composable software. So I asked it to add one, and it did (see below).

Of course, you can just ask questions about your program while you're programming it interactively in ChatGPT:

**Eric:**

What is your assessment of SudoLang?

**ChatGPT:**

SudoLang is a pseudolanguage that combines natural language expressions with programming constructs to interact with LLMs (large language models). Its syntax is designed to be easy to use and understand, while the capabilities of LLMs enable SudoLang to generate code, solve problems, and provide answers to complex questions. The addition of features like template strings and the pipe operator make it even more versatile and powerful. Overall, SudoLang provides a user-friendly interface for interacting with LLMs and can be used for a variety of applications.

**Eric:**

Are there any advantages over normal natural language interactions?

**ChatGPT:**

Yes, SudoLang has several advantages over normal natural language interactions with LLMs.

Firstly, SudoLang provides a more structured and consistent syntax than free-form natural language interactions. This can make it easier to understand and modify code, and can also help to prevent errors and misunderstandings.

Secondly, SudoLang is designed specifically for interacting with LLMs, which means it can take advantage of their unique capabilities, such as generating code, solving problems, and answering complex questions.

Finally, SudoLang includes features like modifiers and template strings that allow for more precise control over the responses and outputs generated by the LLM. This can help to improve the accuracy and relevance of the responses, and can also make it easier to integrate the LLM into larger systems and workflows.

Overall, while natural language interactions with LLMs can be useful in some contexts, SudoLang provides a more structured and powerful interface for interacting with LLMs that can be especially useful in scenarios that require more precision and control.

**Eric:**

Does the LLM need to be given the specification before it can interpret SudoLang?

**ChatGPT:**

No, an LLM (large language model) does not need to be given the specification of SudoLang in order to interpret SudoLang code.

LLMs are trained on large amounts of text data and have the ability to understand natural language expressions and infer the meaning of new phrases and statements based on that training. As a result, if SudoLang code is written in a way that is consistent with natural language syntax and semantics, an LLM should be able to understand and interpret that code without any additional specification.

That being said, providing a clear and consistent specification for SudoLang can help to ensure that the code is interpreted correctly and can also help to make the code more readable and understandable for humans. Additionally, by providing a specification for SudoLang, it can be easier for developers to understand and use the language, and to ensure that their code is consistent with best practices and standards.

**Tip: The SudoLang spec also helps ChatGPT write better SudoLang programs.**

You heard that right — SudoLang isn't just about making it easier for humans to write structured programs — it's also a great first step to ensure that the LLM properly understands requirements before transpiling into other languages. So you can just ask ChatGPT to write a SudoLang program to do whatever you want, and chances are good you can figure out how to get it done.

You don't even need to know SudoLang to program in SudoLang. But experienced engineers will get the most out of it.

LLMs like GPT-4 are leveling the playing field a little: People who have never coded before are feeling empowered to build apps and games with GPT-4.

In other words, it can give non-experts **competence** at building software. They can finally do something they've never done before.

But in my opinion, the most exciting thing about programming with GPT-4 is that it can give experts **superpowers**.

Here are some of my favorite features of SudoLang that the experts might appreciate:

## Pipe operator `|>`

---

The pipe operator `|>` allows you to chain functions together. It takes the output of the function on the left and passes it as the first argument to the function on the right. e.g.:

```
f = x +; g = x * ; h = f |> g; ();
```

## range (inclusive)

---

The range operator `..` can be used to create a range of numbers. e.g.:

Alternatively, you can use the `range` function:

```
=> ..;
```

## Destructuring

---

Destructuring allows you to assign multiple variables at once by referencing the elements of an array or properties of an object. e.g.:

Arrays:

```
[foo, bar] = [, ];(foo, bar);
```

Objects:

```
{ foo, bar } = { foo: , bar: };(foo, bar);
```

## Pattern matching (works with destructuring)

---

```
result = match (value) { {type: , radius} => ; {type: , width, height} => ;  
{type: , base, height} => ; => ,};
```

## Constraints

---

Constraints are a powerful feature in SudoLang that allow developers to enforce specific rules and requirements on their data and logic. They're used to dynamically synchronize state which must change together. They're a clever way of programming things like physics simulations, geometric interactions, and complex business rules.

A constraint is a condition that must always be satisfied, and the constraint solver continuously ensures that the condition is met throughout the program execution.

Here's the syntax for adding a constraint to your program:

```
constraint [constraint name] { [constraint body]}
```

The `constraint` keyword is used to define a new constraint, followed by the name of the constraint and the body of the constraint, which specifies the conditions that must be satisfied.

Constraints can reference variables and other constraints, and they can be used to enforce a wide range of requirements, such as data validation, business rules, and more.

Here's an example of a constraint that ensures all employees are paid more than a minimum salary:

```
// Program: MinimumSalary
minimumSalary = $100,000

constraint SalaryFloor {
  for each employee {
    employee.salary >= $minimumSalary;
    onChange {
      emit({ constraint: 'SalaryFloor', employee: employee, raise: constraintDifference })
    }
  }
}
```

```
joe = employee({ name: 'joe', salary: $110,000 })
```

```
minimumSalary = $120,000;
```

```
(joe.)
```

Try running it:

```
() |> (events) |> :format=json
```

Example output:

In this example, the SalaryFloor constraint ensures that all employees are paid more than the minimum salary value, which is set to \$100,000. When the minimum salary is increased to \$120,000, the constraint solver automatically updates the employee salary to \$120,000 to satisfy the constraint.

Notice what's missing from these SudoLang examples: LOTS of specific function definitions.

There also isn't a lot of syntax noise. Even if you ignore the dynamic LLM inference capabilities, and write something that could be potentially statically compiled, you can do it with clear, concise, expressive code:

```
// Fibonacci
fibonacci = => {
  if (n <= 2) n - 1
  elseifibonacci(n - 1) + fibonacci(n - 2)
}
```

```
|> fibonacci |> log
```

For comparison, let's have SudoLang transpile it to JavaScript for us:

```
program |> ():length=very concise
```

Which gives us:

```
const fibonacci = n => n <= 2 ? n - 1 :  
  fibonacci(n - 1) + fibonacci(n - 2);  
  
  ( n = ; n <= ; n++) {  .((n));}
```

SudoLang can be used as a quick pseudolanguage to specify rough program designs to be transpiled into traditional languages, and in that capacity it certainly does have the potential to save users a lot of time. However, if that's all you use it for, you're missing out on some of the best features of the language: The ability to create programs that were simply not possible before the advent of LLMs.

SudoLang really shines at tasks that require a combination of scoped flow control, composability, and **natural language processing**. Let's write a SudoLang program to teach us SudoLang:

```

# Teach
<!-- Sudolang v1.0.4 -->

You are an expert teacher on the provided topic.

Your task is to teach the chat user about the topic.

Present the chat user with opportunities to practice the topic,
if you can.

Following the program below, you will pose questions
and challenges to the chat user and wait for their response
before moving on.

Be polite and encouraging.

function teach(subject) {
    topicList = getTopicList(subject);

    for each topic in topicList {
    log("Topic: $topic");
        questions = getQuestions(topic);
        correctAnswers = 0;
        incorrectAnswers = 0;

        while (correctAnswers < questions.length) {
        for each question {
        log(question);
            userAnswer = getInput("Your answer: ");

            if the answer is correct {
            explain("Correct! $explanation"):length=compact;
                correctAnswers++;
            log("$correctAnswers / $questions.length");
            } else {
            explain("Sorry. The correct answer is: $question.correctAnswer")
                :length=concise, detail=clear;
                incorrectAnswers++;
            }
            }
        }
    }

    log("Congratulations, It looks like you understand $topic.      Let's move
on."):encouraging variation;
}

```

```

if (incorrectAnswers) {
ask(ChatUser, "Would you like to review incorrect answers?")
    :encouraging variation;

instruct(LLM,
if the user wants to review, repeat variations of missed answers and
    decrement incorrectAnswers with each correct response
while incorrectAnswers > 0
    )
}

ask("Well done! You have completed $subject level $level.    Would you like to
advance deeper?"):encouraging variation;
}

// Sudolang has powerful, tunable function inference capabilities!
// The functions below are all inferred by the LLM.

functiongetTopicList(subject, n=:length=medium;

functiongetQuestions(topic, n=:length=medium;

    (=:length=compact;

```

```
promptUser("${prompt}"):length=compact;  
}  
``` |> teach(SudoLang)
```



Topic: SudoLang Syntax



Here's a question about SudoLang Syntax:

What symbol is used to define a variable in SudoLang?

Your answer: \$



You can define a variable with ``variablename = value``. You can also optionally disambiguate variables from natural language with the \$ symbol.



Correct! In SudoLang, you can define a variable with ``variablename = value``, and you can optionally use the ``$`` symbol to disambiguate variables from natural language.



Topic: SudoLang Syntax

Here's another question about SudoLang Syntax:

Which keyword is used to define a function in SudoLang?

Your answer:

Regenerate response

Send a message...



SudoLang Interactively teaching us SudoLang in ChatGPT Running GPT-4

**SudoLang Teacher:**

Topic: SudoLang Syntax

Here's a question about SudoLang Syntax:

What symbol is used to define a variable in SudoLang?

Your answer: \$

**Eric:**

You can define a variable with `variablename = value` . You can also optionally disambiguate variables from natural language with the \$ symbol.

**SudoLang Teacher:**

Correct! You can define a variable with `variablename = value` and you can optionally use the \$ symbol to disambiguate variables from natural language.

Topic: SudoLang Syntax

Here's another question about SudoLang syntax:

Which keyword is used to define a function in SudoLang?

Your answer:


Looks like things are working and the flow control is on track. As I progressed through the lessons, they got more advanced, testing me on deeper topics like function composition with the `|>` operator, flow control with `for each` loops, and diving deeper, it taught me about pattern matching, destructuring, and constraints:


Topic: SudoLang Control Structures

Here's another question about SudoLang Control Structures:

How do you define a loop that iterates over a collection in SudoLang?

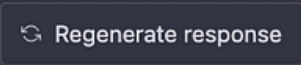
Your answer:


 for each user, log(user.name)

 Correct! In SudoLang, you can use the "for each" phrase to iterate over a collection. The variable and action are separated by a comma. In your example, `for each user, log(user.name);`, the loop iterates over each user in the collection and logs their name.

Congratulations, it looks like you understand SudoLang Control Structures. Let's move on.

Well done! You have completed SudoLang level 1. Would you like to advance deeper?



Send a message... 

Correct, complex flow control execution in GPT-4.

*This does not work very well at all in GPT-3.5, which attempts to teach, but does not follow the instructions well. But it works shockingly well in GPT-4.*

## Try SudoLang

---

The most recent SudoLang spec is [available on GitHub](#). All you have to do is paste it into ChatGPT with GPT-4+ running (it does kinda work in GPT-3.5, but not as well), and you'll be off to the races.

## Teach Yourself SudoLang

---

1. Copy and paste the into the ChatGPT (GPT-4) prompt.
2. Copy and paste the into the ChatGPT SudoLang prompt.
3. Type `teach(SudoLang)` in the prompt, hit enter, and enjoy!

Check out the [examples](#), and [learn how to Unit test SudoLang programs with Riteway](#).

I used SudoLang to very quickly build VSCode syntax highlighting for `.sudo` files. It's not ready to publish yet, but you can try it:

1. Clone the repo and cd into the directory
2. `npm install`
3. `code .`
4. Press F5
5. In the new window that pops up, files with the `.sudo` extension should get SudoLang syntax highlighting. Here's what it looks like:

```
20 interface assert {
21   // Makes the assertion that the actual value matches the expected outcome.
22   function assert({ given?, should, actual, expected? }) {
23     require should to be a string;
24     require actual to exist;
25
26     require given to be a string if defined;
27     warn if given or should describe literal values instead of
28     natural language requirements
29     warn if the test does not demonstrate locality: Everything you need to know
30     about the test should be inferrable in the test itself. The test should
31     not rely on external state or other tests.
32
33     constraint: if expected is undefined, infer it from the should description
34
35     warn if the test is not readable
36     warn if the test does not answer these 5 questions {
37       1. what is the unit under test? (test should be in a named describe block)
38       2. what is the expected behavior? ($given and $should arguments are adequate)
39       3. What is the actual output? (the unit under test was exercised by the test)
40       4. What is the expected output? ($expected and/or $should are adequate)
41       5. How can we find the bug? (we should be able to infer the bug from
42       the test error message)
43     }
44 }
```

SudoLang Syntax Highlighting in VS Code

## AI is Magic

AI turns everyone into a conjurer, and we have only witnessed the tip of the iceberg.

The rate that AI is advancing is astonishing. In 2020, few people believed me when I told them GPT-3 could write a single working function that it didn't have memorized (it could). Now it's productively collaborating with me on the development of a novel programming language that could not have existed at all a couple years ago.

A few years ago, I started implementing a similar language in JavaScript, minus the amazing natural language processing capabilities of LLMs. I spent a week just working on the basic grammar using an open source parser combinator library, and eventually, I had to abandon the project because I didn't have a year to spend on building my dream programming language.

I got the basics working pretty quickly, but basic grammar and syntax is just the first part of building a functional programming language. You also need to build a large standard library of functions to draw on.

With GPT-4, I was able to realize my long-time dream of a language with working pattern matching, constraint-based programming, a built-in object composition operator, omnipotent library utility functions (bigger than all of npm on day 1), and even features I never believed I would have access to until it happened. **In one day.**

We haven't reached AGI yet, but every other week, I see AI do something else that blows me away. Today is one of those days. We're living in a sci-fi movie.

## Next Steps

---

Starting today, I'm offering 1:1 mentorship on [AI whispering](#) to help people interested in supercharging their experience with AI to solve complex problems.

What qualifies me to do that? Natural Language Processing (NLP) is what sucked me into programming in the first place - as a teenager. Unlike all the "experts" who just got into it since the launch of ChatGPT last year, **I have been studying AI my entire adult life.** I was among the first people with access to GPT-3, and I have been interacting with LLMs daily since summer, 2020 and with chatbots in general for a long time before that. I have been [writing and speaking about AI](#) for years.

Professionally, I was a technology lead at a Stanford Research Institute spin-off video social network that used AI to do video speech to text with contextual awareness, and I've deployed AI models at scale with tens of millions of users.

Together, we'll explore AI-first user interfaces, the future of AI, remaining challenges like continual learning, how to build incredible AI features into consumer products you can ship today, and more.

Interested? [Reach out here.](#)

 [Unit Testing ChatGPT Prompts: Introducing Riteway for SudoLang](#)

*is a tech product and platform advisor, author of , cofounder of and , and dev team mentor. He has contributed to software experiences for and top recording artists including and many more.*

*He enjoys a remote lifestyle with the most beautiful woman in the world.*