Project: building Desmos Expressions

Daus Carmichael

Oct-Dec 2022

Contents

1 Abstract

The inspiration for this project came from a place of boredom. While working on intro-level physics problems for my job as an SI, I soon realized that I was often solving the same equation over and over. I would simply be solving for different variables. So, I used desmos to write the equation out and get the values that the students were given. I would get the final result of the question very easily but I still had to rewrite the equation. If I were able to write the equations necessary for one problem could I plug in the values and find all of the answers from the problem. Perhaps, not strictly the answers but all the information that can be determined with the initial information...

Desmos: link

2 Desmos Syntax

The first step in the process is to create abstract syntax trees (ASTs). When copying text from desmos the syntax is very specific. I will go through each term that desmos supports.

2.1 Numbers

Floats/doubles and integers negative and positive all exist. Desmos's syntax for these is very simple, simply plain text.

Desmos	Plain Text
0	0
-1	-1
8	8
5.2	5.2
-4.0	-4.0

2.2 Variables

Variables like x, y, z, etc. are allowed in desmos. They are represented in plain text directly as written in desmos. That is desmos "a" $\rightarrow$ plain text "a". The only point of note is subscripts. These are represented by a underscore (_) following the variable and the subscript label. a_0, "a-nought" from desmos is a_{0} in plain text. It can be written to desmos as only a_0, however desmos will write this back as the previous expression with the curly braces.

2.3 Plus, Minus, Multiply, Divide

Plus and minus are simple. In desmos $a+b \quad a-b$, are in plain text "a+b" and, "a-b". Multiplication is the first step where it is no longer the simple characters. Desmos has the expression for multiplication $a \cdot b$ this when copied to plain text is "a \cdot b". In a similar vein, division uses a escape character and string. $\frac{a}{b}$ in desmos is "\fracab" in plain text. Here is a table of the expressions from desmos to plain text.

Desmos	Plain Text
a + b	a + b
a - b	a - b
$a \cdot b$	a \cdot b
$\frac{a}{b}$	\frac{a}{b}

Note: The first thing I noticed was that all of the equations in desmos are the same as mathmode in latex. This is undoubtedly not a coincidence and a good place to base what the syntax will be for desmos if something isn't clear. The documentation of latex:mathmode also allows to ensure certain expressions aren't forgotten when creating the syntax for the grammar and interpreter. (Some expressions in mathmode are not supported by desmos)

2.4 Parentheses and Brackets

Parentheses are used for order of operations rules to denote what should be evaluated first. Brackets {, } are used to specify arguments of a function or the subscript of an identifier. **Parentheses**

Desmos	Plain Text
(	\left(
)	\right)

For brackets, there is some weird behavior. When next to a function it appears as plain text but for piece-wise functions it appears as a escaped character. **Brackets**

Desmos	Plain Text
{1}	\left{1\right}
$ident_{subscript}$	ident_{subscript}

These really can only be explained with examples and understanding the grammar. There isn't a great rhyme or reason yet determined.

2.5 Exponents

A simple function in a sense. It is denoted with the caret ^ and curly braces around the "argument" or exponent. **Exponents**

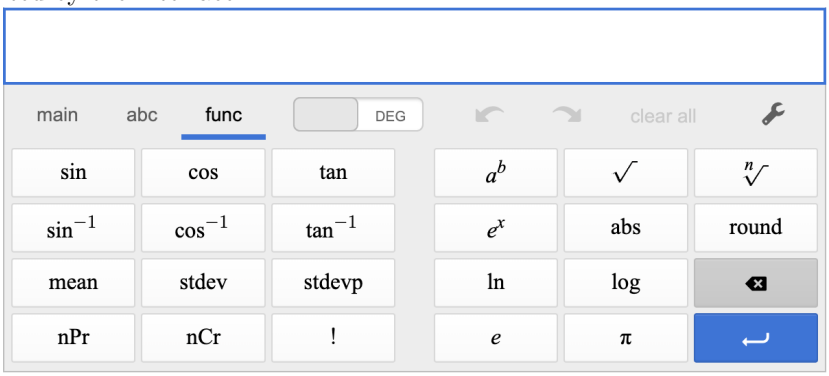
Desmos	Plain Text
x^{y+2}	x^{y+2}
$x^{y}+2$	x^{y}+2

This should make it slightly clearer what the braces are needed for as well as the syntax of the exponent syntax.

2.6 Functions

2.6.1 Common Functions

Desmos supports a number of common mathematical functions. There is a functions tab in the interface as well as keywords that create functions when entered into the input field. These have more specific syntax and are not strictly limited by the interface.



The function tab in desmos. This is not a full set of functions. The following table will be the fully supported syntax of the language.

Functions[1]

Desmos	Plain Text
$\sin(\theta)$	\sin\left(\theta\right)
$\cos(\theta)$	\cos\left(\theta\right)
$\tan(\theta)$	\tan\left(\theta\right)
$\sin^{-1}\left(\frac{O}{H}\right)$	\sin^{-1}\left(\frac{O}{H}\right)
$\cos^{-1}\left(\frac{A}{H}\right)$	\cos^{-1}\left(\frac{A}{H}\right)
$\tan^{-1}\left(\frac{O}{A}\right)$	\tan^{-1}\left(\frac{O}{A}\right)

You may notice some functions missing that are present on normally calculators. $\csc\theta, \sec\theta$, etc. These are not supported by desmos but can be represented in the language identically in some other format.

2.6.2 operatorname Functions

This next section begins the regime where "operatorname{}" syntax is used. This is used for mean, stdev, and the following functions. Each function will

[1]The functions $\sin^{-1}, \cos^{-1}, \tan^{-1}$ are not $\frac{1}{\sin\theta}$... but rather sine inverse, cosine inverse and tan inverse.

be explained as this is important for the evaluation and solution to problems involving said functions.

Mean

Mean is the arithmetic average of the set passed to it. Sets will be described in a later subsection.[2] Mean is defined as

$$\text{Mean: } \bar{x} = \sum_{i=1}^{n} x_i = \frac{x_1+x_2+\ldots x_i+\ldots x_n}{n}$$

Stdev is next, this is the standard deviation of the set. This is a good reference for the next 2 functions STDEV.P vs. STDEV.S in Excel: What's the Difference? As the articles describes, standard deviation **P** is treating the given data as a population while standard deviation **S** is treating the data as a sample of the population, They thus have separate formulas for evaluation.

Sample Standard Deviation

$$\text{STDev.S} = \frac{\sqrt{\sum (x_i-\mu)^2}}{N}$$

Σ: The sum over all the measurements in the dataset
x_i: The i-th value in the dataset
μ: The population mean
N: The total number of observations

Population Standard Deviation

$$\text{STDev.P} = \frac{\sqrt{\sum (x_i-\bar{x})^2}}{N-1}$$

Σ: The sum over all the measurements in the dataset
x_i: The i-th value in the dataset
$\bar{x}$: The population mean
N: The total number of observations

operatorname Functions

Desmos	Plain Text
mean(list)	\operatorname{mean}\left(list\right)
stdev(list)	\operatorname{stdev}\left(list\right)
stdevp(list)	\operatorname{stdevp}\left(list\right)
nPr(n,r)	\operatorname{nPr}\left(n,r\right)
nCr(n,r)	\operatorname{nCr}\left(n,r\right)
round(x)	\operatorname{round}\left(x\right)
floor(x)	\operatorname{floor}\left(x\right)
ceil(x)	\operatorname{ceil}\left(x\right)

The last supported functions are miscellaneous random functions that are not frequently used.

[2] Starting i at 1 or 0 will be dependent on the implementation of the interpreter and rewrite system

2.6.3 Piece-wise Functions

This is a specific subsection of functions the notation is using the curly braces, a condition for where the function applies, a colon, and the function for that region. These functions can be written in the language but the parser cannot recognize them as they are two separate equations in desmos. Additionally the inverse of a piece wise function (solving a piece wise function) is not necessarily possible. Thus the functions are not supported.

2.7 Translation to Grammar

All of the functions, operators and values included in this description have a plain test representation. This is our input so we can move seamlessly between desmos and the text version. [3] As seen in the first projects for the Programming Languages class, the parser reads a file and attempts to give the tokens a name. Sometimes this can prove to be difficult as certain expressions "look" very similar. There are ways to solve this issue but for the sake of the project cases like this were ignored both due to the complexity of evaluation and parsing. The first step ignoring the limitations is to create an abstract syntax tree. Each operator or function has a specific number of argument it accepts. The general form of an expression once parsed is:

```
Operator (Expression1) (Expression2)
```

OR

```
Operator (Expression)
```

This is an expression, this fact allows for each of the expressions that are arguments to contain operators or functions and those can contain operators and so on. The grammar's goal is to read the input and create sentences from the tokens it parses. These tokens, in order to recognize what expression to apply, must match the plain text description given in the grammar. This is the application of the plain text component of the previous section. This is also what is copied into the clipboard when taking a sentence from desmos. There is documentation for the grammar at the bottom of this section. The grammar more specifically shows what it means. It is linked here.
A single example is described and what happens at each step of parsing.

```
Equal.      Exp1 ::= Exp1 "=" Exp2 ;
```

If the parser sees the structure "X=Y", it will construct a node on the tree with the Equal expression. X and Y will then be the corresponding 2 expressions. X and Y are recognized by the rule for Var's.

```
Var.        Exp10 ::= Id ;
```

X and Y then are Id's so the rule for Id's which is just a letter is a applied.

[3] In testing it was discovered that some of the outputs can be sent to further processing in the system but not sent to desmos, however they are human readable. This will be addressed in limitations and future developments.

```
Equal (Var (Id "X")) (Var (Id "Y"))
```

So our grammar parses $X = Y$ into the abstract syntax tree of

```
Equal (Var (Id "X")) (Var (Id "Y"))
```

The grammar includes 46 rules. This includes 2 for comments. The program that runs the language also has a rule for comments that is separate from these two. The rules on syntax are not interesting however. The rules that actually are consequential to

Grammar documentation: github

2.8 Order of Operations

Of course if there was no order of operations then equations written out would have no meaning. So how does the system know what comes first when reading the input? This is what the precedence level means in the expressions. For example, addition, the Plus. expression has a precedence level of 2. This is the same as subtraction as the order these operations are applied doesn't matter. Multiplication, the Times. expression has a higher precedence level of 3. That is because multiplication must be applied before addition. Sidebar: On social media one of the most common "unsolved math problems" is equations used to confuse the order of operations. It may be something like $6/2(1+2)$. This is a) written ambiguously and can be interpreted in multiple ways, and b) not clear what operation is used for the second portion of the equation. This is one of my pet peeves. These aren't interesting questions, for 1 the numbers don't mean anything, and at most its just the application of rules in a strict manner. There isn't much deeper meaning to the question. What is interesting about problems like this is how to program something to read this. This requires giving the reader a way to apply each operator and think which operations need to be applied. In this grammar, the is not ambiguity of operators. The operator is well defined and tells what the equation is doing. Additionally, if the numbers meant something (physically or theoretically) then the equation would not be written like this. That said, the implementation of order of operations in anything that calculates results of equations is incredibly important and must be justifiable. The list of precedences is as follows.

2.8.1 List of Precedence Levels

1. Exp1
   - Equal (=)
2. Exp2
   - Plus (+)
   - Minus (-)
3. Exp3

- Times (+)
- Div (-)
- Negate (-)[4]

4. Exp4
    - Exp (∧)
    - Sqrt (\sqrt)
    - NRt (\sqrt[ ])
5. Exp5 –functions
    - Sin (sin)
    - Cos (cos)
    - Tan (tan)
    - ASin (asin)
    - ACos (acos)
    - ATan (atan)
    - Ln (ln)
    - Log (log)
6. Exp6 –operatorname
    - Mean (mean)
    - Stdev (stdev)
    - StdevP (stdevp)
    - NPR (nPr)
    - NCR (nCr)
    - Round (round)
    - Percent (% of)
7. Exp7
    - Abs (| |)[5]
8. Exp8
    - Elems (<>, <>)
    - List ([ ])
    - Index <>[ ]

[4]Unary negation operator
[5]Absolute value operator

9. Exp9
    - Pi (π)
    - E (e)
    - Integer ($n \in \mathbb{Z}$)
    - Double ($d \in \mathbb{Q}$)
10. Exp10
    - Var Id. (letter)

Hopefully this list makes logical sense. The larger the expression number the earlier the expression is compiled in to the abstract syntax. This makes sense as both sides of the equality for instance should be evaluated before the equality operator is applied. Just as multiplication should be applied before addition and their counterparts likewise. Following the list down it is clear to see how these rules for order can be applied. For single argument operations, however, the order is not as important because the expression is always evaluated fully before the function is applied. So sin(a+b) evaluates Plus (Var (Id "a")) (Var (Id "b"))) and then Sin (Plus (Var (Id "a")) (Var (Id "b"))).

3 Interpreter

Given an expression, one ought to be able to evaluate and solve it. One problem, there isn't memory and the rule "exactly 1 unknown for every 1 equation" in the system. So without substitution, it is impossible to solve more complicated equations (more than one variable). However, if it is one variable the equation can be rewritten and the side without the variable sent to the Interpreter to be evaluated. The rules for interpreting the equation are relatively simple. As of now however (as before) there is no way to evaluate variables. The supported functions are included in the following list.

- Double d = d
- Integer i = i (cast to double)
- Pi = pi
- E = 2.71828182845
- Plus i j = i + j (expressions are evaluated again, potentially recursively)
- Minus i j = i - j
- Times i j = i * j
- Divide i j = i / j
- Negate i = -1 * i

- Fact i = i![6]
- Exp i j = i**j (in traditional math characters i ^ j)
- Sqrt i = sqrt i
- Nrt i j = j**1/i
- Ln i = ln i
- Log i = $\log_{10}$ i
- Sin i = sin i [7]
- Cos i = cos i
- Tan i = tan i
- ASin i = asin i
- ACos i = asin i
- ATan i = atan i
- Abs i = abs i
- Round i = round i (cast to double)
- Percent i j = (i / 100) * j

Source code: Interpreter.hs
Aux functions: Factorial.hs

4 Procedure

To solve each equation the trees must be rewritten to solve for a single variable. How do we know how to do this? It is taught in middle school and high school and gradually we just recognize what to move over from side to side. However, there is a much more mechanical background behind this. We recognize that

$$y = m \cdot x + b$$
$$\frac{y}{m} = x + b$$

is the incorrect solution. But why? In this case because we ignored the correct order of operations. The plus is the last operation, so to "undo" it it must be applied to the opposite side first. Because of the structuring of the grammar we have the plus on the top of the expression. This is because the grammar prioritizes parsing m*x over the addition. We can see correctly that the solution should be

[6] Factorial is implemented using helper functions
[7] All trig functions use exclusively radians

$$y = m \cdot x + b$$
$$y - b = mx$$
$$\frac{y-b}{m} = x$$

Thanks to my middle school algebra teacher Ms. Brewster, I have ingrained in my head, "whatever you do to one side, you must do to the other. This is represented ion the apply function in the algebra code. This code was somewhat complicated/spaghetti code. This was because sometimes in non-commutative operations require moving the variable to the right side, but this goes against what the code wants to happen so a flip of the sides is used.

Every time an operator is in an expression the other side has the inverse applied to it. There is some "thinking" done with conditionals by the computer but it is boring and while important relatively evident. (If you read this paper you should be able to do basic algebra)

```
(Var (Id "y")) =
(Plus (Times (Var (Id "m")) (Var (Id "x"))) (Var (Id "b")))

Times (Var (Id "m")) (Var (Id "x"))
=
Minus (Var (Id "y")) (Var (Id "b"))

Var (Id "x")
=
Div (Minus (Var (Id "y")) (Var (Id "b"))) (Var (Id "m"))
```

The process is literally as simple as this excerpt. It rewrites the equation searching for a specific variable. In this case we are looking for x. Ensuring that x stays on the left, the first step is to flip the sides.

```
Equal
    (Plus (Times (Var (Id "m")) (Var (Id "x"))) (Var (Id "b")))
    (Var (Id "y"))
```

Next the function is broken into it's right and left components.

Left

```
(Plus (Times (Var (Id "m")) (Var (Id "x"))) (Var (Id "b")))
```

Right

```
(Var (Id "y"))
```

Following this the next step is to remove the operatro fromt he left side while "digging" for the desired variable. To remove the operation we must apply the inverse. The inverse of $z = f(x) = x + b$ is $f^{-1} = z - b$ **Left**

```
Times (Var (Id "m")) (Var (Id "x"))
```

Right

```
Minus (Var (Id "y"))  (Var (Id "b"))
```

Again applying the same logic but now with multiplication. The inverse of $z = m$ is $x = \frac{z}{m}$.

Left

```
Var (Id "x")
```

Right

```
Div (Minus (Var (Id "y")) (Var (Id "b"))) (Var (Id "m"))
```

This has met the criteria to break, and exits return the equation as a set of 2 parts, the left and the right.

$$x = \frac{y-b}{m}$$

5 Instructions

The code is executable which is very exciting. The execution and build is simple. It is currently only supported on Mac/LinuxOS. rust must be installed which can be done simply by going to the rust homepage or rather than running it yourself you can check this VOD. If all requirements are met then the execution is simple.

5.1 Requirements

- Rust Installation
- Java Installation
- GHC - Glasgow Haskell Compiler 9.4.2
- BNFC 2.9.1
- MacOS or Linux

The first step is to clone the repository from daus-s. After the repo is cloned the next step is to build everything. If everything is installed run build.

```
$ chmod +x build
$ ./build
```

For security please check the content of build. A copy is linked here. Following this everything will be compiled. This is when it can now be run. Go to desmos and grab an expression with as many variables and numbers as you would like using basic functions (not including factorial or round). Copy the text into a the window and run the solve command.

```
$ chmod +x solve
$ ./solve "\pi-10=56.5\cdot a"
```

The output should look something like:

```
>a=\frac{(\pi-10)}{(56.5)}
```

This can be run through the interpreter now. This is one of the future developments that would be automatically done for single-variable systems. Take the part entirely to the right of the equal sign and pass this to Desmos, the Haskell executable for the interpreter.

```
$ echo "\frac{(\pi-10)}{(56.5)}" | ./Desmos
```

And the result will be.

```
> -0.12138774064442844
```

Mess around with what is solvable and is not solvable. Some rules in general if the variable exists twice in the expression it is unsolvable. If there are more than 1 variables in the equation, you will get that number of equations back. If this number is greater than 1, no arrangement of replace equations will give you a value in the interpreter. These are some of the developments I would like to see.

Another worked example.

```
$ ./solve "56.5=a\cdot(\pi+837)"
> a=\frac{(56.5)}{(\pi+837)}
$ echo "\frac{(56.5)}{(\pi+837)}" | ./Desmos
>6.725056882560067e-2
```

6 Limitations

This code is severely limited as difficult as it was. It is based in a functional language for evaluating expressions parsing trees and rewrite algebraically. This is not a limitation from what I experienced.

6.1 Numerical Evaluation

The code unfortunately can only evaluate numerically and cannot be returned to the form that is parsed. This means that in certain expressions it may be possible to have tokens like:

```
Minus (Int 1) (Int 1)
```

This normally could be evaluated as 0 but in this case it remains a whole expression. This can also cause mathematical issues. I'm sure you can see that if an expression evaluates to 0 but isn't the expression for 0 it would be incredibly easy to have a divide by 0 error. This can be replicated by the following code.

```
$ echo "1-1=\frac{1}{x}" | ./TestDesmos
> Equal (Minus (Int 1) (Int 1)) (Div (Int 1) (Var (Id "x")))
$ ./solve "1-1=\frac{1}{x}"
> x=\frac{(1)}{(1-1)}
```

```
$ echo "\frac{(1)}{(1-1)}" | ./Desmos
> Infinity
```

This could prove to be a more serious problem if memory is created and variables persist their values. You can get divide by 0 errors very easily.

6.2 Decimal Format

During the testing of the grammar in ./testgrammar.sh and the data used, testgrammar.file a mysterious error started showing up. Certain numbers would make the computer complain and have parsing errors. This turned out to be due to some floats having no leading zero. That is 0.01 is a valid double in the language but .01 is not. In desmos syntax both of these are valid. This means that it is a flaw and limitation that can not perfectly match the desmos text.

6.3 Parentheses

Desmos is not stranger to weird syntax's. For example the syntax for absolute value is \left—_\right—. This is the same for parentheses except using the parentheses characters: \left(_\right). However the parser only recognizes the specific parsing for parentheses when they are part of a larger expression.

6.4 Variable memory

One of the most exciting possible future developments is also one of the biggest flaws and shortcomings of this "language" compared to desmos itself. Desmos allows for values to be stored as

6.5 Lists vs. Tuples

The grammar miserably failed at differentiation tuples and lists. Both are demarcated by commas in desmos and to stay true to the grammar made it impossible to change to a different delimiter. The current solution that will allow erroneous input to parse successfully is to say the functions requiring a tuple take a list as an argument. This does allow for all functions part of desmos to be parsed but it misses the separate intent of the functions.

6.6 Evaluation of lists

While there were no functions used that operated on lists in the algebra system, lists can not be evaluated nor there elements.

7 Future Enhancements and Developments

While the interpreter does provide some use in the current state, it could be used much more extensively. One idea is to check the number of variables in a

system and if it is one, then solve for the variable and plug the resulting equation into the interpreter. This is a relatively easy extension and application of the interpreter. Another thought about the interpreter is to create some sort of memory. This could again check for variables and add the solutions to a table. This table could have these variables saved and easily replace them in future equations. In terms of an actual programming language, it may be possible to implement something similar to what is described above. That is each variable would have an equation and a table could store the information for each. Then when it is accessed the . This is the more programming languages side of the problem. In the sense that choosing our value from the table and plugging it into some equation(function) like lambda calculus applying a value to some function and taking the result. However, there are other developments to be done. Mainly mathematically and in the domain of software development. The first mathematically, is to make the algebra system much more rigorous and complete. Currently the system can solve very few equations. This may involve implementing evaluation in the rewrite system(rust program) in order to get to forms that are solvable. This could be the quadratic formula (3rd, 4th and 5th? order polynomials also have this kind of solution) , Newton-Raphson method; that is set the equation to 0 and solve for better and better approximations around the zero point(s). There are also more ways to solve expressions that I have not listed here. Software development-wise; this program could return the input to desmos. This would require writing multiple versions of the resultant equations because of some of the formatting issues such as desmos taking the asterisk (*) and the grammar taking \cdot for the multiplication operator. This is not a mistake as the grammar takes the result of desmos when pasted into a text file. However, this can be solved relatively simply. This would allow for the program to look for information from desmos, solve the equations inputted to desmos and then return all the other variables Additionally, it can always be made to be more efficient. This can be done by looking at the code checking run times, multi-threading many different processes. If it was made to be Perhaps the most ambitious goal and or development would be to completely negate my entire undergraduate career. Using natural language processing and loading files with physics equations loaded (it would require the support for calculus and differential equations; but once again mechanical solutions exist) one could pass in a document containing exam questions and the system would be able to solve the problems without thinking.

7.1 Author's note/Disclaimer

Of course, this wouldn't neglect the necessity to learn the basics (2+2=5) nor encourage cheating on exams and homework, but just as computer science development no longer requires students to know all the minute details of an assembly program has, so too physics students may be able to focus on the higher level questions ignoring the lower level details, with a basic understanding. This seems to be the way things are going, computers solving everything, and as much as I hate something else doing the work for me, this project has also made

me realize how much work must go into developing systems like this and how we will just keep asking harder and harder questions.

8 Conclusion

This project highlights the creation of a grammar to interpret and parse equations in a plain text based calculator system. desmos uses text to allow for variables to be assigned and expressions evaluated. This is the perfect arena to begin looking at the structure of the mathematical language. The reason it is ideal is because the structure and syntax of the operators is copy and pasted into a text file and will never change. That is the multiplication operator is applied every single time in the same way no matter what arguments the multiplication argument takes. This means it is possible to use the rules desmos provides and create abstract syntax trees. As discussed, abstract syntax trees make solving equations very simple. As I am wrapping up this project I realize just how far short I fell from an actual programming language but also how difficult this was to teach a computer to do algebra, albeit incredibly simple algebra.

9 Sources

Works Cited Works Cited

Aditya, G. "From Zero to Hero Free Learning Tutorials for HTML, CSS, Javascript, PHP, Python, C++, C#, Java and More. Page Topic: Rust Execute Command." Mockstacks, https://mockstacks.com/Rust_Execute_command.

ANimator120ANimator120 2, and dpbriggsdpbriggs 8222 silver badges88 bronze badges. "Print Value of Bool in Rust." Stack Overflow, 1 Jan. 1968, https://stackoverflow.com/questions/65056996/print-value-of-bool-in-rust.

By: IBM Cloud Education. "What Is Natural Language Processing?" IBM, https://www.ibm.com/cloud/learn/natural-language-processing.

Callum, Victoria Mac. "Parentheses Singular: Definition and Uses." Between the Lines by English Forward, Between the Lines by English Forward, 24 Sept. 2020, https://www.englishforums.com/blog/parentheses-singular-definition-and-uses/.

"Commutative Property." Wikipedia, Wikimedia Foundation, 15 Nov. 2022, https://en.wikipedia.org/wiki/Commutative_property.

"How to Swap Two Numbers without Using a Temporary Variable?" GeeksforGeeks, 13 Dec. 2022, https://www.geeksforgeeks.org/swap-two-numbers-without-using-temporary-variable/.

"Integer." Wikipedia, Wikimedia Foundation, 11 Dec. 2022,
https://en.wikipedia.org/wiki/Integer.

Just a learnerJust a learner 25.6k4949 gold badges148148 silver badges229229 bronze badges, et al. "Understanding of Rust Option's unwrap_or_default Method." Stack Overflow, 1 June 1968, https://stackoverflow.com/questions/67578062/understanding-of-rust-options-unwrap-or-default-method.

Linuxize. “Bash: Write to File.” Linuxize, Linuxize, 4 Jan. 2021,
https://linuxize.com/post/bash-write-to-file/.

“Newton’s Method.” Wikipedia, Wikimedia Foundation, 18 Dec. 2022,
https://en.wikipedia.org/wiki/Newton%27s_method.

“R/Rust - [Question] What Is the Some Keyword?” Reddit,
https://www.reddit.com/r/rust/comments/4ryu7a/question_what_is_the_some_keyword/.

“Rational Number.” Wikipedia, Wikimedia Foundation, 17 Dec. 2022,
https://en.wikipedia.org/wiki/Rational_number.

“Rust - File Input/ Output.” Tutorials Point,
https://www.tutorialspoint.com/rust/rust_file_input_output.htm.

“Rust by Example.” Strings - Rust By Example, https://doc.rust-lang.org/stable/rust-by-example/std/str.htmlstrings.

Rust Loop over String Chars - Dot Net Perls, https://www.dotnetperls.com/loop-chars-rust.

“Rust Loop over Vector.” Hacker Touch, https://www.hackertouch.com/rust-loop-over-vector.html.

user15435182user15435182, and Chayim FriedmanChayim Friedman 27.9k44 gold badges3232 silver badges5555 bronze badges. “Get Value of Nth Char in String in Rust.” Stack Overflow, 1 May 1969,
https://stackoverflow.com/questions/71824249/get-value-of-nth-char-in-string-in-rust.

vilvil 88711 gold badge77 silver badges1212 bronze badges, et al. “How Do I Do a Basic Import/Include of a Function from One Module to Another in Rust 2015?” Stack Overflow, 1 Nov. 1961,
https://stackoverflow.com/questions/26224947/how-do-i-do-a-basic-import-include-of-a-function-from-one-module-to-another-in-r.