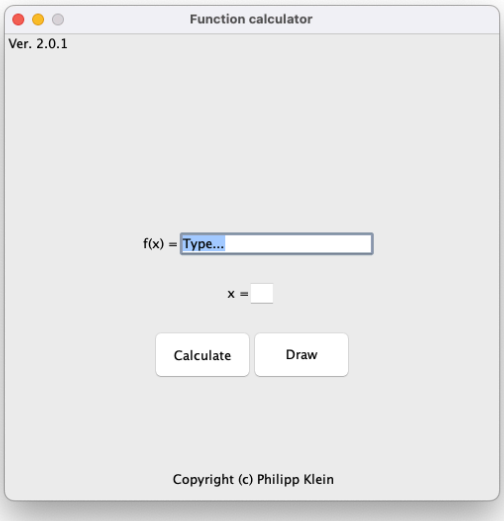
Math function calculator
Version 2.0.1

Contents

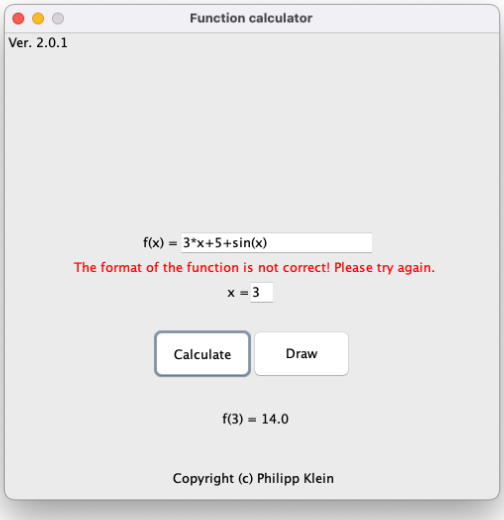
How to use the software?

When you first launch the JAR-File you'll find yourself on the dashboard:



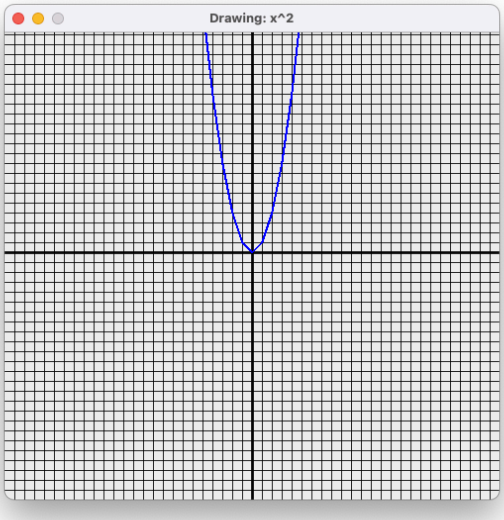
In the text field, currently with the text „Type…" you can type in your function. Not every function is supported e.g. $\sin, \cos, \tan$ are not supported.

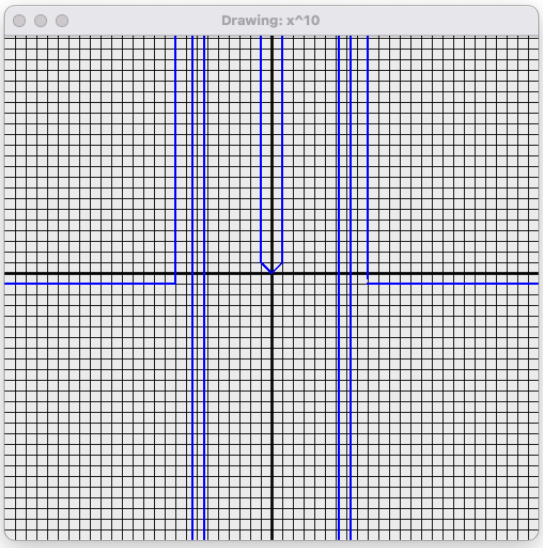
You'll have to figure out, which function works perfectly fine and which do not. If you input a function, that is not supported, you'll get an error message, that'll look like this: ∖/



The function calculator is very limited! In the text field, with the „x=" on the left, you input the value for x. If you mistakenly put a character, left the field blank or a number with a character, you'll receive an error message as well.

At the bottom, you can access two buttons. The button „Calculate", will give you the result of the function to the given x-value. The button „Draw" will try to draw the function into the coordinate system. I'm especially saying ‚trying' because there are functions, which are not drawn correctly.





As you can see, the function $f(x) = x^{10}$ is not drawn correctly. You'll have to figure out, what function works and which does not.

How does the program work?

When you type in a function, it'll be parsed as a String. That String will get prepared for the splitting (Breaking the String at certain points, to fit into an array). If there is a minus in the front of the String or the first slot of the array, it'll get replaced by -1:

```
[-,3,*,x,^,2] > [-1,*,3,*,x,^,2]
```

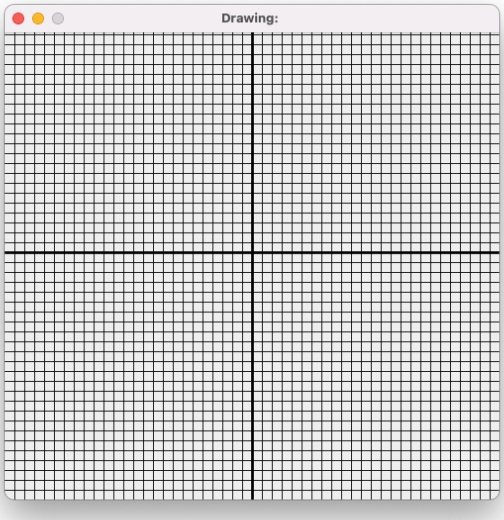
Now the String is ready for the calculations. First, the brackets are calculated until none are left. Inside the brackets and in normal conditions, the operators are executed in the following order, they comply with the mathematical rules: (^) Power-, (*,:) Point- and (+,-) Line-operators. Here's an example:

```
f(x)= „3*x^2+(10*x-4)", be x = 1:

1. String.split() > [3,*,x,^,2,+,(,10,*,x,-,4,)]
2. PutXIntoArray  > [3,*,1,^,2,+,(,10,*,1,-,4,)]
3. CalculateBracketsFirst > [3,*,1,^,2,+,(,10,*,1,-,4,)]
4. ()PowerOperators > [3,*,1,^,2,+,(,10,*,1,-,4,)]
5. ()PointOperators > [3,*,1,^,2,+,(,10,-,4,)]
6. ()LineOperators  > [3,*,1,^,2,+,(,6,)]
7.   RemoveBraces   > [3,*,1,^,2,+,6]
8. PowerOperators   > [3,*,1,+,6]
9. LineOperators    > [3,+,6]
                    > [9]
```

Not shown in the graphic: if an operator is found, because we're in an array, we can take the left and right slots, from the slot, where the operator sits and operate on those two. When finished, the left and right slots will be erased and the slot of the operator will be overwritten with the result of the operation. After that, the array gets shortened, eliminating all blank slots. When the program is completely done, there'll be only one value left in the array, the total result.

When you're drawing the function, the program will do the following: every window has a size of 500x500 pixels:



The problem is, the coordinates in JFrames are differently interpreted e.g. the point (0/0) is at the top left. Therefore the origin has the coordinates (250/250). Every block in this coordinate system has a width of 10 pixels, resulting in a total of 25 blocks per axis.

In JFrames it’s sadly not possible the pass in a Double value when drawing a line. But that is understandable since there are no half or fewer pixels ;) So when the program is drawing the function e.g. x^2, the count of the block (in pixels), gets divided by 10, giving the ‚real‘ value to calculate with:

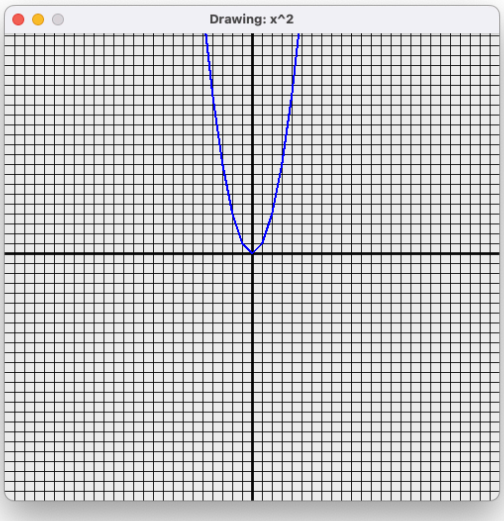
```
Given function: f(x)=x^2

1. Block 10 Pixels --/ 10--> 1 real value → f(1) = 1 --* 10--> 10 Pixels

2. Block 20 Pixels --/ 10--> 2 real value → f(2) = 4 --* 10--> 40 Pixels

3. Block 30 Pixels --/ 10--> 3 real value → f(3) = 9 --* 10--> 90 Pixels
```

To connect the points, the program calculates two results at the same time and then connects the two points. Giving us the graph:



In the current example, the drawing commands would look like this:

```
… > drawLine(f(0),f(1)) > drawLine(f(1),f(2)) > drawLine(f(2),f(3)) > …
```

All this is done via a for-loop which iterates from 0 to 500 (Pixels). As I've stated before on my website, the programs I'm building aren't aiming to be perfect, but they and the feedback from others will help me improve my coding skills. It is just a project I wanted to realize and share with you.