The BaabnqIDE

The BaabnqIDE is an integrated development environment for the Baabnq programming language. This is a small guide on how to use it and how it works behind the scenes.

1. Basic Usage

The screen is divided into two sections, the editor on the left and the console on the right.
The editor and file handling works like any other IDE. The console just acts as an interface to the program running, like a terminal.
Now the fun stuff, press F1 to run the currently opened tab in debug mode.
This however will probably not work, because the IDE doesn't know how to run a program.
The Compiler and Virtual Machine configuration can be found under "Options->Run Config".
If you don't know how the Baabnq system works, just download the lastest version of the Baabnq compiler and replace <path2baabnq> with the location of the compiler.py file:

python "<path2baabnq>" --input {input} --output {output}
python "S1monsAssembly4 Virtual Machine Debug.py" --file {file}

A running program can be kill by pressing Shift + F1 and can be run normally, in "release mode", by press Ctrl + F1. Breakpoints can be set by pressing F2 and are indicated with a red marker, between two line numbers. Breakpoints can be cleared by pressing Shift + F2.
(most options/triggers can be found in the menu too)

2. Other features

- Clear console with Ctrl + 1.
- Refresh editor content under "File->Refresh Editors"
- Zooming in, Ctrl + "+" and out, Ctrl + "-"
- Search in current editor with regex, Ctrl + F
- Editor text completer, Ctrl + Space
- Compiler error check (terminates process queue when error in the compiler output is detected), "Options->Compiler Error Check"
- Jump to error line, "Options->Jump To Error Line"

3. Internals

To achieve live debugging with breakpoints, this IDE has a special version of the S1monsAssembly Virtual Machine, that supports a so called "debug protocol".
When the user sets a breakpoint into a program and runs it, that IDE will insert an inline assembler instruction at the breakpoint that looks like this: asm 'breakpoint 0';

(The 0 acts as a placeholder and could be replaced by any other number)

When the breakpoint instruction is hit, the Virtual Machine will halt and await input on how to deal with the breakpoint. Once the IDE tells the Virtual Machine to resume it will continue running the program.

In addition to that if a program is run in debug mode, the IDE will call the Virtual Machine with the "—debug" flag. This causes that Virtual Machine to send all the data described in the "debug protocol".

All of this is done to avoid changing the compiler itself, which makes developing new version much easier. There is one exception to that, but it's trivial to implement and doesn't require a separate version:

The compiler needs to have a "–MoreInfo" flag, that will make it list the xVarMapper (which is used for variable mapping) at the top of the output assembler file. This feature is needed to tell the IDE, what memory addresses to listen to, to display the variables in the debug window correctly.