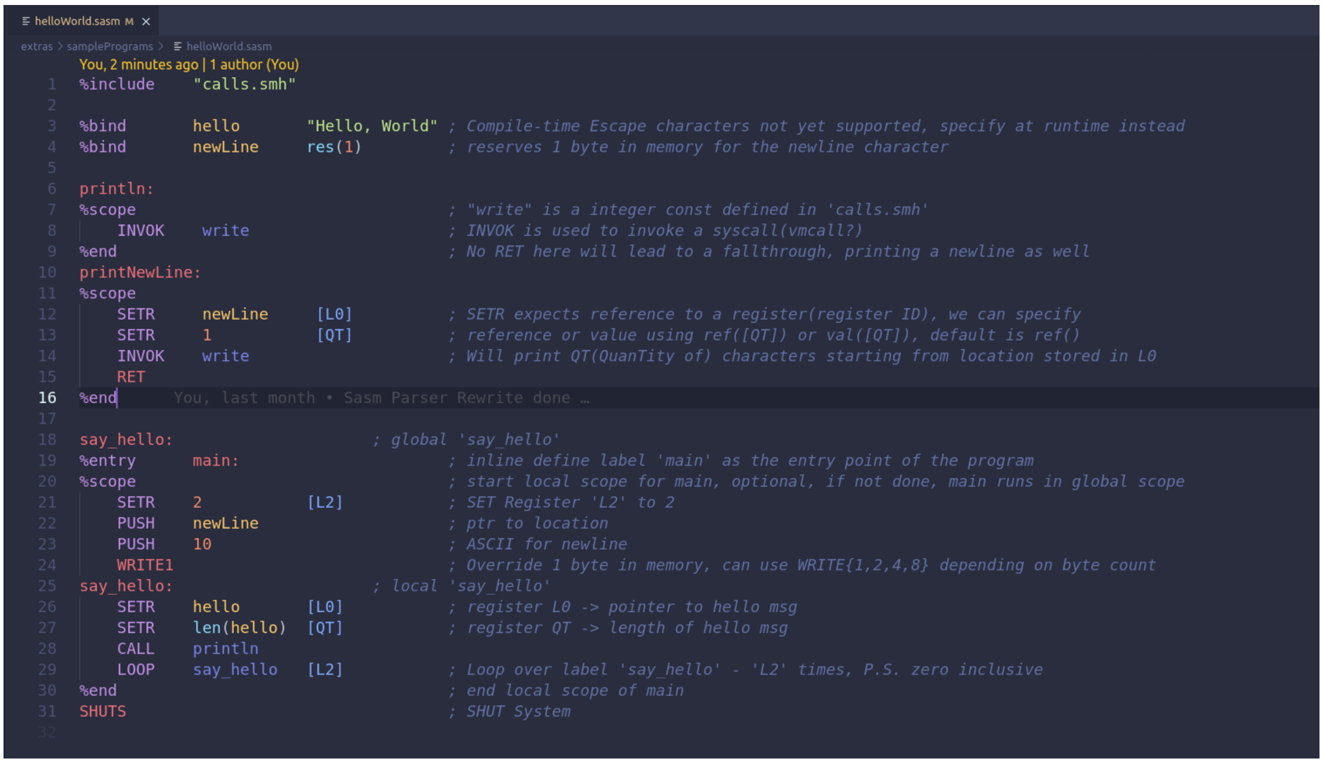
🛠️ Current Features

- ✅ **VS Code syntax highlighter** for SASM
- 🌲 **AST visualizer** for seeing how your SASM code is parsed and compiled
- 🔧 A new programming language called **ORIN** is currently under development. It is being designed to compile directly to SASM.

> If you're interested in compilers, language design, or virtual machines — **contributions are very welcome**!

Syntax Highlighting:


helloWorld.sasm M ×
extras > samplePrograms > helloWorld.sasm
You, 2 minutes ago | 1 author (You)

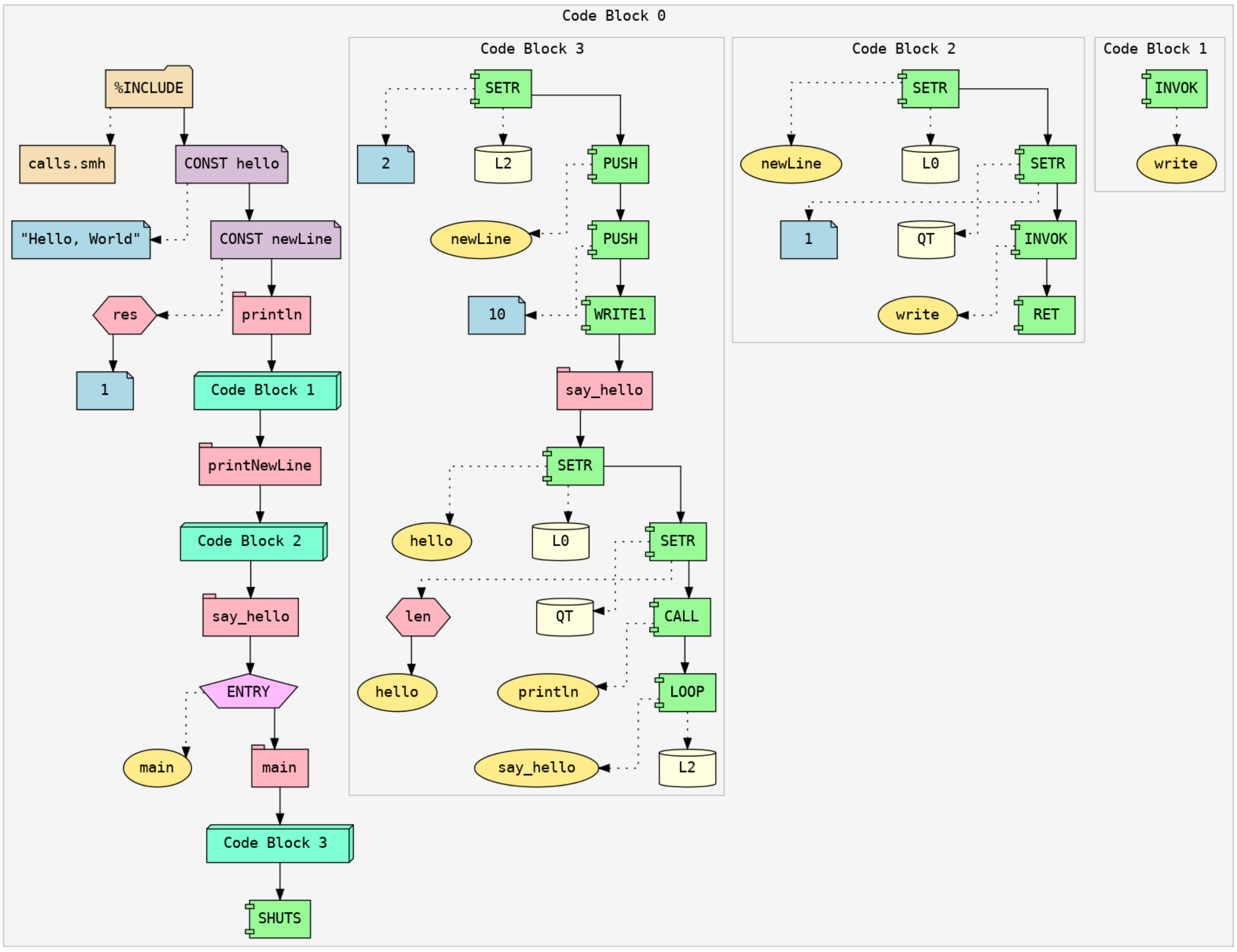
```
%include    "calls.smh"

%bind       hello       "Hello, World" ; Compile-time Escape characters not yet supported, specify at runtime instead
%bind       newLine     res(1)         ; reserves 1 byte in memory for the newline character

println:
%scope                                 ; "write" is a integer const defined in 'calls.smh'
    INVOK    write                     ; INVOK is used to invoke a syscall(vmcall?)
%end                                   ; No RET here will lead to a fallthrough, printing a newline as well
printNewLine:
%scope
    SETR     newLine     [L0]          ; SETR expects reference to a register(register ID), we can specify
    SETR     1           [QT]          ; reference or value using ref([QT]) or val([QT]), default is ref()
    INVOK    write                     ; Will print QT(QuanTity of) characters starting from location stored in L0
    RET
%end        You, last month • Sasm Parser Rewrite done …

say_hello:                     ; global 'say_hello'
%entry      main:                      ; inline define label 'main' as the entry point of the program
%scope                                 ; start local scope for main, optional, if not done, main runs in global scope
    SETR    2           [L2]           ; SET Register 'L2' to 2
    PUSH    newLine                    ; ptr to location
    PUSH    10                         ; ASCII for newline
    WRITE1                             ; Override 1 byte in memory, can use WRITE{1,2,4,8} depending on byte count
say_hello:                     ; local 'say_hello'
    SETR    hello       [L0]           ; register L0 -> pointer to hello msg
    SETR    len(hello)  [QT]           ; register QT -> length of hello msg
    CALL    println
    LOOP    say_hello   [L2]           ; Loop over label 'say_hello' - 'L2' times, P.S. zero inclusive
%end                                   ; end local scope of main
SHUTS                                  ; SHUT System
```

AST:



!!! info "Local/Global Scopes"

```
Each Code Block in the visualized AST represents a Scope, Block 0 being
global scope.
```