Jonathan Ke

Kosbie

Fundamentals of Programming Section A

20 November 2019

Final Project Proposal

Abstract:

This project will be an interactive IDE that will recursively parse and translate a subset of Java code into similarly performing Python code. The application allows the user to convert and then run Java code on the Python Virtual Machine. It will also include numerous functions common to IDEs and code editors like Visual Studio, including error highlighting, terminal outputs, file saving, and autoindent. The application will also include functions to generate and visualize parse trees for Java code inputs and Python translations with a minigame in which the user writes legal Java code that generates a specific parse tree to pass a level.

Competitive Analysis:

While most parsers, including the ones that Java and Python use, implement some form of look-ahead parsing for fast parse and compile times, this application does not implement such a robust parsing algorithm. This application instead uses a simpler recursive descent algorithm to tokenize and parse Java code. The application also relies on the Python Virtual Machine to run compiled code, so in essence, this program is a weak implementation of functions inIDLE and VSCode.

The primary differences in this application is its focus on translating Java code to Python code. Online applications converting Java to Python are often unreliable and can only translate a small set of Java. This project uses source code to source code conversion to write Java in Python and has an easily expandable library for a more robust conversion. The code also includes self modifying grammar functions that can speed up conversion by saving and tracking grammar appearances, providing commonly used grammar structures priority in the parsing selection process. An implementation of syntax tree visualizations is also unique to this project.

Structural Analysis:

* IDE user interface for code editing in both Java and Python
* Connection to Python 3 terminal for running code
* Tokenizer and parser for parsing Java code
* Minigame for visualizing syntax trees from parsed Java code
* Syntax tree visualizations for parsed Java code
* Numerous other widgets and functions commonly found in code editors

Algorithmic Plan:

* Tokenizer applies the abstract Java tokenizing algorithm found at CMU 15200 website (<https://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/tokens/lecture.html>). This is already finished.
* Parser is a recursive descent algorithm using recursive backtracking to parse a list of tokens into a parse tree. This is already finished.
* Error tracking that highlights the depth of parsing in case of an error. This is finished.
* Strong smart compile function to speed up parsing by saving common grammar parses and sorting them into the root grammar used to parse Java. A lightweight version of this is implemented, but a full version is not finished.
* Recursive translator that navigates a parse tree and converts nodes to legal Python grammar, including numerous functionalities that allow for multiple node manipulations and altering the parse tree during translation. This is completed. Further additions to the manually written translation library can be made to expand the functionality of conversions.
* Recursive parse tree debugger and recursive Python syntax tree visualizer for printing out legal Python syntax and grammar. This is completed.

Timeline Plan:

By TP2:

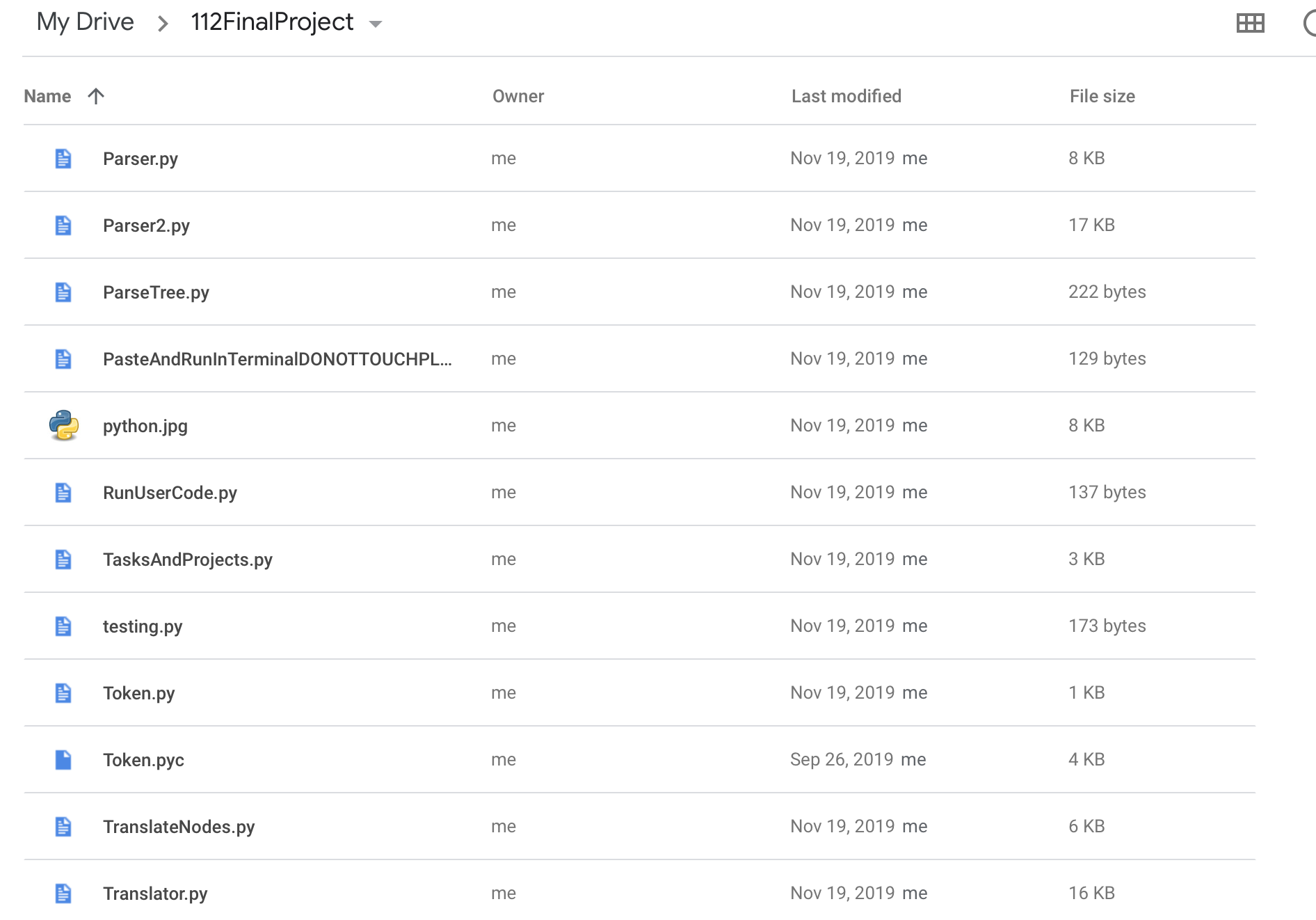
Create visualization and minigame GUIs, likely using cmu\_112\_graphics with the MochaPython application. Implement functions and tools into MochaPython application like autotabbing, file saving, error highlighting, maybe syntax highlighting, and improved GUI visual appearance with different color schemes.

By TP3:

Expanded translation library to cover most of common Java functions, not including imports. A primary focus on translating different Java method syntax, Java static method versus non-static method syntax, expanded array library, field and attribute declaration syntax, and Java string methods to Python methods. An expanded smart compile function will also be researched and implemented.

Version Control:

Project code is regularly uploaded to Google Drive for cloud backups every one to two days of working.

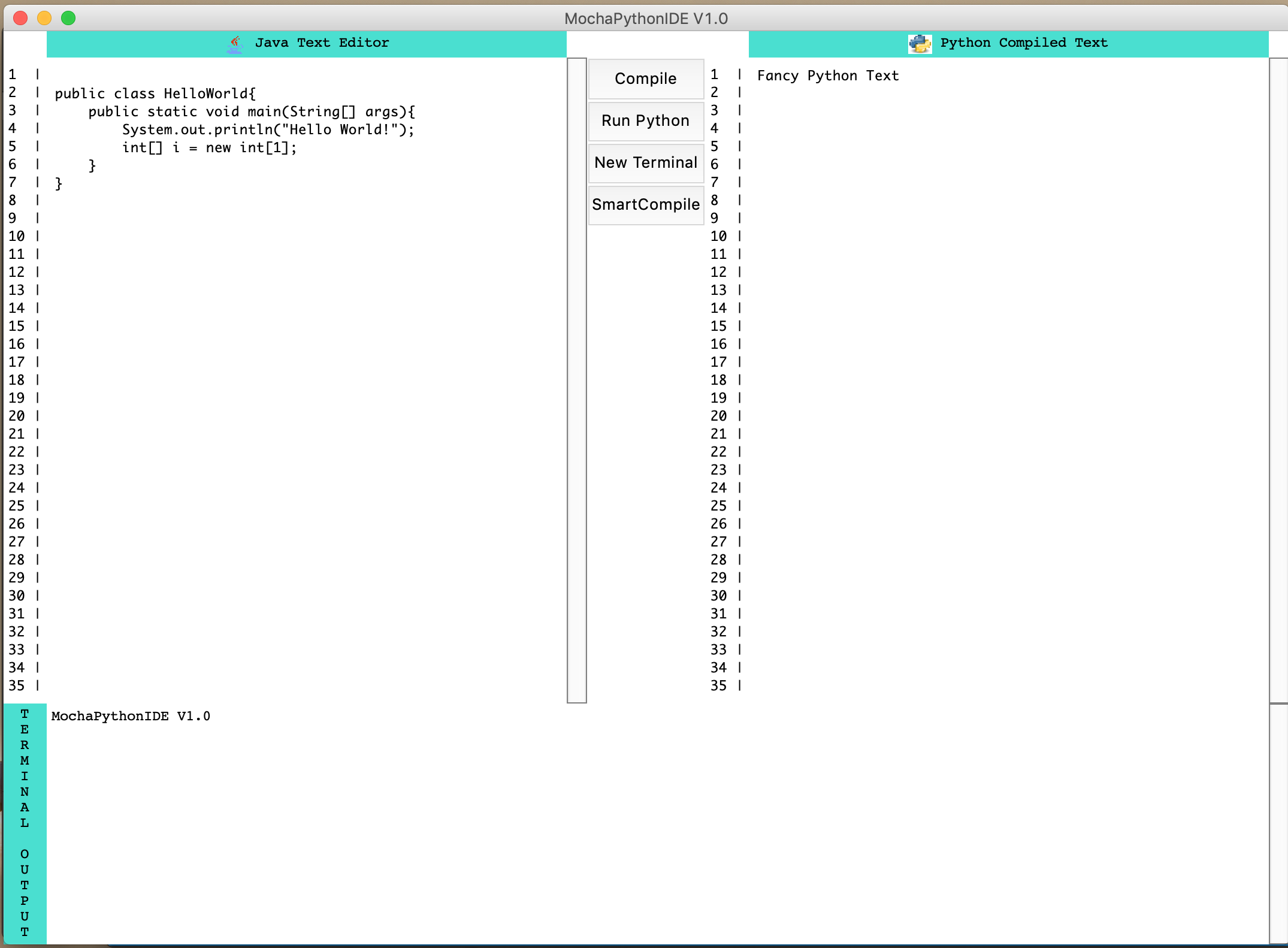


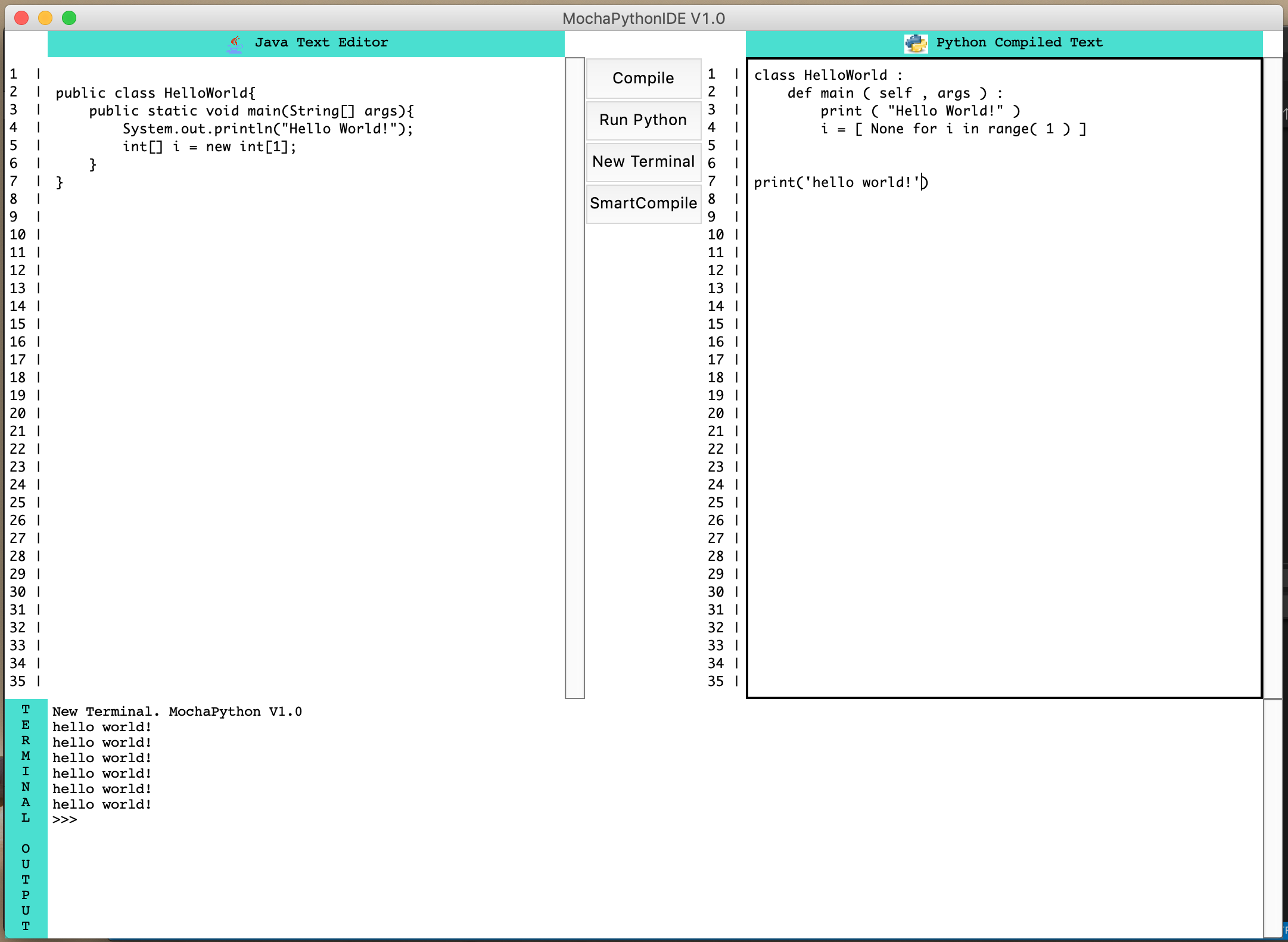
Module List:

* Tkinter
* CMU 112 Graphics
* PIL

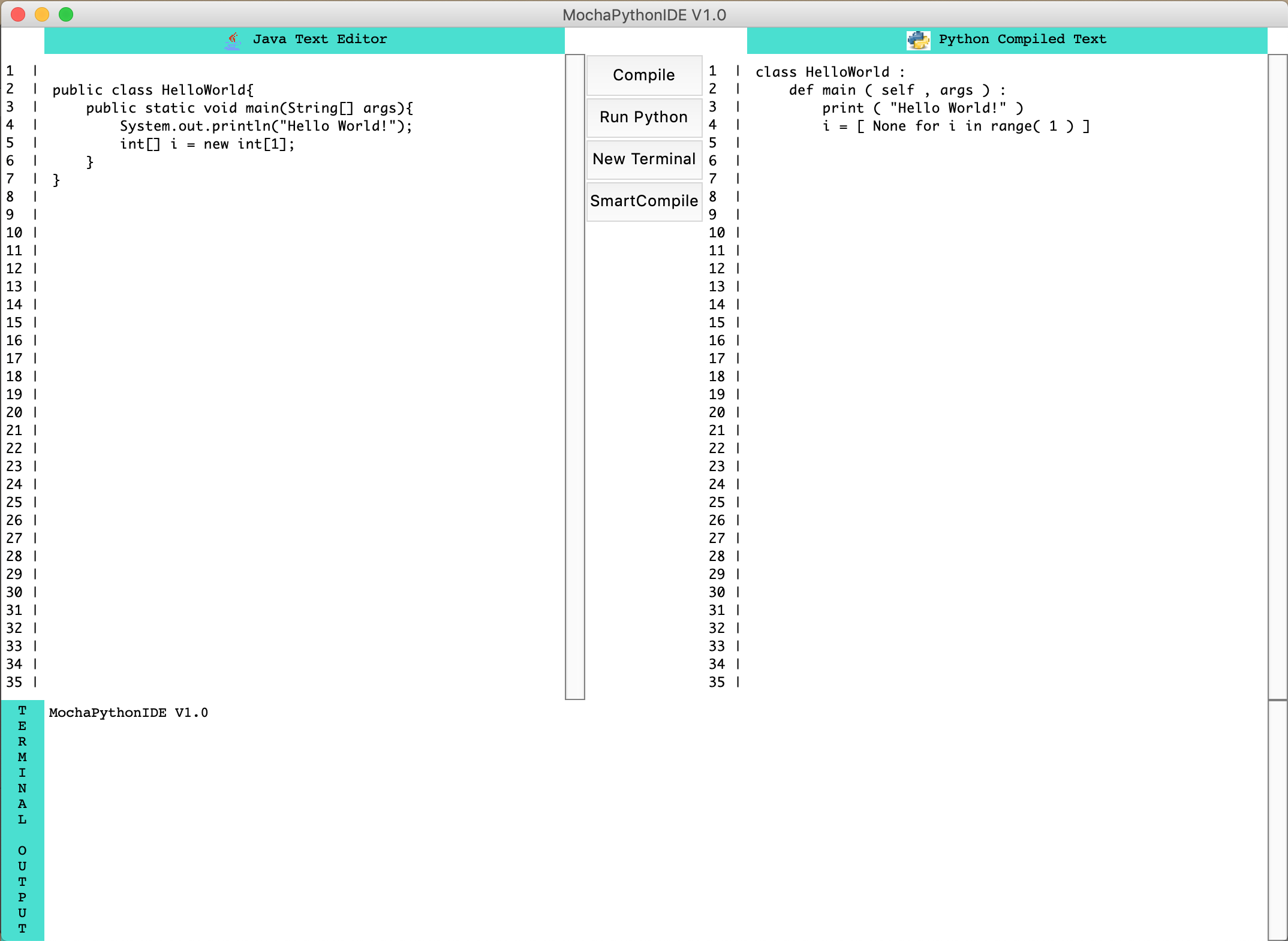
Storyboard:

Easy to read GUI. Allow for both Java and Python text editing.

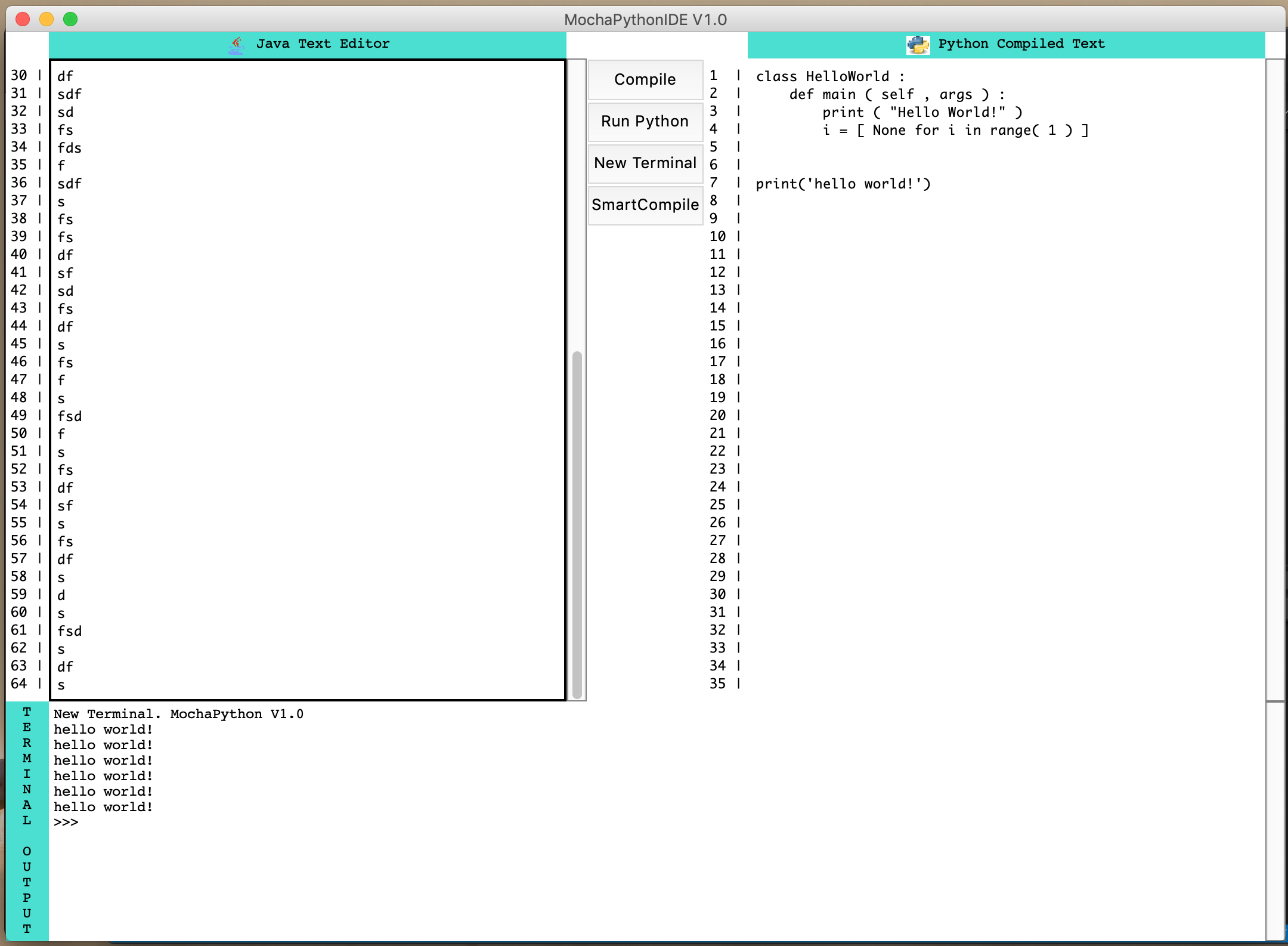




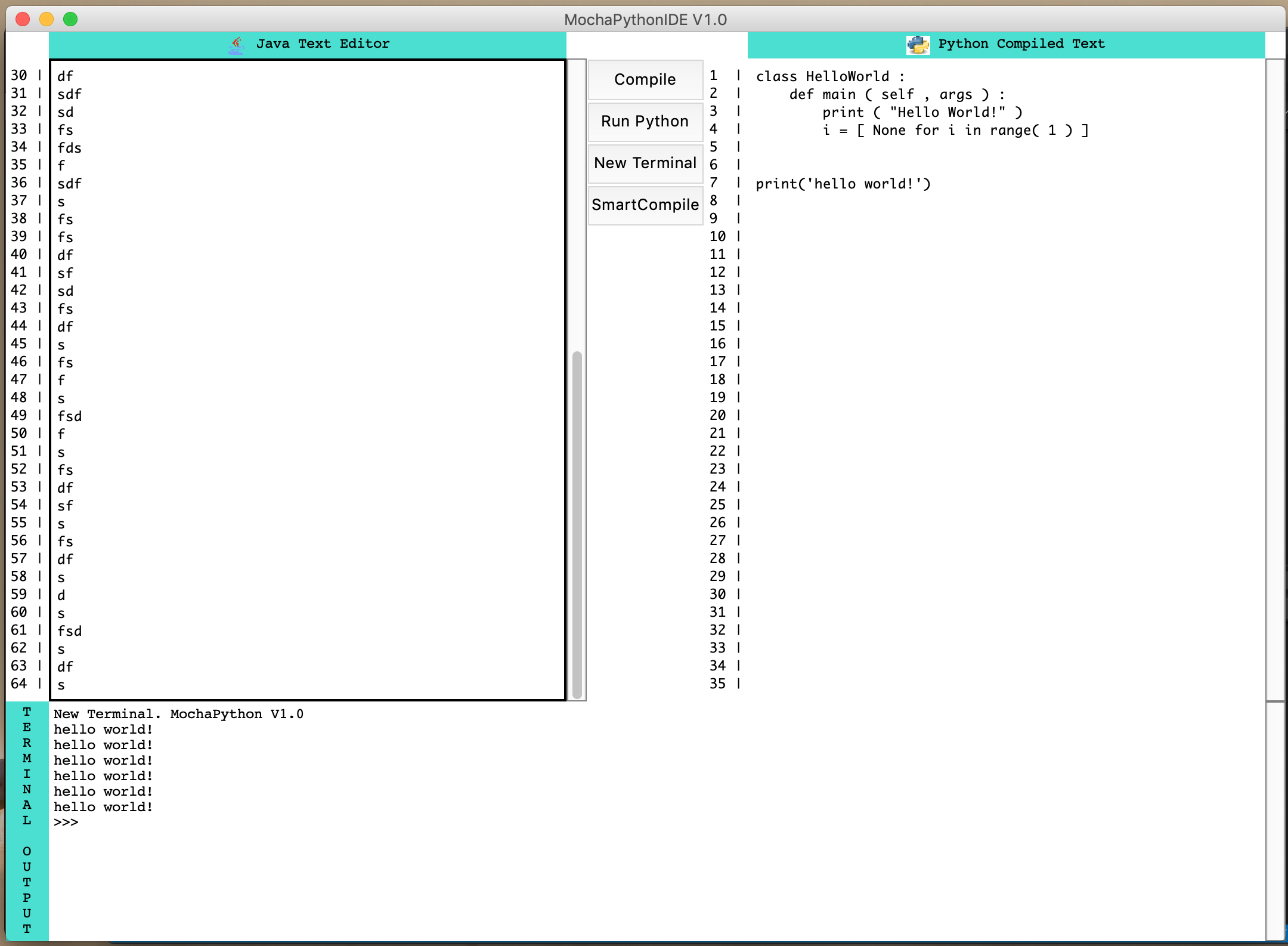
Java program compiles into Python with press of a button!



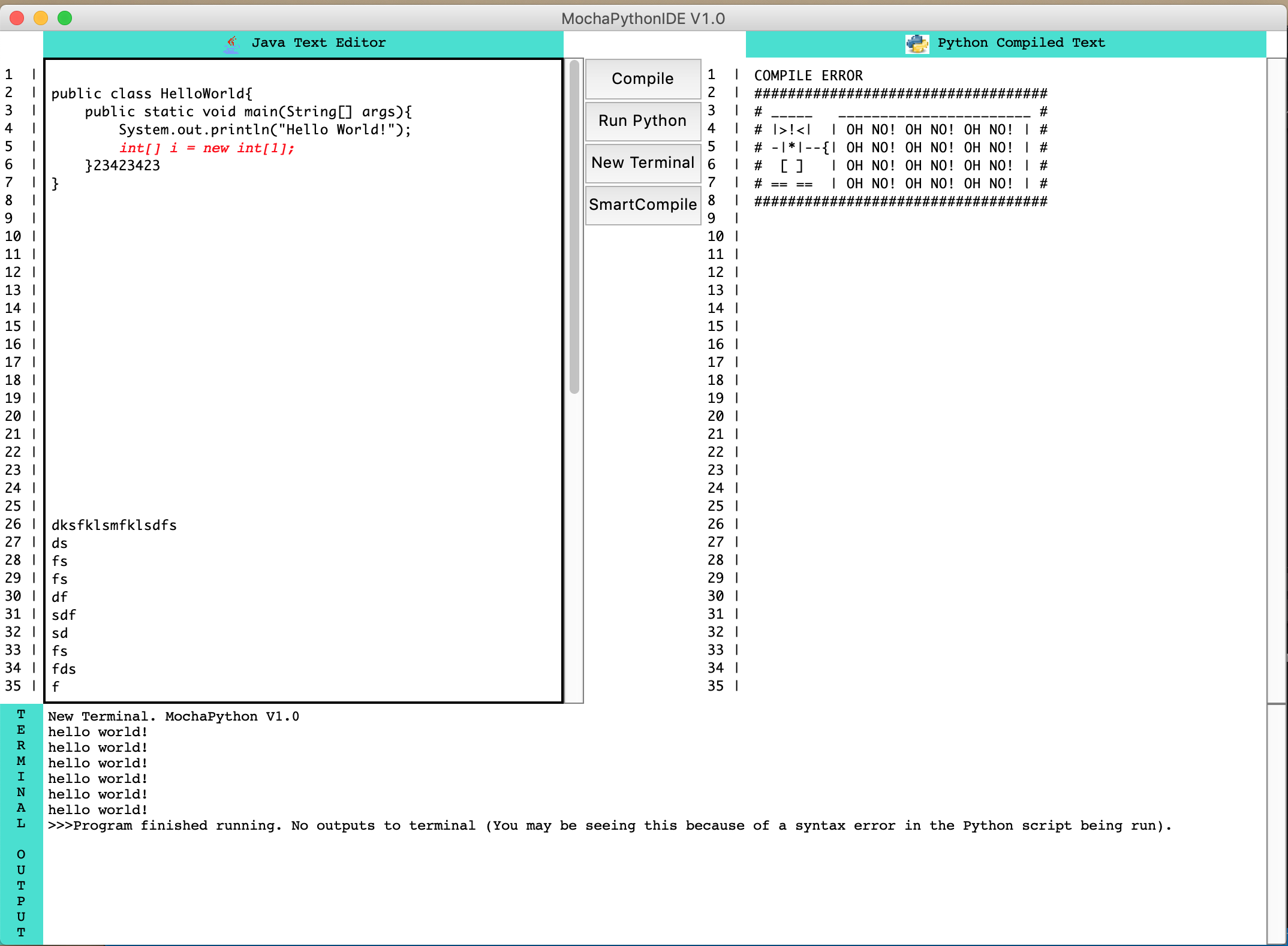
Scrollable text editor and you can edit both the Java and Python sections before running code.



Run Python code right in the application.



If there is an error during compile, Python window will display error message and editor will highlight the furthest token it managed to parse successfully. Simple error matching!



Saves and tracks commonly used grammar definitions for faster parsing. Arrows point to appearances of that grammar in all smart-compiled runs.



TP2 Update:

* Have not created game UI. Completed visualization tool.
* Created a zoom and dilation function (It’s customizable!) using bijection convergence for visualization tool.
* Minor improvements to parser function. Want to find simple ideas to add faster parsing if possible.

TP3 Update:

* Integrated visualization tool with main interface. Completed functioning minigame accessible from the main interface.
* Created syntax highlighter for code editor.
* Minor bug fixes in scrollbar and line numbers.
* Highlights current line the user is on.
* Changed coloring schemes of code editor and visualization tool.
* Created a load and save widget in main interface.
* Improvements to translation library and parsing algorithm.
* Updated translation library to translate functions, hashCode, toString, static fields, and multiple method/function headers.
* Error highlighting when JavaCode cannot be parsed.