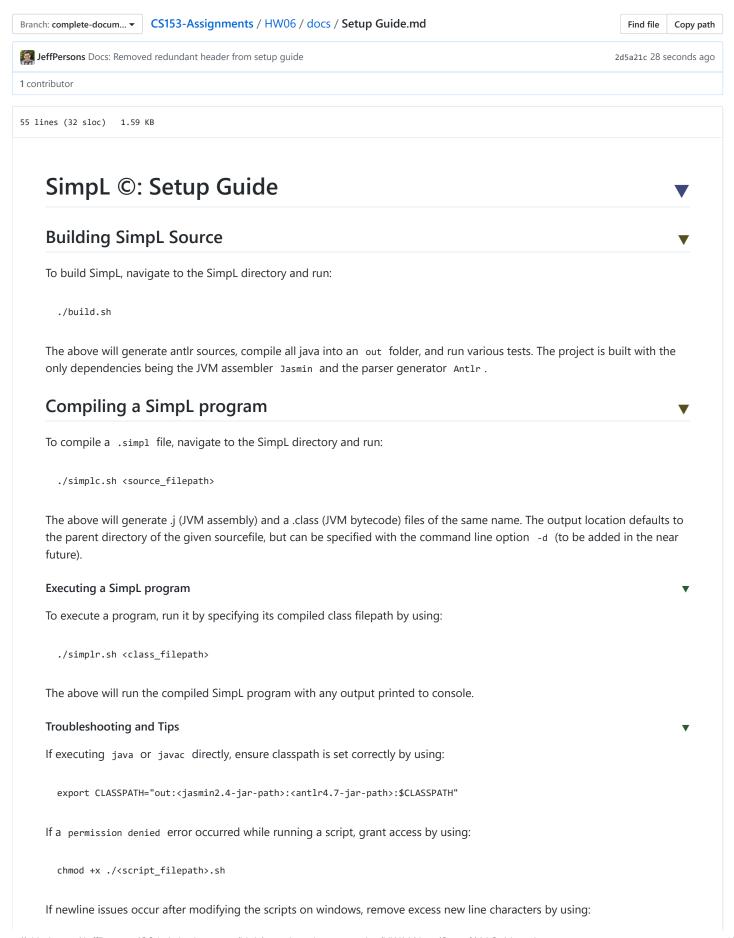
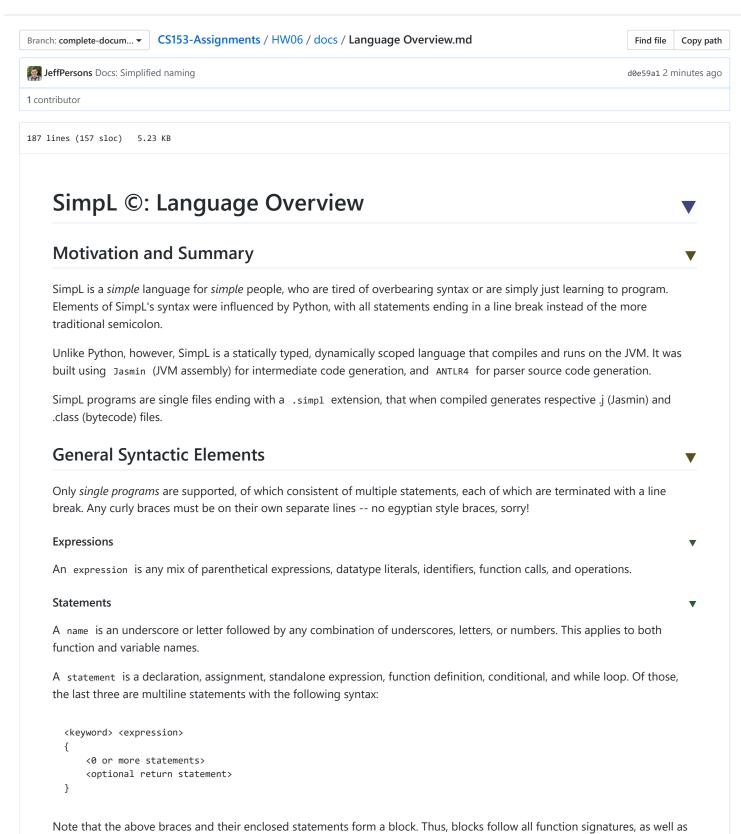
☐ JeffPersons / CS153-Assignments | Private



```
sed -i 's/\r$//' ./<script_filepath>
Delete any generated jasmin and class files by using (optionally limit depth by adding -maxdepth 1):
```

 $\label{lem:continuity} \mbox{find } \mbox{coutput_directorypath} - \mbox{regex ".*\.\(j\|\class\)" - type f - delete}$

☐ JeffPersons / CS153-Assignments Private



all while, if, else if, else conditions.

Comments

Any comment is ignored by the SimpL parser, along with tabs and excess newlines.

Single line comments have the following syntax:

```
## <some single line comment>
```

Multiline comments have the following syntax:

```
##
<some multiline comment>
##
```

Datatypes

lacksquare

Number

An integer or decimal number (internally stored as a double), with the following syntax for literals:

```
<1 or more digits> <1 or more digits> .<1 or more digits>
```

Text

•

A character sequence (internally stored as a *String*), with the following syntax for literals:

```
'<0 more characters>'
```

Note that quotes and slashes can be escaped with a slash (eg ' \).

Boolean

▼

A true or false token (internally stored as a boolean), with the following syntax for literals:

True False

Operators



Support for parenthetical, arithmetic, boolean, comparison operations, ordered from high to low precedence:

Operator Precedence					
	order	operator		meaning	
	0	()		parenthesis	
	1	^		exponentiation	
	2	* /		multiply and divide	
	3	+ -		add and subtract	
	4	< > <= >=		comparison	
	5	==!=	(equality and inequality	
	6	not		logical negation	
	7	and		logical conjunction	
	8	or		logical disjunction	
	8	=		assignment	

Equality, parenthesis, and assignment operators apply to *any* expression and *any* datatype. Comparison and arithmetic operators apply only to <code>Numbers</code> . Logical operators apply only to <code>Booleans</code> .

Variables, Declarations, and Assignments



Support for variables is restricted to datatypes Number and Text . Variables can declared with or without an initial value, as follows:

```
<datatype> <name> = <expression> <datatype> <name>
```

Control Flow

Syntax for conditionals is as follows:

```
if <expression>
{
      <0 or more statements>
}
elif <expression>
{
      <0 or more statements>
}
else
{
      <0 or more statements>
```

Syntax for loops is as follows:

```
while <expression>
{
     <0 or more statements>
}
```

Functions

Function Definitions

A function definition consisting of a signature and a body, with the following syntax:

```
<void or datatype> <name>(<parameter list>)
{
    <0 or more statements>
}
```

Function Calls

A function defined within the current scope is invoked, with the following syntax:

```
<name>(<argument list>)
```

Above, <argument list> is defined a comma separated list of one or more <name> . When calling functions, arguments are passed by value, and must correspond to the parameters specified in the function definition.

Builtin Functions

Predefined functions print and println are available, and take any number of arguments. Just like in Java, print writes the expression to standard out,

Error Handling Errors and exceptions encountered during compilation are written to standard error. Type Checking Errors are raised to ensure operators are between appropriate type(s), as specified in the Operators section from before. Error Recovery Like most compilers, Simpl will continue parsing the rest of the file even if an error occurs.