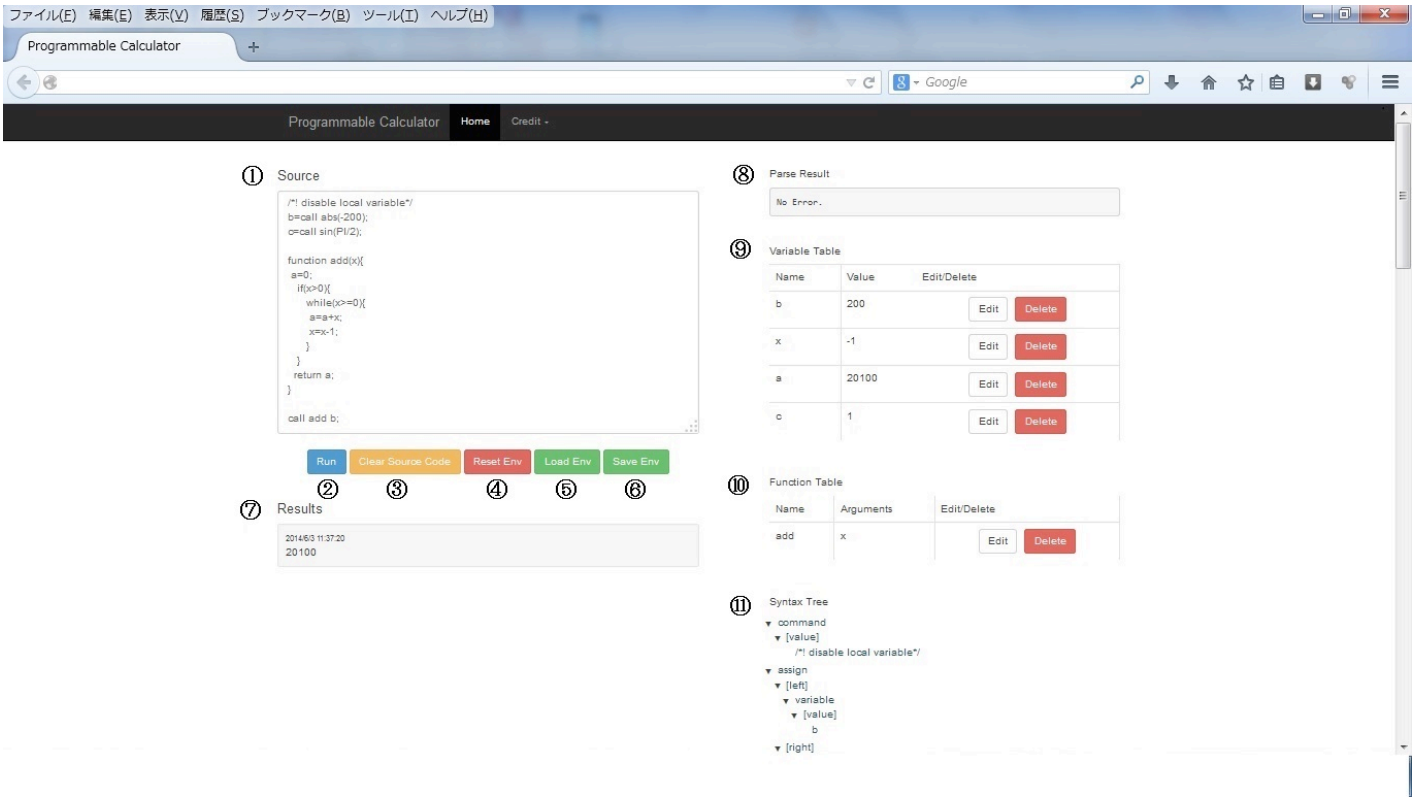
Programmable calculator ユーザーマニュアル

１：操作説明について



画面説明

① ソースコード入力画面

② 実行ボタン

③ ソースコード入力画面クリア

④ 環境リセットボタン

⑤ 環境読み込みボタン

⑥ 環境セーブボタン

⑦ 実行結果表示画面

⑧ 構文エラー表示画面

⑨ 変数状態表示画面

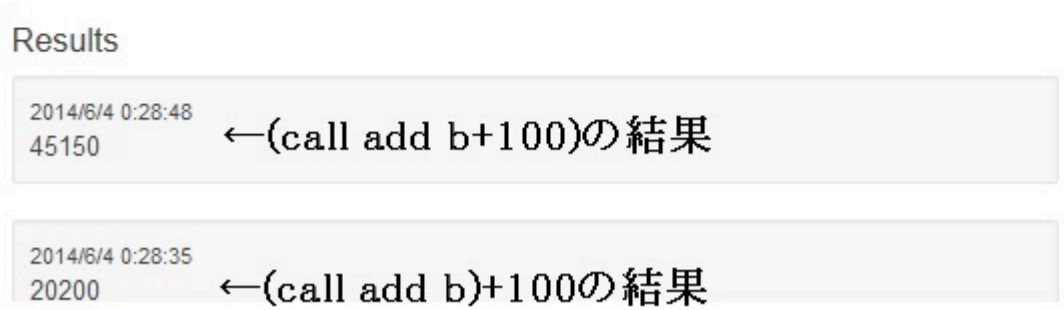
⑩ 関数表示画面

⑪ 構文木表示画面

①②⑦ソースコード入力、実行について

Programmable calculator では 標準的な計算機能に加え、sin,cos などの関数の計算や自分で自作した関数を扱って計算することもできます。扱うことの出来る関数や実装されている定数は後述します。

ソースコードの例



ソースコードの書き方

C 言語でプログラムする時のように 1 行ごとにセミコロン(;)を入れてください。

このプログラムでは、function 関数名(引数 1,引数 2,…){ 命令文 }という構文によって関数を定義することができます。

このプログラムでは、if 文と while 文を実装していますので、ある程度自由に関数を作ることが出来ます。関数は C 言語のように書くことで作れます。ただし、x--などは実装していないので、while ループを作る場合などは x=x-1;のようにしてください。

また、自作関数に限らず、関数を使用する場合には call 関数名 引数 1 引数 2 …という形で使用できます。なので関数の結果に足し算したい、という時は(call 関数名 引数)+足したい数という風にしてください。

また、デフォルトではスコープ機能を有効になっていますが、文の先頭に

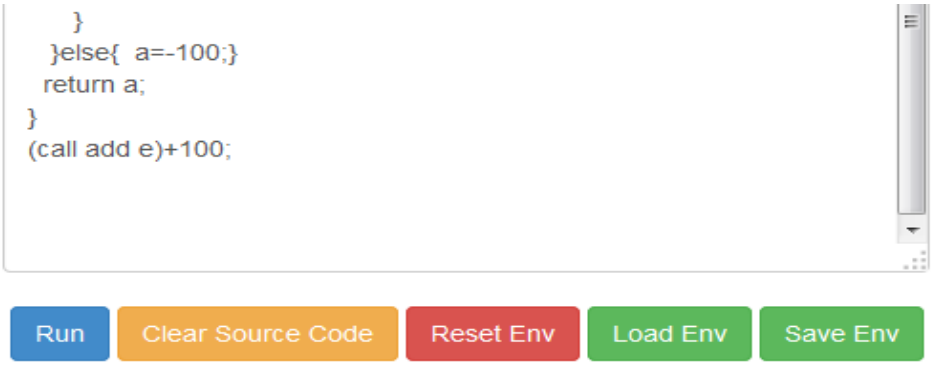
/*! enable scope */を書くことでスコープを有効にでき

/*! disable scope */を書くことでスコープを無効にできます

Run を押すと、最後に書かれた数字を出力する命令の行の結果を Result に表示します。また、同時に variable table ,Function table を更新します。

variable table ,Function table に定義された変数、関数はソースコードを消した状態でも、④の Reset Env を押

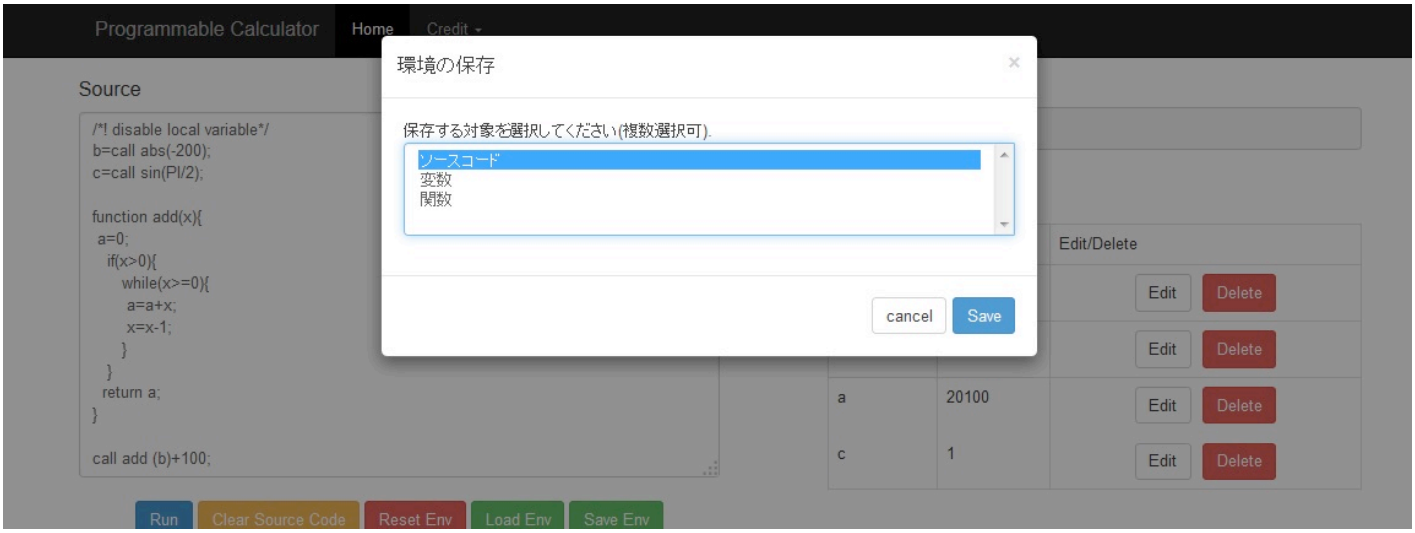
して環境をクリアにしたり、Delete ボタンを押して削除しない限りソースコード画面で定義せずに使用することができます。
Result は最新の結果を一番上に表示します。画像の例では、1 から x までを足し合わせた数を出力するプログラムですが、b=200 で、(call add b+100)と(call add b)+100 では結果が違うことがわかります。
たとえば、max という関数に対して、a, b+10, 3!, 2^6 を渡し、その結果に 10 を足したい場合には
call max a (b+10) 3! (2^6) + 10 ではなく、(call max a (b+10) 3! (2^6)) + 10 とすると、うまくいきます。
このように、関数に引き渡す引数までを()で囲ったほうがより自分の思ったような挙動になります。



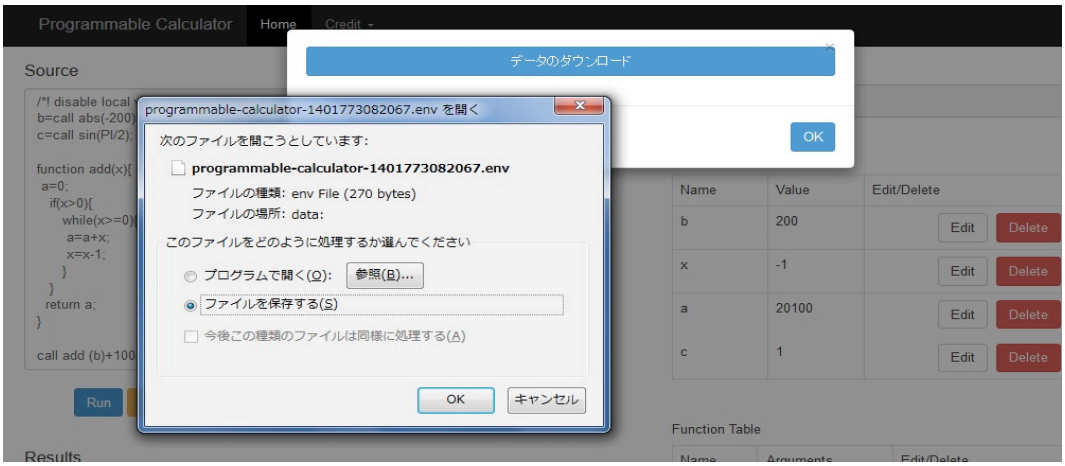
さらに、エラーが起きた時にエラーの種類と何行目にそのエラーが起きたのか、またその該当する行を表示することより、作業がより捗りやすくなりました。

扱うことのできる関数
abs() ,acos(), acosh(),asin() ,asinh() ,atan() ,atanh(),atan2(),cbrt(),ceil(),clz32()
cos(),cosh(),exp(),expml(),floor(),fround(),hypot(),imul(),log() ,logip(),log10(),log2()
min(),max(),pow(),random(),round(),sign(),sin() ,sinh() ,sqrt(),tan(),tanh(),trunc()
実装されている定数
E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2, EPSILON, MAX_VALUE, MIN_VALUE, NEGATIVE_INFINITY, POSITIVE_INFINITY
詳しい説明は
https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Math#Properties
にのっています、が、NOW については現在の UNIX 時を返します。

⑤⑥環境のセーブ、ロードについて



Save Env を 押すと今まで作ったソース、variable table 、Function table などの環境を保存できます。選択したあと save を押すとデータのダウンロードという画面になり、そのボタンを押すと.env ファイルが保存できます。

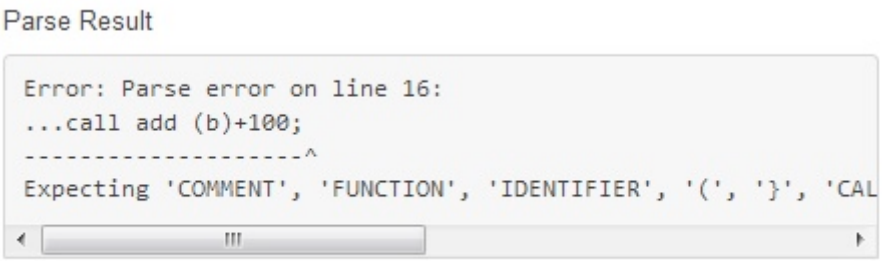


環境のロードについて

保存した env ファイルを選択またはドラック&ドロップすることで環境を引き継ぐことが出来ます。
実は、Road Env ボタンを押さなくても、直接 Programmable calculator のページにドラック&ドロップすること

で環境を引き継ぐことが出来ます。

⑧エラー画面について

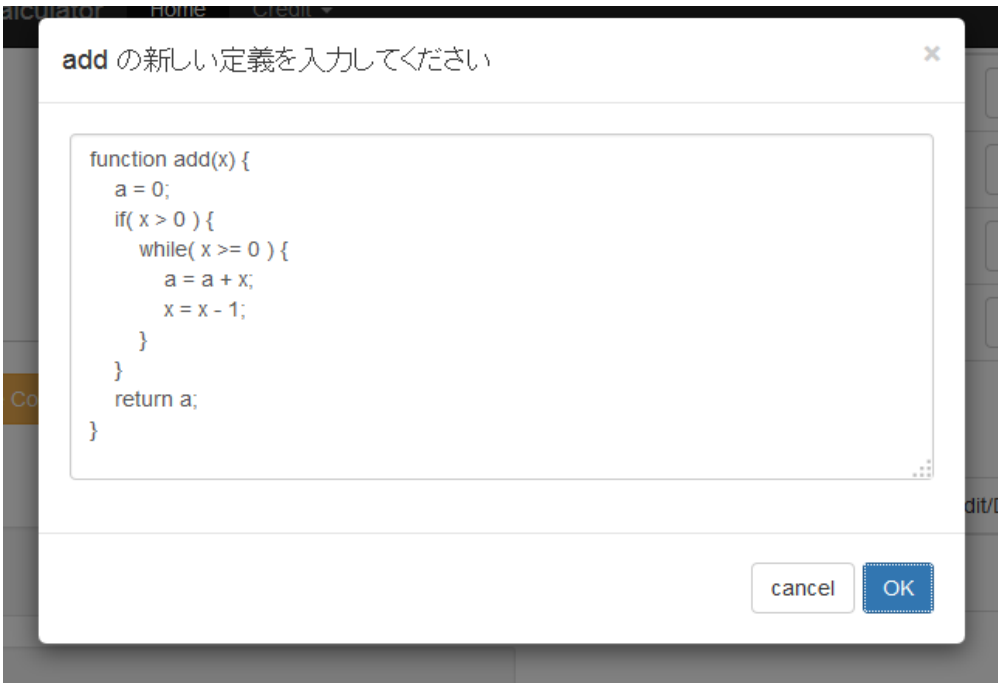


ソースコード画面で、構文エラー(}が足りないなど)によりエラーが発生した場合にそのことを知らせてくれる画面です。

⑨⑩variable table , Function table について

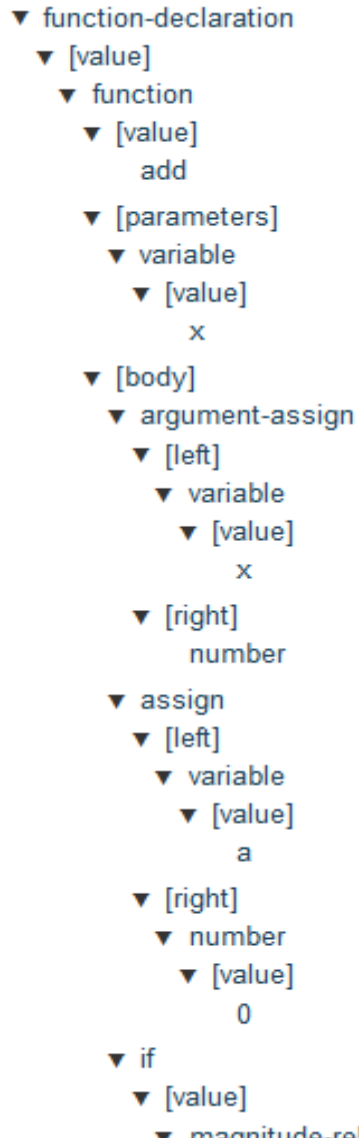
ソースコードを入力し、Run を押すと定義された変数、関数をそれぞれの table に登録します。variable table ,Function table に定義された変数、関数はソースコードを消した状態でも、④の Reset Env を押して環境をクリアにしたり、Delete ボタンを押して削除しない限りソースコード画面で定義せずに使用することができます。　変数/関数の内容を変更したい場合は Edit ボタンを押すことで変更できます。削除したい場合は Delete ボタンを押すことで削除できます。

Function table の add 関数に対して、Edit ボタンを押した時の画面



また、Function table 上では、関数名にマウスを当てると詳細が表示される機能も搭載しています。

⑪構文木表示画面について



構文木表示画面では、文字通り入力したソースコードを構文解析した結果を表示します。とても長いので画像は一部分ですが、これは function 部分の構文木の一部です。どのように構文解析がなされているかがわかります。

２：使用した構文について

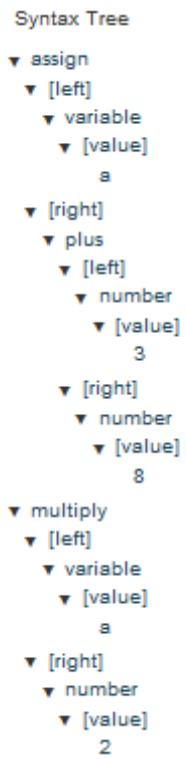
入力できる数式の構文（EBNF による構文規則）

```
number = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
alphabet = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
| "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" |
"B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P"
| "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "_";
real_number = number [ "." number { number } ];
relational_operator = "==" | "!=" | ">=" | "<=" | ">" | "<";
logical_operator = "&&" | "||";
arith_operator = "+" | "-" | "/" | "*" | "%" | "^";
program = { program_source };
program_source = { program_body };
program_body = { block };
block = if_block | while_block | function_declaration | line | comment;
line = ( statement | expression ) [ ";" ];
statement = variable_assignment | return_statement;
function_declaration = "function" function_identifier "(" [ function_parameter ] "){"
program_body "}";
function_identifier = identifier;
function_parameter = variable_identifier { "," variable_identifier };
function_call = "call " function_identifier { " " expression };
variable_assignment = variable_identifier "=" expression;
variable_identifier = identifier;
if_block = if_true_block [ if_else_block ];
if_true_block = "if(" boolean_expression "){" program_body "}";
if_else_block = "else{" program_body "}";
while_block = "while(" boolean_expression "){" program_body "}";
return_statement = "return" expression;
expression = simple_expression | boolean_expression;
```

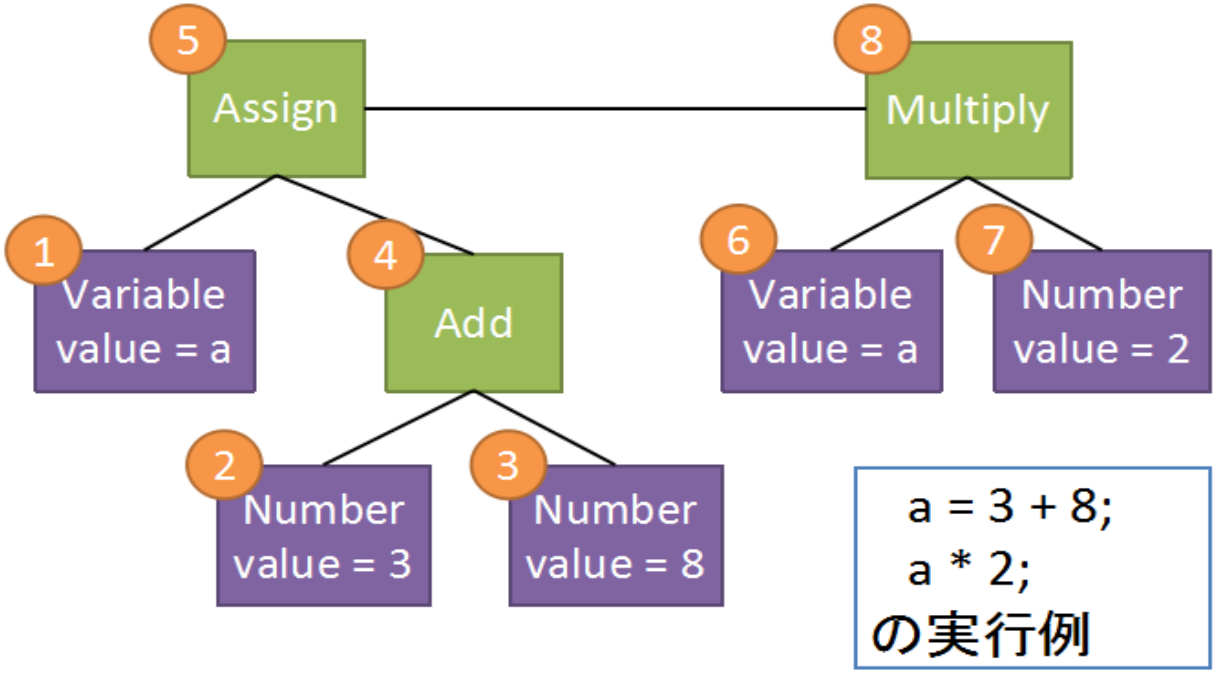
```
boolean_expression = binary_boolean_expression { logical_operator binary_boolean_expression } | "(" boolean_expression ")";
binary_boolean_expression = simple_expression { relational_operator simple_expression } | "(" binary_boolean_expression ")";
simple_expression = primary | "-" simple_expression | simple_expression "!" | simple_expression arith_operator simple_expression | "(" simple_expression ")";
primary = function_call | variable_identifier | real_number;
identifier = alphabet { alphabet | number };
comment = "/*" { alphabet | number } "*/" | "//" { alphabet | number };
```

構文的に正しい数式の意味（表している値の意味規則）

例えば、a=3+8; a*2の計算の場合、



このような構文木が生成されます。この図を見ればわかるとおり、assign の項で最初に3 plus 8 が処理され、a に代入されます。　次に a multiply 2を処理し答えが出力されます。



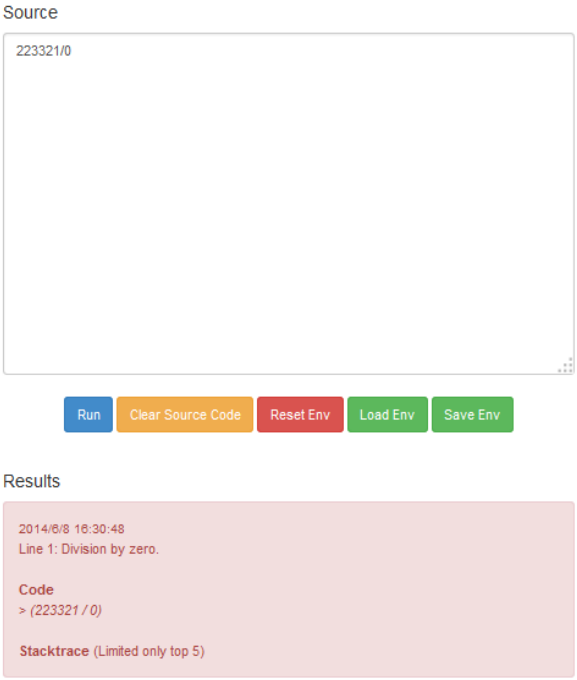
構文誤りに対してどんな誤り表示を出すか

構文エラーの場合、 Error: Parse error on line エラーの行:が表示され、その下に本来期待される文字や記号が表示されます。例えば、

このように、2行目に)が足りない場合、Parse Result の画面にその旨が表示されます。詳しい構文エラーに関しては、ソフトウェア検査書で網羅されていますのでそちらをご覧ください。

実行時に生じうる誤りに対する動作

構文的には誤りがないが実行時にはエラーが出る場合、Result 画面にそのエラーが表示されます。例えば0で割っている、という場合には画像のように



と、Line 1: Division by zero.という風にエラーがでます。他にも定義してない関数を使用した、定義してない変数を使用した。などの場合もありますが、そちらもソフトウェア検査書のほうで網羅しています。