

Catscript Documentation

Catscript Program:

Catscript program as defined in grammar:

```
catscript_program = { program_statement };
```

Program Statement:

Catscript program statement as defined in grammar:

```
program_statement = statement |  
                  function_declaration;
```

A Catscript program statement can be either a statement or a function declaration. Both statement and function definition are documented further in this documentation.

Statement:

Catscript statement as defined in grammar:

```
statement = for_statement |  
            if_statement |  
            print_statement |  
            variable_statement |  
            assignment_statement |  
            function_call_statement;
```

A Catscript statement is a general form of subtype statements. These subtypes consist of: for, if, print, variable, assignment, and function call statements.

For Statement:

Catscript for statement as defined in grammar:

```
for_statement = 'for', '(', IDENTIFIER, 'in', expression ')',  
'{', { statement }, '}';
```

The Catscript for statement begins with the 'for' keyword indicating the start of the for loop. Within a set of required left and right parentheses an identifier is used as the name of a variable which is to eventually be each piece of data contained within something. That is where the 'in' keyword comes into play providing the information that the for loop will run for each instance within some expression. In more code oriented terms these pieces of the Catscript for loop are worded as: for some variable in expression, do something. The something mentioned earlier is the statement contained inside of two required curly braces, a closing brace and opening brace.

Example of a simple Catscript for statement:

```
int x = 5;
for (i in x)
{
    print(i);
}
```

If Statement:

Catscript if statement as defined in grammar:

```
if_statement = 'if', '(', expression, ')',
'{' , { statement } ,
'}' [ 'else', ( if_statement | '{', { statement }, '}' ) ];
```

Example of a simple Catscript if statement:

```
int x = 1;
if (x > 0)
{
    print(x);
}
```

Print Statement:

Catscript print statement as defined in grammar:

```
print_statement = 'print', '(', expression, ')'
```

The Catscript print statement uses the keyword 'print' as the beginning of the statement. It is then followed by a set of opening and closing parentheses. Inside these parentheses goes the input or in other words what should be "printed" as output to the console.

Examples of simple Catscript print statements:

```
int x = 1;
int y = 2;
print(x);
print(y);
```

Variable Statement:

Catscript variable statement as defined in grammar:

```
variable_statement = 'var', IDENTIFIER,
[':', type_expression, ] '=', expression;
```

The Catscript variable statement starts with the keyword 'var' followed by an identifier serving as the name of the variable. The variable's value can then be set with or without specifying type. A Catscript variable will infer its type from the value it is set to if no type is provided.

Simple example of Catscript variable statement:

```
var x = 10  
iar x : int = 10
```

Function Call Statement:

Catscript function call statement as defined in grammar:

```
function_call_statement = function_call;
```

The Catscript function call statement directly translates to a function call.

Example of a simple Catscript function call:

```
name(input1, input2)
```

Assignment Statement:

Catscript assignment statement as defined in grammar:

```
assignment_statement = IDENTIFIER, '=', expression;
```

The assignment statement takes a value and assigns it to the specified variable given by its identifier (or in other words its name).

Example of a simple Catscript assignment statement:

```
var x = 10;  
x = 1;
```

Function Declaration:

Catscript function declaration as defined in grammar:

```
function_declaration = 'function', IDENTIFIER, '(', parameter_list, ')' +  
    [ ':' + type_expression ], '{', { function_body_statement }, '};
```

Function declaration is initiated with the 'function' keyword. The name of the function is the identifier. A set of opening and closing parentheses follows after the identifier. Inside these parentheses goes a list of parameters. The function declaration can then optionally have its type specified by adding a colon followed by the relevant type expression. Finally a set of opening and closing curly braces marks the body of the function declaration. Inside the curly braces goes a function body statement which is documented below.

Function Body Statement:

Catscript function body statement as defined in grammar:

```
function_body_statement = statement |  
                          return_statement;
```

The function body statement is either a statement or a return statement. Functions do not always have to return a value so any statement can be used in the body as the statement extends to all other documented statements in this documentation.

Parameter List:

Catscript parameter list as defined in grammar:

```
parameter_list = [ parameter, {' parameter } ];
```

A parameter list is a list that is constructed with any number of parameters. This means the list could be empty. Parameter is documented below.

Parameter:

Catscript parameter as defined in grammar:

```
parameter = IDENTIFIER [ , ':', type_expression ];
```

A parameter is an identifier with an optional type expression after a colon indicating the type will be specified.

Example of a parameter;

```
x : int
```

Return Statement:

Catscript return statement as defined in grammar:

```
return_statement = 'return' [, expression];
```

The return statement starts with the 'return' keyword followed by an expression. The expression is evaluated as some value so functions can return integers, strings, booleans, etc...

Example of a Catscript return statement:

```
var x = 1;  
return x;
```

Expression:

Catscript expression as defined in grammar:
expression = equality_expression;

A Catscript expression follows a long line of expression types with expression being at the very top of that chain. As can be seen from the grammar, the expression is made up of an equality expression. Looking further ahead it can be seen that an equality expression is made up of a comparison expression, an operator, and another comparison expression. This pattern continues until ultimately ending at the primary expression. This means the Catscript expression could be some combination, or some other type of expression from further down the list.

Simple examples of Catscript expression:

1 + 1
1 >= 1
1 * 1

Equality Expression:

Catscript equality expression as defined in grammar:
equality_expression = comparison_expression { ("!=" | "==") comparison_expression };

The Catscript equality expression is one of two operators used on two instances of comparison expressions. The first operator is the bang equal or not equal which is used when the first expression should not be equal to the other expression. The second operator is the equal equal operator which is used when the first expression should equal the second expression. Once evaluated, returns true or false based on the expressions and operator used.

Simple example of Catscript equality expression:

1 == 1

Comparison Expression:

Catscript comparison expression as defined in grammar:
comparison_expression = additive_expression { (">" | ">=" | "<" | "<=") additive_expression };

The Catscript comparison expression is the comparison of two expressions using some operator. The usable operators consist of: greater, greater equal, less, less equal. These operators are used to see if the first expression is larger, larger or equal, lesser, lesser or equal to the second expression. Once evaluated, returns true or false based on the result of the chosen operator on the expressions.

Simple examples Catscript of comparison expression:

1 >= 2
1 < 2
1 <= 1

Additive Expression:

Catscript additive expression as defined in grammar:

`additive_expression = factor_expression { ("+" | "-") factor_expression };`

The Catscript additive expression adds or subtracts (based on the operator) a factor expression from another factor expression. Evaluation of the additive expression is the result of addition or subtraction on the involved expressions. Additive expressions can be made up of multiple additive expressions for example: an additive expression where both sides are additive expressions.

Simple examples of Catscript additive expression:

`1 + 1`

`1 + 1 + 1 + 1`

Factor Expression:

Catscript factor expression as defined in grammar:

`factor_expression = unary_expression { ("/" | "*") unary_expression };`

The Catscript factor expression either multiplies or divides (based on the operator) a unary expression by another unary expression. Evaluating the factor expression results in the multiplication or division of the both sides expressions.

Simple examples of Catscript factor expression:

`1 * 1`

`1 / 1`

Unary Expression:

Catscript unary expression as defined in grammar:

`unary_expression = ("not" | "-") unary_expression | primary_expression;`

The Catscript unary expression is the word 'not' or the '-' negative sign followed by another unary expression or a primary expression.

Primary Expression:

Catscript primary expression as defined in grammar:

`primary_expression = IDENTIFIER | STRING | INTEGER | "true" | "false" | "null" |
list_literal | function_call | "(", expression, ")"`

The Catscript primary expression can be one part of a large list of options. This list includes: identifiers, strings, integers, keyword 'true', keyword 'false', keyword 'null', list literal, function calls, and other expressions.

Simple examples of Catscript primary expressions:

"This is a string"

1

x

List Literal:

Catscript list literal as defined in grammar:

list_literal = '[', expression, { ',', expression } '];

The Catscript list literal is an expression followed by any number of optional expressions thus forming a list of expressions.

Function Call:

Catscript function call as defined in grammar:

function_call = IDENTIFIER, '(', argument_list, ')'

The Catscript function call is initiated by an identifier representing the name of the function and is then followed by an argument list.

Argument List:

Catscript argument list as defined in grammar:

argument_list = [expression , { ',', expression }]

The Catscript argument list is very similar to the list literal in that it is an expression followed by any number of additional expressions. It differs as it is the input parameters for function calls.

Type Expression:

Catscript type expression as defined in grammar:

type_expression = 'int' | 'string' | 'bool' | 'object' | 'list' [, '<' , type_expression, '>']

The Catscript type expression represents variable types such as: int, string, bool, object, and list of some type represented by a type expression.

List type example:

list<int>