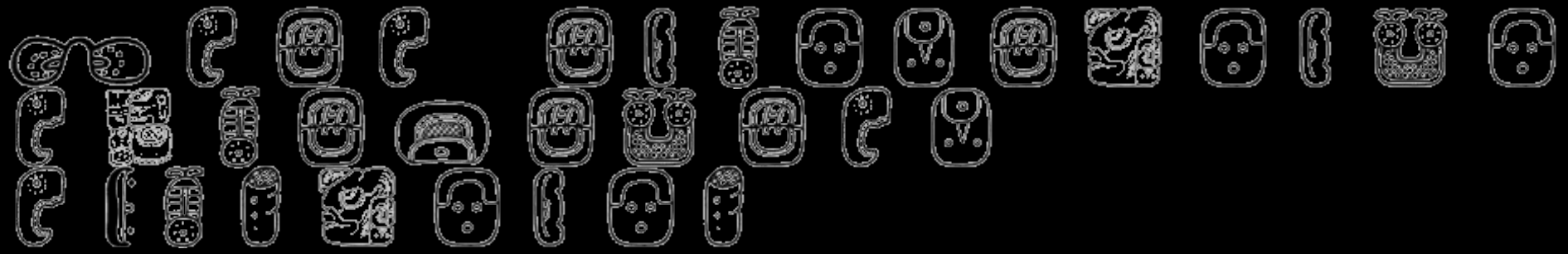
Maia Cloud Lab

MaiaScript
A short introduction

by

Roberto Luiz Souza Monteiro



M.A.I.A. MaiaScript MaiaStudio MaiaRecorder Meet LearningBLockly

Maia Cloud Lab

What is MaiaScript and what is intended with it?

- MaiaScript is a programming language focused on building adaptable and intelligent applications, with an emphasis on ease of learning and high performance. Operations with complex numbers and matrices, creation and analysis of complex and social networks, artificial neural networks, access to SQL databases, parallel programming with threads and GPU, advanced statistics, algebraic computing, including differential and integral calculation and programming of desktop and web applications are natively supported.

Data types

- MaiaScript supports three types of data natively: integer, real and string. These types are automatic, and you do not have to define them when creating common variables and functions. For use exclusively with functions in WebAssembly and MaiaAssembly are supported the types integer 32 bits, ì32, integer 64 bits, ì64, real 32 bits, f32 and real 64 bits, f64. Functions in MaiaAssembly are handled in the chapter on functions. Functions in WebAssembly are beyond the scope of this guide. For more information see the official project website: https://webassembly.org.

Maia Cloud Lab

Data output

- MaiaScript allows the display of messages on the computer screen, or in the standard output, through various functions of the system library, among them the most used are print, println, printf and showMessageDialog. The following example illustrates the use of these functions:

```
system.println("Hello World!")
system.showMessageDialog("Hello World!")
system.printf("%d, %.3f, %s", 1, 1.23456, "Hello World!")
```

Maia Cloud Lab

Data entry

- You can read data entered by the user using the showInputDialog function of the system library. This function displays a dialog box with the message passed as a parameter and a confirmation button, which when clicked, returns the value entered in the displayed text box. The following example illustrates the use of this function:

```
a = system.showInputDialog("Type a number:")
system.println(a)
```

Variables

- Variables are containers where we store data for processing or processing results. In MaiaScript variables can store values of any type, and it is not usually necessary to specify the type of data that the variable will store at the time of its creation. However, when creating functions in MaiaAssembly or WebAssembly, you must specify the type of data that the variable will store and this variable can only store values of this type of data throughout its existence. The types integer 32 bits, ì32, integer 64 bits, ì64, real 32 bits, f32, real 64 bits, f64 are supported. Functions in MaiaAssembly are handled in the chapter on functions. Functions in WebAssembly are beyond the scope of this guide. For more information see the official project website: https://webassembly.org.

Maia Cloud Lab

Variables

- The following example shows how to create variables of various types:

```
a = 1
system.println(a)
b = 2.0
system.println(b)
c = "Hello World!"
system.println(c)
D = [1, 2.0]
system.println(d)
```

Maia Cloud Lab

Variables

- The following example shows how to create variables of various types:

```
e = []
system.println(e)

// Vectors similar to JavaScript.
f = [[1, 2],[3, 4]]
system.println(f)
```

Maia Cloud Lab

Variables

- The following example shows how to create variables of various types:

```
// Matlab-like matrices.
g = [5, 6; 7, 8]
system.println(g)

// JavaScript-like objects.
h = {a: 1, b: 2.0, "c": "Hello World!"}
system.println(JSON.stringify(h))
i = {}
system.println(JSON.stringify(i))
```

Maia Cloud Lab

Operators

- MaiaScript supports mathematical, relational, logical, bit offset operatio, combined assignment operators and ternary conditional operator.

Maia Cloud Lab

Operators

- MaiaScript supports mathematical, relational, logical, bit offset operation, combined assignment operators and ternary conditional operator.

Maia Cloud Lab

Operators

- Mathematical operators
  - MaiaScript supports the following mathematical operators: add, +, subtraction, -, potentiation, **, multiplication, *, division, / and rest of division, %. The following examples show how to use these operators:

```
a = 1
b = 2
c = a + b
system.println(c)
c = a - b
system.println(c)
c = a * b
system.println(c)
c = a / b
system.println(c)
c = a % b
system.println(c)
```

Maia Cloud Lab

Operators

- Mathematical operators

```
// Python-like power operator.
c = a ** b
system.println(c)

// Increment and decrement operators similar to C.
c = a++
system.println(c)
c = b--
system.println(c)
c = ++a
system.println(c)
c = --b
system.println(c)
```

Maia Cloud Lab

Operators

- Relational operators
  - MaiaScript supports the following relational operators: equal, ==, different, !=, minor, <, less or equal, <=, major, > and greater or equal, >=. The following examples show how to use these operators:

```
a = 1
b = 2
c = a == b
system.println(c)
c = a != b
system.println(c)
c = a < b
system.println(c)
c = a <= b
system.println(c)
c = a > b
system.println(c)
c = a >= b
system.println(c)
```

Maia Cloud Lab

Operators

- Logical operators
  - MaiaScript supports the following logical operators: and, &&, or, || , and bitwise, &, exclusive or bitwise, ^ and or bitwise, |. The following examples show how to use these operators:

```
a = 1
b = 0
c = a && b
system.println(c)
c = a || b
system.println(c)
c = a & b
system.println(c)
c = a | b
system.println(c)
c = a ^ b
system.println(c)
```

Maia Cloud Lab

Operators

- Bit offset operators
  - MaiaScript supports the following shift operators: left shift, <<, and right shift, >>. The following examples show how to use these operators:

```
a = 3
c = << 2
system.println(c)
c = >> 2
system.println(c)
```

Maia Cloud Lab

Operators

- Assignment operators
  - MaiaScript supports the following special operators of operation followed by assignment: *= , /= , %= , += , -= , <<= , >>= , &= , ^= , |=. The following are examples of the most common uses of these operators:

```
c = a += b
system.println(c)
c = a -= b
system.println(c)
```

Maia Cloud Lab

Operators

- Conditional operator (ternary)
  - The MaiaScript language offers a ternary conditional operator. This operator receives three operands: a conditional expression, an expression that is returned if the condition is evaluated to true and an expression that is returned if the condition is evaluated as false. In the following example, as the variable a contains the value 1 the condition a == 1 will be evaluated as true and the expression Hello will be returned.

```
a = 1
c = a == 1 ? "Hello" : "World"
system.println(c)
```

Maia Cloud Lab

Operators

- Complex numbers
  - MaiaScript supports complex numbers natively for the +, -, **, * and \ and for the mathematical functions abs, arg, cos, cosh, exp, log, sin, sinh, sqrt, tan and tanh. Several specialized functions are also available in the core library. For all MaiaScript functions that support complex numbers, see the library documentation in the docs folder in your MaiaScript compiler distribution. the following example illustrates the sum operation with two complex numbers:

```
e = 1.0+2.0*i
f = 3.0+4.0*i
g = e + f
system.println(g)
```

Maia Cloud Lab

Operators

- Matrices
  - MaiaScript supports matrices natively for the +, -, **, and * operators and offers the matrix library for linear algebra. Several specialized functions are also available in the core library. For all MaiaScript functions that support matrices, see the documentation for the libraries in the docs folder in your MaiaScript compiler distribution. n MaiaScript you can use both Matlab and JavaScript matrices notation. In Matlab notation the columns are separated by commas, “,”, and the lines by semicolons, “;”. In javascript notation each line must be indicated between brackets [] and lines separated by commas, “,”.

Maia Cloud Lab

Operators

- Matrices
  - The following example presents examples of operations with matrices using the two notations:

```
a = [1, 2; 3, 4]
b = [[5, 6], [7, 8]]
c = a + b
system.println(c)
c = a - b
system.println(c)
c = a ** 2
system.println(c)
c = a * b
system.println(c)
```

Maia Cloud Lab

Decision structures

- MaiaScript offers two statements for flow control: **if… else…** and **switch**. Both structures are available in both MaiaScript and MaiaAssembly. These statements will be presented in the next sessions, as well as examples of their uses.

Maia Cloud Lab

Decision structures

- If… So… Statement
  - The **if… else…** statement allows you to decide, by evaluating a condition by running a program code session or not. The conditional expression should be written immediately after the word if and in parentheses. If this expression is evaluated as true the expression or command block immediately after the parentheses is executed, otherwise the expression or command block immediately after the word else is executed. The else clause is optional.

Maia Cloud Lab

Decision structures

- If... So... Statement
  - The following example illustrates the use of the if... else... statement:

```
a = 1
b = 2
// If statement similar to C.
if (a < b) {
    system.println("a = " + a)
    system.println("b = " + b)
    system.println("a < b")
    if (a == 1)
        system.println("a == 1")
    else
        system.println("a != 1")
} else if (a > b) {
    system.println("a = " + a)
    system.println("b = " + b)
    system.println("the > b")
} else {
    system.println("a = " + a)
    system.println("b = " + b)
    system.println("a == b")
}
```

Maia Cloud Lab

Decision structures

- Switch... Case... statement
  - The s**witch... case... default...** statement allows you to decide, by comparing an expression with several provided cases, by running a program code session or not. The conditional expression should be written immediately after the word switch and in parentheses. This expression will be compared with each provided case and if an equivalence is found the expression or command block immediately after the colon of the case is executed. If none of the cases matches the given expression, the expression or command block immediately after the colon of the default case is executed. The default clause is optional.

Maia Cloud Lab

Decision structures

- Switch… Case… statement
  - The following example illustrates the use of the switch… Case… default… statement:

```
a = 1
// Switch statement similar to C.
switch (a) {
    case 0:
    case 1:
        system.println("a == 0 || a == 1 || a == 2")
    case 2:
        system.println("a == 2")
        Break
    default:
        system.println("a = " + a)
        system.println("a != 1 &; a!= 2")
}
```

Maia Cloud Lab

Repeating structures

- Loops structures allow you to run a program session a number of times or until a condition is satisfied. MaiaScript offers four looping structures: do… while, while…, for and foreach. All of these statements are available in both MaiaScript and MaiaAssembly. These statements will be presented in the next sessions, as well as examples of their uses.

Maia Cloud Lab

Repeating structures

- Do… statement
  - The **do… while…** statements executes an expression or command block while a given condition is evaluated to true. The difference from this statement and the while… statement is that this statement executes the code session at least once, even if the condition is already false when the program execution stream reaches it, while the while… statement will not execute at all if the condition is already false when the program execution flow reaches it. If you wish to stop the execution of the loop before the condition becomes false, you can use the break statement. If you want to stop running the current iteration of the loop before the command block has been fully executed and jump to the next iteration, you can use the continue statement.

Maia Cloud Lab

Repeating structures

- Do… statement
  - The following example illustrates the use of the do… while… statement:

```
a = 0
do {
    system.println(a)
    a++
} while (a < 10);
```

Maia Cloud Lab

Repeating structures

- While… statement
  - The **while…** statement executes an expression or command block while a given condition is evaluated to true. The difference of this and the declaration do… while… statement is that that statement executes the code session at least once, even if the condition is already false when the program execution stream reaches it, while the while… statement will not execute at all if the condition is already false when the program execution flow reaches it. If you wish to stop the execution of the loop before the condition becomes false, you can use the break statement. If you want to stop running the current iteration of the loop before the command block has been fully executed and you jump to the next iteration, you can use the continue statement.

Maia Cloud Lab

Repeating structures

- While… statement
  - The following example illustrates the use of the while… statement:

```
a = 0
while (a < 10) {
    if (a % 2 == 0) {
        continue
    }
    if (a >= 5) {
        system.println("Break the loop.")
        Break
    }
    system.println(a)
    a++
}
```

Maia Cloud Lab

Repeating structures

- For... statement
  - The **for...** statement executes an expression or command block while a given condition is evaluated as true. The difference of this statement and the while... declaration is that that declaration requires internal control of the execution so that at some point of execution flow the condition becomes false and the execution of the code is stopped. This statement allows you to pass three arguments: an expression that will be executed before the first interaction, a conditional expression and an expression that will be evaluated at the end of each iteration. You can use the first parameter to initialize a control variable, and the last parameter to modify it. If you wish to stop the execution of the loop before the condition becomes false, you can use the break statement. If you want to stop running the current iteration of the loop before the command block has been fully executed and you jump to the next iteration, you can use the continue statement.

Maia Cloud Lab

Repeating structures

- For… statement
  - The following example illustrates the use of the for… statement:

```
b = [1, 2, 3]
for (a = 0; a < 10; ++a) {
    system.println(a)
}
for (i = 0; i < b.length; i++) {
    system.println(b[i])
}
```

Maia Cloud Lab

Repeating structures

- Foreach... statement
  - The **foreach...** statement executes an expression or command block for each element of an associative vector or object. This statement receives three parameters: an associative array or object, a variable to contain the array key or object property name and a variable to contain the value of the array element or object. If you wish to stop the execution of the loop before the condition becomes false, you can use the break statement. If you want to stop running the current iteration of the loop before the command block has been fully executed and you jump to the next iteration, you can use the continue statement.

Maia Cloud Lab

Repeating structures

- Foreach… statement
  - The following example illustrates the use of the foreach… statement:

```
c = {a: 1, b: 2}
// Foreach statement similar to Tcl.
foreach(c; key; value) {
    system.println(key + ": " + value)
}
```

Maia Cloud Lab

Function declaration

- We declare a function by writing its name, followed by parentheses, which may or may not contain arguments separated by commas, ,, and a command block between curly braces, {}. Functions in MaiaScript may or may not have declared return types and use or not special assignment operators, =, ?=, #=, :=, in their declaration.
- If a return type is indicated in the function declaration, it is interpreted as being a function in MaiaAssembly or in WebAssembly. In both cases you must specify the value types of the function arguments if it has arguments. If the curly braces, /{ /} of the command blocks are preceded by the character / the function is interpreted as being in WebAssembly, otherwise it is considered to be in MaiaAssembly. MaiaScript functions can be recursive, that is, call themselves to perform complex tasks.

Maia Cloud Lab

Function declaration

- The following example illustrates the factorial function implemented using a recursive algorithm:

```
// A recursive function.
factorial(n) {
    if (n == 0 || n == 1) {
        return 1
    }
    return n * factorial(n - 1)
}
system.println(factorial(5))
```

Maia Cloud Lab

Function declaration

- Inline functions
  - For simpler functions, which can be implemented in only one line, you can use the simplified form of function declaration. This way allows you to write a function as it usually is done in mathematics, using the assignment, =, operator and omitting the curly braces of the command block. The following example shows the declaration of a second-degree function:

```
// An inline function.
f(x) = 2 * x ** 2 + x - 1
system.println(f(2))
```

Maia Cloud Lab

Function declaration

- Asynchronous functions
  - Functions can be executed asynchronously. To do so, you must declare the function using the asynchronous execution operator, ?=. To wait for the asynchronous function to finish its execution, blocking the execution stream of the rest of the program, you must assign the function to a variable using the asynchronous execution operator, ?=.

```
// An asynchronous function.
f(x) ?= {
    return x
}
// An asynchronous function call.
a ?= f(2)
```

Maia Cloud Lab

Function declaration

- Parallel functions
  - MaiaScript allows you to create parallel functions using threads or GPU cores. In both cases the functions need to be of the kernel type. Kernel functions must be created using the kernel function declaration operator, #=. A kernel function is compiled differently from the other functions. They do not support operations with complex numbers or matrices. Only the basic data types and features of JavaScript are supported.

Maia Cloud Lab

Function declaration

- Parallel functions
  - The following example shows how to create a thread in MaiaScript. For more details see the documentation of the task library available in the docs folder of your MaiaScript compiler distribution.

```
// A parallel function.
task1(x) #= {
    i = 0
    timedCount #= () {
        i = i + 1
        postMessage(i)
        if (i < 10) {
            setTimeout(timedCount(), 500)
        }
    }
    timedCount()
}
onMessage1(m) {
    system.log("Task 1: " + m.data)
    if (m.data >= 5) {
        t1.terminate()
    }
}
t1 = task.new(task1)
t1.onmessage = onMessage1
```

Maia Cloud Lab

Function declaration

- Functions in MaiaAssembly
  - MaiaAssembly is a build-optimized programming language for WebAssembly. It allows you to create algorithms as fast as programs written in C language, embedded in high-level programs in MaiaScript. Functions in MaiaAssembly are typed, which means that you must declare the types of functions and variables at the time of their creations. The types supported in MaiaAssembly are integer 32 bits, ì32, integer 64 bits, ì64, real 32-bit, f32 and real 64-bit, f64. All MaiaScript decision and loop structures are supported in MaiaAssembly. In addition, arbitrary dimensions matrices of supported data types are supported. You cannot pass objects or even arrays as MaiaAssembly function arguments, but you can import them. The import declaration is used for this. It allows you to import properties of objects into the function and use them as if they were local variables. In MaiaAssembly it is possible to create global variables using the global declaration. Global variables are accessible from anywhere in the program.

Maia Cloud Lab

Function declaration

- Functions in MaiaAssembly
  - The following example shows how to create a function to sum two values passed to it as arguments. The function also creates a local variable to store the sum result. Local variables must be declared in the function header and should appear after the function arguments.

```
// A function in MaiaAssembly.
i32 f4(i32 a, i32 b, site i32 c) {
    c = a + b;
    return c;
}
// Calling a function in MaiaAssembly.
c = f(1, 2)
```

Maia Cloud Lab

Function declaration

- Functions in JavaScript
  - Functions in JavaScript can be declared preceding the curly braces, /{ /} of the command blocks with the character /. JavaScript functions are not compiled, and are inserted into compiler-produced code as they were written. The following example shows how to define a function in JavaScript:

```
// A function in JavaScript.
f(x) /{
    y = x + 1;
    return y;
}/
// Calling a function in JavaScript.
c = f(2)
```

Maia Cloud Lab

Function declaration

- Functions in WebAssembly
  - Functions in WebAssembly are assembled by the assembler and inserted into binary form into the code resulting from the build. They are typed, which means that you need to declare the types of functions and variables at the time of their creations. The types supported in WebAssembly are integer 32 bits, ì32, integer 64 bits, ì64, real 32-bit, f32 and real 64-bit, f64. Local variables must be declared in the function header and should appear after the function arguments.

Maia Cloud Lab

Function declaration

- Functions in WebAssembly
  - The following example shows how to create a function to sum two values passed to it as arguments:

```
// A function in WebAssembly.
i32 f(i32 a, i32 b) /{
    (i32.add
      (local.get $a)
      (local.get $b)
    )
}/
// Calling a function in WebAssembly.
f = f(1, 2)
```

Maia Cloud Lab

Creating namespaces and objects

- Namespaces are a way to organize functions and variables to build libraries. The use of namespaces not only makes code more organized and reusable, but also makes access to library resources more efficient. Every namespace is an object, but namespaces are not object constructors. To create objects we must create constructors for them. In the next few sessions we'll see how to create namespaces and object constructors.

Maia Cloud Lab

Creating namespaces and objects

- Creating namespaces
  - We create a namespace by defining a name for it and a block of code containing variables and functions. The following example illustrates how to create a namespace containing a variable, property and a function, method:

```
// Creating a namespace (an object)
a {
    b = 1
    f(n) {
        if (n == 0 || n == 1) {
            return 1
        }
        return n * this.f(n - 1)
    }
}
system.println(a.b)
system.println(a.f(5))
```

Maia Cloud Lab

Creating namespaces and objects

- Object constructors
  - Object constructors allow you to create class instantiations defined by them. Classes are templates for objects. They define their properties, changeable characteristics at runtime, and their methods, functionalities of objects. To create an object constructor we define a function using the object creation operator, :=. To instantiate an object we assign to a variable the return value of the object constructor, using the object creation operator, :=. The following example creates an object that has a y property and assigns to that variable the value passed to the constructor at the time of its creation:

```
/// An object constructor.
A(x) := {
    y = x
}
c := A(2)
system.println (c.y);
```

Maia Cloud Lab

Complex and social networks

- MaiaScript provides several functions for creating and analyzing complex and social networks. These features are available in the **cna** and **snet** libraries.
  - Some supported operations:
    - Creating random, scale-free, small-world, and hybrid networks. Calculation of degrees, mean degree, density, shortest paths, mean shortest path, clustering coefficients, mean clustering coefficient, diameter, centralities, degree centrality, centralities and betweenness, centrality and proximity, global efficiency. CPU and GPU utilization in calculations. For a complete reference see the documentation available in the docs folder of your MaiaScript compiler distribution.

Maia Cloud Lab

Artificial neural networks

- MaiaScript provides functions for creating and training artificial neural networks of various topologies. These features are available in the **ann** library.
  - Some supported operations:
    - Creation of random topologies, scale-free, small world, hybrid and multilayer perceptron neural networks. Automatic neural network training and use of the trained network for intelligent operations.

Maia Cloud Lab

SQL database

- MaiaScript natively supports the SQLite database but can use any database supported by Node.js. These features are available in the core library. The following example creates a database, a table, and inserts data into the created table. For a complete reference see the documentation available in the docs folder of your MaiaScript compiler distribution.

```
dataHandler(transaction, results) {
}
errorHandler (transaction, error) {
}
createTable(transaction) {
    scheme = ""
    scheme = scheme + "CREATE TABLE people(id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,"
    scheme = scheme + "name TEXT NOT NULL DEFAULT `John Doe`,"
    scheme = scheme + "shirt TEXT NOT NULL DEFAULT `Purple`);"
    transaction.executeSql(scheme, [], dataHandler, errorHandler)
    transaction.executeSql("insert into people (name, shirt) VALUES (`Joe`, `Green`);", [], dataHandler, errorHandler)
    transaction.executeSql("insert into people (name, shirt) VALUES (`Mark`, `Blue`);", [], dataHandler, errorHandler)
    transaction.executeSql("insert into people (name, shirt) VALUES (`Phil`, `Orange`);", [], dataHandler, errorHandler)
    transaction.executeSql("insert into people (name, shirt) VALUES (`jdoe`, `Purple`);", [], dataHandler, errorHandler)
}
// Opens the database if it exists or creates a new one if it does not exist.
db = core.openSQLDatabase("Test", "1.0", "Test", 65536)
// Creates a table and inserts data into it.
if (typeof(db) != "undefined") {
    db.transaction(createTable)
}
```

Maia Cloud Lab

Parallel programming using GPU

- You can speed up processing on some issues by using parallel programming. MaiaScript allows for real parallelism using GPU cores if this feature is available on the host machine. If not, the MaiaScript compiler will compile the program for sequential execution. GPU computing functions are called shaders. These functions are compiled differently by the MaiaScript compiler and do not support complex numbers or calculations with matrices. GPU programming features are available in the gpu library. For a complete reference see the documentation available in the docs folder of your MaiaScript compiler distribution.

Advanced statistics

- MaiaScript offers several statistical functions for operations with matrices and CSV files. These functions are available in the matrix, statistics and dfa libraries. The statistics library implements functions for calculations of averages, deviations and standard errors, as well as functions involving random numbers and normal distribution, including the calculation of the inverse of the normal distribution. The dfa library implements calculations of DFA, DCCA and rhoDCCA. For a complete reference see the documentation available in the docs folder of your MaiaScript compiler distribution.

Maia Cloud Lab

Algebraic computing

- MaiaScript has a complete CAS (Computer Algebra System) implemented in the cas library. This CAS allows you to simplify expressions, solve equations and perform complex operations of linear algebra and differential and integral calculation. The CAS is based on the open source Algebrite library. For a complete reference see the official Algebrite project documentation http://algebrite.org. The only exception is that Algebrite originally uses the ^ operator for powering and in MaiaScript the power operator is **.