Choo Choo Train Compiler

Write and review code like an expert!

Choo Choo Train User Manual

Getting Started

This reference manual describes the syntax and “core semantics” of the Choo Choo Train language. The language is friendly to new programmers. Programs are built block by block, where each block is a function or procedure that contains a body of code.

The main objective of Choo Choo Train language is to be understandable, visual, and easy to use so that people are be motivated to learn how to program. The programmer first codes with blocks, then runs the code with the Choo Choo Train Compiler and finally can review how the code behaves as if it were a train ride.

By mapping the elements of a language (such as loops, statements, etc.) to a specific type of train track section or element, we can help people understand the logic and data flow of their program.

Language Elements

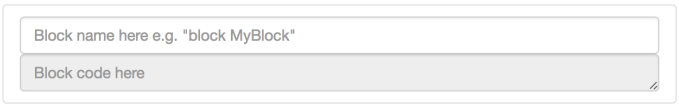
Input and Output

Operations with Blocks

First Choo Choo Train Program

The Choo Choo Train language is similar to most programming languages. The basic principle of the language is to code by blocks. These blocks are sections of code and can receive and send information to other blocks.

To code your first Choo Choo Train program you can either add a text file with code or click on the Add Block button. The following prompts will appear.



To start coding a block, write the block name on the first section as: "starting block <block-name>"

> The first block/main block must always have the keywork **"starting"** at the beginning of the block name. Any other block must not contain this keyword. The starting block is the first block to be executed.

To write code inside the block, start typing on the second block. Block code must be inside two { } (curly braces). A simple block with no parameters and no return values with only one print statement looks like this.

After the code is written or uploaded in a text file, the next step is to click on the compile code button and your code will start compiling and running.

The previous example produces the following result.

```
Hello World
```

More details on how to code blocks are seen in the next sections.

Back to top

Identifiers

A Choo Choo Train identifier is a name used to identify a variable, list, block, or any other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Choo Choo Train does not allow special characters such as @, $, and % within identifiers.

> Choo Choo Train is a case sensitive programming language. Thus, **Apple** and **apple** are two different identifiers.

Back to top

Reserved Words

The following list shows the Choo Choo Train keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Choo Choo Train keywords contain lowercase letters only.

A line containing only whitespace, to separate statements for visual effect, is known as a blank line and Choo Choo Train totally ignores it.

Reserved				
block	starting	receives	returns	return
call	variable	list	of	type
do	until	if	else	input
print	whole	decimal	words	boolean
and	or	not	True	False

Back to top

Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a whole value and his or her address is stored as alphanumeric characters. Choo Choo Train has various standard data types that are used to define the operations possible on them and the storage method for each of them. Choo Choo

Train has four standard data types.

Whole

Whole data types store numeric values that are whole numbers or integers.

Decimal

Decimal data types store numeric values that are decimal numbers.

Words

Words are identified as a continuous set of characters represented in quotation marks.

Boolean

Boolean data types store values that are either True or False.

Data Type	Example
Whole	10, 1000, 234, 0, -3
Decimal	3.3, 3.000, 32.3+e18, 7.2-E12
Words	"Hello", "123", "a", "This is a sentence.", "1+2"
Boolean	True, False

Back to top

Operators

Operators are the constructs which can manipulate the value of operands. Consider the expression 1 + 2 = 3. Here, 1 and 2 are called operands and + is called operator.

There are different types of operators that can handle different types of operands. Choo Choo Train can handle several operators: Arithmetic, Comparison, Assignment, and Logical.

Operators behave differently with different data types.

Arithmetic

Arithmetic operators are the ones that can manipulate the value of operands that are whole or decimal. They are interchangeable and can be used between the two data types, if the operation contains a decimal, the result will be decimal.

+ Addition
Adds values on either side of the operator.

- Subtraction
Subtracts right hand operand from left hand operand.

* Multiplication
Multiplies values on either side of the operator

/ Division
Divides left hand operand by right hand operand

Comparison

Comparison operators are the ones that compare the values on either sides of them and decide the relation among them. Decimal and whole can use all operators interchangeably and the result will always be boolean. Words data types can only use equal to and not equal to.

== Equal to
If the values of two operands are equal, then the condition becomes true.

!= Not equal to
If values of two operands are not equal, then condition becomes true.

< Less than

If the value of left operand is less than the value of right operand, then condition becomes true.

> Greater than

If the value of left operand is greater than the value of right operand, then condition becomes true.

>= Less than or equal to

If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

>= Greater than or equal to

If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.

Assignment

Assignment operators are the ones that can assign values of the right operand to the left operand of the operation, they must be the same type. Decimal data types can accept whole data types as an exception to the rule.

= Assign

Assigns values from right side operands to left side operand, they must be of the same type.

Logical

Logical operators are the ones which can relate the value of operands that are boolean, both operands are of type boolean and the result is also boolean.

and

If both the operands are true then condition becomes true.

or
If any of the two operands are true then condition becomes true.

not
Used to reverse the logical state of its operand.

Back to top

Variable Declaration

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, Choo Choo Train reserves memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store wholes, decimals, words, or booleans in these variables.

Variables need to be explicitly declared with their type. The variable declaration happens on the top of the block body, which happens inside the two { } (curly braces). Variables are not assigned any value at this point. The variable declaration has the following syntax.

```
variable <variable-name> of type <variable-type>;
```

Variables can also be declared as groups of the same type as follows.

```
variable <variable-name>, <variable-name>, <variable-name> of type <variable-type>;
```

An example of how to code variable declarations of multiple data types is as follows.

```
starting block Variables
{
 variable a, b, c, d, e of type decimal;
 variable g of type whole;
 variable h, i of type words;
}
```

Back to top

List Declaration

Lists declarations are very similar to variable declarations.

The list is a very useful datatype which can be decalred as a variable with a fixed size defined between square brackets. The important thing about a list is that items in a list need to be of the same type.

The list declaration is also on the top of the block body. The list declaration has the following syntax.

list <list-name> **[** <list-size> **] of type** <variable-type>;

An example of how to code a simple list declaration is as follows.

```
starting block myListBlock
{
 list myThings [5] of type words;
}
```

Back to top

Assigning Values

The equal sign (=) is used to assign values to variables. Variables must be previously declared. More information on how to declare variables can be found here.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For list assignment, each element of the list must be accessed as follows.

<list-name> **[** <list-index> **] =** <any value>;

The value to be assigned can be a constant, another variable, a list element, a block call that returns a value, or an expression. An example of how to code assignments with only constants is as follows.

```
starting block AssignBlock
{
 variable a, b, c, d, e of type decimal;
 variable g of type whole;
 variable h, i of type words;
 list z[10] of type whole;

 a = 1.1;
 b = 2.34;
 c = 105;
 d = 0.4567;
 e = 1;
 g = 3;

 h = "hello";
 i = "world";

 z[1] = 1;
 z[2] = 0;
}
```

Back to top

Conditional Statements

What makes Choo Choo Train so much more powerful are conditional statements. This is the ability to test a variable against a value and act in one way if the condition is met by the variable or another way if not.

The condition can be anything that evaluates as True or False. Comparisons always return True or False. The else part is optional. If you leave it off, nothing will happen if the conditional evaluates to ‘False’.

An **if** statement consists of a boolean expression followed by one or more statements. The structure of a simple if statement is as follows.

if (<some-expression> **) {** <some-code> **}**

The use of a simple if statement is shown below.

```
starting block myConditional
{
 variable a, b, c of type decimal;

 a = 1;
 b = 2;
 c = a * b * b;

 if (c + 5 > 5){
       c = 0;
 }

}
```

An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is False. The structure of a simple if else statement is as follows.

if (<some-expression> **) {** <some-code> **} else {** <some-code> **}**

You can use one **if** or **else** statement inside another **if** or **else** statement(s). The use of a simple if else statement is shown below.

starting block myConditional

```
{
 variable a, b, c of type decimal;

 a = 1;
 b = 2;
 c = a * b * b;

 if (c + 5 > 5){
       c = 0;
 }

 else {
      c = 1;
 }
}
```

Back to top

Loop Statements

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

Choo Choo Train provides a simple loop statement, called a do until loop. It repeats a statement or group of statements until a given condition is True. It tests the condition after executing the loop body. So the loop body is executed at least once.

The structure of a simple do until loop is as follows.

do { <some-code> **} until (** <some-expression> **)**

The use of a simple do until loop is shown below.

```
starting block myLoop
{
 variable a, b, c of type whole:
 a = 1;
 b = 3;
 c = 0;
 do {
  print( a );
 c = a + b;
 a = a + 1;
 until ( a >  b );
}
```

Back to top

Input and Output

Choo Choo Train provides a way to produce output by using the print statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a word data type and writes the result to standard output as follows.

```
starting block HelloWorld
{
 print ("Hello World");
}
```

The previous example produces the following result.

```
Hello World
```

Choo Choo provides a way to read a line of text from standard input. The function **input(** <some-variable-name> **);** reads one line from standard input and assigns it to the words data type variable specified.

```
starting block myInputOutput
{
 variable h, i of type words;
 h = "Hello ";
 input( i );
 print( h + i );
}
```

This prompts you to enter any string and it would display that same string on the screen with an added "Hello " When typed "World", its output is like this.

```
Hello World
```

Back to top

Operations with Blocks

A Choo Choo Train block is a block of organized, reusable code that is used to perform a single, related action. Blocks provide better modularity for your program and a high degree of code reusing. Blocks have to be called in order to be executed. A block call has a simple syntax using the keyword **call** . Blocks can receive parameters separated by commas, these parameters can be variables, constants, expressions, or anything that holds a value. We will review parameters further down.

call <block-name> **(** <some-parameter> **);**

call <block-name> **(** <some-parameter>, <some-parameter> **);**

You can define blocks as simple as naming them and adding code using the following syntax.

block <block-name>

{ <some-code> **}**

The previous syntax is the simplest block one can create with Choo Choo Train.

The first block/main block must always have the keywork **"starting"** at the beginning of the block name. Any other block must not contain this keyword. The starting block is the first block to be executed.

Blocks have optional syntax elements that boost their functionality. Blocks can either have **parameters** and/or **return statements**. Parameters are variables that can be sent to blocks and are used within the block's scope. Parameters are local to the block's scope and are disposed of after finished the block execution. If a block receives parameters, the parameter names and types must be specified separated by commas (if more than one) as follows.

block <block-name> **receives :** <variable-name> **of type** <variable-type>

{ <some-code> **}**

block <block-name> **receives :** <variable-name> **of type** <variable-type> , <variable-name> **of type** <variable-type>

{ <some-code> **}**

A return statement specifies the passing back an expression to the caller block, and can only exist once in the code block. If the block contains a return statement, the type must be specified as follows.

block <block-name> **returns** <variable-type>

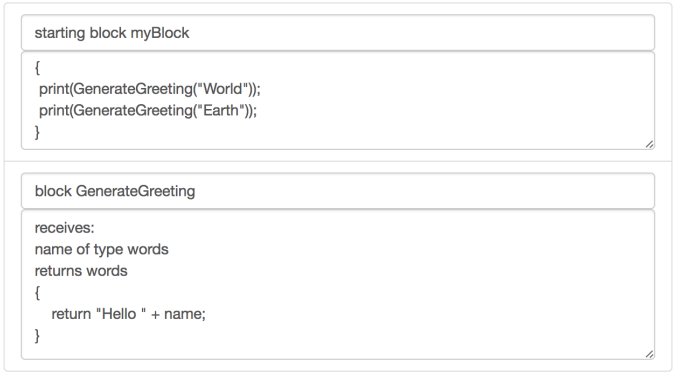
{ <some-code> **}**

If a block receives parameters and contains a return statement, the parameter definition must be stated first, followed by the return data type as follows.

block <block-name> **receives :** <variable-name> **of type** <variable-type> **returns** <variable-type>

{ <some-code> **}**

The use of the last block definition mentioned would look as follows.



The previous has both parameters and a return statement. The return value is then used in a print function. The following output is generated.

Hello World

Hello Earth

Back to top