Nini Programming Language

Reference Manual

Document version: 0.1-alpha
Nini version: 0.1-alpha
Last update: December 19th, 2024

Website: NiniDraws.com
Email: nini.loves.coding@gmail.com

About Nini

Nini is a simple programming language that is primarily made to help non-programmers learn the basics of computer programming. In this context Nini's target demographics are young students and people who haven't had much exposure to -- or experience with, tech or more specifically, personal computers.

Given many schools and people without great computer skills don't have access to up-to-date hardware and software, Nini's code editor has been made using legacy techniques and technologies, to increase the chance of it being able to run on older hardware and software.

Nini also could be useful to others, regardless of their tech literacy. For example, many artists could potentially use Nini to create either artwork or even graphics that could be useful in various applications.

Nini is a new programming language and it is still under active development with a large to-do list of features and bug fixes. This means both core language features and its code editor could be updated weekly, if not daily. Also since the current version is 0.1-alpha, breaking changes could be introduced at any point.

Initially, Nini was inspired by a programming language called Design by Numbers, DBN, which was created at MIT Media Lab during the 1990s; However, it very quickly found its own set of aims and goals and besides some keywords and basic concepts, shares no other similarities with DBN.

Core Concepts

Nini is an imperative, declarative and procedural programming language that compiles text-based source code into HTML5 Canvas drawings which could be saved as PNG files. Each

line of code is called an Instruction, except for Comments and Directives. Comments have no effect in the final result and Directives apply to whatever instructions that come after them.

The canvas element is referred to and represented via ‘paper’ instructions. You must use ‘paper’ instructions at the very beginning of your code, otherwise some drawings or instructions may not render properly. Variable declaration can come before ‘paper’ instructions as they are not drawing instructions.

There are two distinct methods for creating an object on the canvas: ‘draw’ and ‘paint’ methods. Instructions that start with ‘draw’ create hollow objects e.g. only the border or outline is drawn, whilst instructions that start with ‘paint’ create solid objects, filled with a colour.

In order to draw a visible shape on the canvas, the ‘pen’ instructions must be used to set the size and the colour of the border that will be drawn. Think of this as picking up a certain pen from a set of pens that has a specific colour and tip size.

To create solid objects, the ‘brush’ instruction must be used to set up the color for solid objects. The ‘brush’ instruction does not have a size property, so choosing the colour is all you need.

Both ‘pen’ and ‘brush’ instructions will apply to every instruction that comes after them, until their values and set up changes again. For example, if you set up the pen to have size of 2 and colour of red, this will apply to all ‘draw’ instructions until the pen size or colour is changed again.

In Nini, many instructions require numerical values, for example the length of a line or the degree that it should rotate. Currently all numeric values are valid from -9999 to 9999, unless otherwise is specified. Some values don’t make a lot of sense for some instructions, however, to keep things simple, all numbers follow one single rule. This may change in future versions of the Nini language.

Coordinate System

Nini uses a 2-dimensional Cartesian coordinate system to know exactly where to place an object on the paper. Once the width and height of the paper is set via ‘paper’ instructions, the compiler places the 0,0 origin at the centre of the paper. Since both width and height of the paper are in pixels, all coordinates are also units of pixel.

The correct format for an X and Y coordinate is to separate them with a comma and surround them by square brackets. This creates a ‘list’ which has other uses in Nini as well. Here are some examples of valid coordinates: [0,0] [-10,90] [-1000,-1000].

X and Y coordinates are numeric values, which means they are limited to values from -9999 up to 9999.

Supported Colours

Colours are used everywhere in Nini programs. Currently a colour could be specified in two different formats: One being via Hexadecimal syntax for sRGB values and the other one CSS Named Colours.

Hex Colours

All Hex Colours have three main colour components and one optional alpha or opacity/transparency component. The order of components is the amount of colour Red, the amount of colour Green and the amount of colour Blue, followed by the optional level of opacity. The opacity if not set will default to 100%.

Hexadecimal values start from 0 to 9 and A to F for values 10 to 15. sRGB values for each component could be represented via either one or two hexadecimal values. The full value starts from 0 and ends at 255. In hexadecimal this is represented by 00 for 0 and FF for 255. To represent the colour red in hexadecimal sRGB format, we can write FF0000FF. The first FF means 100% red. The following 00 means 0% green. And the second 00 means 0% blue. The final FF means the opacity should be 100% e.g. fully opaque. Since the default value for opacity is FF by default, we can shorten our representation to FF0000.

In this case since all pairs have repeating values e.g. FF is a pair of Fs, we can even further shorten this to F00. And finally to indicate that this is indeed a hexadecimal sRGB value, we should add a # character to the beginning of our code. So the final result for colour red could be any of the following: #FF0000FF, #FF0000, #F00F or #F00.

Here are some examples of Hex Colours:

- Black: #000, #000F, #000000, #000000FF
- White: #FFF, #FFFF, #FFFFFF, #FFFFFFFF
- Yellow: #FF0, #FF0F, #FFFF00, #FFFF00FF
- Silver: #C0C0C0, #C0C0C0FF
- Brown: #A52A2A, #A52A2AFF

Named Colours

Some sRGB colours have a unique name assigned to them, so instead of their sRGB value, their name could be used. There are two sets of Named Colours defined in web standards, however, in practice there is no difference between using any of these values in your Nini programs.

Standard Colours: black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua.

Extended Colours: aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkgrey, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkslategrey, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dimgrey, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, grey, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgray, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightslategrey, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, rebeccapurple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, slategrey, snow, springgreen, steelblue, tan, teal, thistle, tomato, transparent, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen.

References:

- https://developer.mozilla.org/en-US/docs/Web/CSS/hex-color
- https://developer.mozilla.org/en-US/docs/Web/CSS/named-color

Comments

Double-slash Comment

Formats:

- // comment

Parameters:

- comment: The text that will not be executed as an instruction.

Examples:

- // first box from top-left corner

Version: 0.1-alpha and higher

Description: Comments are lines of code that will not run by the compiler. You can use this feature to make your programs more readable.

Directives

All directives start with a single dash/minus character at the beginning of the line. The instruction of the directive will apply to every instruction that follows, unless it is overwritten by another directive of the same instruction.

Change Order

Formats:

- - order value

Parameters:

- value: The order of execution of the following instructions.

Examples:

- - order 3
- - order -100

Version: 0.1-alpha and higher

Description: In normal conditions all instructions run sequentially -- one after each other, exactly in the same order as they appear in the source code. This means the later instructions will be drawn on top of the earlier drawings. Sometimes it could be useful to draw something either earlier or later, but without physically moving the lines of code. For this purpose, the Change Order directive could be used with values from -9999 up to 9999.

Be mindful of the fact that behind-the-scenes, the compiler will literally move the lines that come after the Change Order directive to their appropriate place either earlier or later in the code. This may create some confusing situations for beginner Nini programmers. For example, if you change the pen colour to red, then change the order to execute earlier in the code, the drawing instructions will not be able to pick up on the red colour, and will rather follow whatever colour instruction was set earlier and closest to their execution.

Under-the-hood, the parser assigns ‘order = 0’ to all instructions, by default. However, the compiler sorts all instructions by order, before running them. The Change Order directive assigns custom orders to instructions that come after it. Which means when the compiler sorts the parsed instructions, they will appear earlier or later in the code, depending on their new order value.

For example, if the directive ‘- order 5’ is used and after a few instructions ‘- order 3’ is applied, the order of execution will be:

1. The instructions that have the default order of 0, according to their order and appearance in the code.

2. The instructions that come after '- order 3'.
3. The instructions that come after '- order 5' up until '- order 3'.

Note: Internally some instructions have hard-coded order values that cannot be overwritten by Change Order directive. For example 'paper' instructions will always have the order of -10,000. This makes sure that these instructions are always run by the compiler first, otherwise the orders that run before the 'paper' instructions will be normally ignored and not appear in the compiled program.

Change Rotation

Formats:

- - rotate degree

Parameters:

- degree: The desired rotation degree.

Examples:

- - rotate -35

Version: 0.1-alpha and higher

Description: The global rotation directive rotates every object that comes after it to the specified angle. All rotation angles are in degrees, rather than radians. The global rotation angle will not apply to the objects that are created on the paper with rotation degrees already specified. The valid values for the rotation angle are -9999 to 9999, although in most cases specifying -360 to 360 makes more sense.

Change Centre

Formats:

- - center [x,y]

Parameters:

- x: X coordinate of the new centre of the paper.
- y: Y coordinate of the new centre of the paper.

Examples:

- - center [-100,100]

Version: 0.1-alpha and higher

Description: The change centre directive will move the centre of the paper from its original position -- width divided by two and height divided by two, to the specified coordinate. All

instructions coming after this directive, will assume the new coordinate as [0,0]. For example, after setting the centre to [-50,0], drawing at [0,0] will be placed at [-50,0]. All change centre directives are based on the true centre of the canvas, so changing the centre to [-50,0] and then back to [50,0] will indeed place the new centre on [50,0] according to the true centre rather than relative to [-50,0].

Loops

Repeat

Formats:

- repeat number

Parameters:

- number: The number of times the next instruction should repeat.

Examples:

- repeat 10

Version: 0.1-alpha and higher

Description: The repeat instruction is a basic loop mechanism that repeats only the next non-commented instruction that comes after it, for the number that is specified in the instruction itself. The valid values for the number of repeats are 0 to 9999. In the next instruction the sharp character # will be replaced by the compiler with the number of the current loop e.g. like an index or a loop counter in other languages. This can be used with the built-in calc() function to calculate new values based on the current loop e.g. calc(# * 10)

Variables

Variable Declaration

Formats:

- data $name value

Parameters:

- name: The name of the variable.
- value: The value of the variable.

Examples:

- data $width 800

Version: 0.1-alpha and higher

Description: Variables are a set of Alphanumeric characters, similar to a word, that start with the $ character and are declared using the ‘data’ instruction. Variables, as their name suggests, hold a reference to a piece of data that could change in the course of execution of a program.

This means one variable can be used in the declaration of another variable. For example if we have already declared ‘data $width 400’, then we can also declare $height as ‘data $height $width’.

In the current version Nini, the value of a variable is assigned when it is declared. In order to change the value of a variable, we need to use the same syntax and instruction again to assign a new value to the same variable. This behaviour may change in the future versions and the variables may be able to update their value, without having to use the ‘data’ keyword again.

Variable Usage

Formats:

- $name

Parameters:

- name: The name of the variable.

Examples:

- paper size $width $height

Version: 0.1-alpha and higher

Description: When a variable is used in an instruction, its name will be simply replaced by its value, before the instruction is executed.

Paper

Paper Size

Formats:

- paper size width height

Parameters:

- width: The width of the paper in pixels, from 0 to 9999.
- height: The height of the paper in pixels, from 0 to 9999.

Examples:

- paper size 800 400

Version: 0.1-alpha and higher

Description: Normally ‘paper size’ instruction is the first instruction in all of your programs. The only other logical option is having some or all of your ‘data’ variable instructions before this instruction. You should note that most instructions that deal with painting or drawing objects on the paper, will not work properly if the paper dimensions are not set.

Other exceptions are Pen and Brush instructions, as well as some directives like the Change Centre and Global Rotation, since they don’t directly draw anything on the paper.

Paper Colour

Formats:

- paper color value

Parameters:

- value: The colour value.

Examples:

- paper color yellow

Version: 0.1-alpha and higher

Description: This instruction must come after Paper Size instruction and its purpose is to paint the entire paper with the specified colour.

Pen

Pen Size

Formats:

- pen size value

Parameters:

- value: The thickness of the lines in pixels.

Examples:

- pen size 4

Version: 0.1-alpha and higher

Description: This instruction sets the thickness of the lines for all ‘draw’ instructions that come after it. Valid values are 0 up to 9999. Setting pen size to 0 will skip ‘draw’ instructions. The default pen size is 1 pixel.

Pen Colour

Formats:

- pen color value

Parameters:

- value: The colour value

Examples:

- pen color red
- pen color #AE580F
- pen color #AE580F55

Version: 0.1-alpha and higher

Description: This instruction selects the colour that ‘draw’ instructions must use and only affects the outline or border of a shape. The default pen colour is black.

Brush

Brush Colour

Formats:

- brush color value

Parameters:

- value: The colour value

Examples:

- brush color pink
- brush color #ff0000
- brush color #ff0000AA

Version: 0.1-alpha and higher

Description: This instruction selects the colour that ‘paint’ instructions must use and only affects the colour of the body of a shape. The default brush colour is black.

Lines

Draw Line with Two Coordinates and an Optional Degree.

Formats:

- draw line [ax, ay] [bx, by] rotation

Parameters:

- ax: X coordinate of the point A of the object.
- ay: Y coordinate of the point A of the object.
- bx: X coordinate of the point B of the object.
- by: Y coordinate of the point B of the object.
- rotation: The optional rotation degree of the object.

Examples:

- draw line [0,-10] [85,120]
- draw line [-10,-10] [50,50] 90

Version: 0.1-alpha and higher

Description: This instruction will draw a line from the first coordinate, to the second coordinate.

This instruction will be affected by Pen instructions. Coordinates must be in the correct format and contain valid values. If a global rotation degree is already set using the Rotation Directive, the drawn shape will be affected. A rotation degree, if specified, will overwrite the global rotation value.

Draw Line with Starting Coordinates, Length and Optional Rotation Degree.

Formats:

- draw line [x, y] length rotation

Parameters:

- x: X coordinate of the starting point of the object.
- y: Y coordinate of the starting point of the object.
- length: The length of the object in pixels.
- rotation: The optional rotation degree of the object.

Examples:

- draw line [10,-10] 100
- draw line [-20,30] 120 -15

Version: 0.1-alpha and higher

Description: This instruction will draw a line from the starting coordinate, for the length value that is specified in the instruction. By default the line will be drawn from the starting point, plus the length value added to the X portion of the starting coordinate e.g. if [0,0] is coordinate and length is 40, the end point of the line will be [40,0].

Valid values for length are from -9999 to 9999. Negative values have the effect of rotating the line by 180 degrees.

This instruction will be affected by Pen instructions. Coordinates must be in the correct format and contain valid values. If a global rotation degree is already set using the Rotation Directive, the drawn shape will be affected. A rotation degree, if specified, will overwrite the global rotation value.

Rectangles

Draw or Paint Rectangle with Centre Coordinate, Width and Height. Optional Degree.

Formats:

- draw box [x, y] width height degree
- paint box [x, y] width height degree

Parameters:

- x: X coordinate of the centre of the object.
- y: Y coordinate of the centre of the object.
- width: The width of the object in pixels.
- height: The height of the object in pixels.
- degree: The optional rotation degree of the object.

Examples:

- draw box [20,30] 150 120
- paint box [20,30] 150 120
- paint box [10,-10] 100 100 -15

Version: 0.1-alpha and higher

Circles

Draw or Paint Circle with Centre Coordinate and Radius.

Formats:

- draw circle [x, y] radius
- paint circle [x, y] radius

Parameters:

- x: X coordinate of the centre of the object.
- y: Y coordinate of the centre of the object.
- radius: The radius of the object in pixels.

Examples:

- draw circle [5,25] 30
- paint circle [0,0] 30

Version: 0.1-alpha and higher

Arcs & Ellipses

Draw or Paint Arc with Centre Coordinate, Two Radiuses and Start and End Points. Optional Degree.

Formats:

- draw arc [x, y] [x radius, y radius] [start, end] rotation
- paint arc [x, y] [x radius, y radius] [start, end] rotation

Parameters:

- x: X coordinate of the centre of the object.
- y: Y coordinate of the centre of the object.
- x radius: X radius of the object.
- y radius: Y radius of the object.
- start: The starting point of the object.
- end: The end point of the object.
- rotation: The optional rotation degree of the object.

Examples:

- draw arc [-100,0] [100,50] [0,75]
- paint arc [100,0] [50,50] [75,25]
- draw arc [-100,0] [100,50] [0,75] -15

Version: 0.1-alpha and higher

Description: The starting point of the arc from 0 to 100. 0 equals to 0 degree angle on the arc. 100 equals to the 360 degree angle on the arc.

Triangles

Draw or Paint Triangle with Centre Coordinates, Base and Height. Optional Degree.

Formats:

- draw triangle [x, y] base height rotation
- paint triangle [x, y] base height rotation

Parameters:

- x: X coordinate of the centre of the object.
- y: Y coordinate of the centre of the object.
- base: The base of the object, in pixels.
- height: The height of the object, in pixels.
- rotation: The optional rotation degree of the object.

Examples:

- draw triangle [20,30] 80 40
- paint triangle [0,0] 60 60
- paint triangle [-100,0] 60 60 90

Version: 0.1-alpha and higher

Draw or Paint Triangle with Three Coordinates. Optional Degree.

Formats:

- draw triangle [ax, ay] [bx, by] [cx, cy] rotation
- paint triangle [ax, ay] [bx, by] [cx, cy] rotation

Parameters:

- ax: X coordinate of the point A of the object.
- ay: Y coordinate of the point A of the object.
- bx: X coordinate of the point B of the object.
- by: Y coordinate of the point B of the object.
- cx: X coordinate of the point C of the object.
- cy: Y coordinate of the point C of the object.
- rotation: The optional rotation degree of the object.

Examples:

- draw triangle [20,140] [90,40] [-100,-30]
- paint triangle [0,0] [60,60] [60,0]
- paint triangle [0,0] [-60,-60] [-60,0] 15

Version: 0.1-alpha and higher

Artwork Signature

Sign Paper with Colour and Text

Formats:

- sign colour text

Parameters:

- colour: The colour of the signature.
- text: The text of the signature of any length.

Examples:

- sign white My Name, 7 Dec, 2024

Version: 0.1-alpha and higher

Description: This instruction writes a text signature at the bottom right corner of the paper.

Built-in Functions

Calc

Formats:

- calc(expression)

Parameters:

- expression: A maths expression

Examples:

- paper size $width calc($width / 4)
- pen size calc($size / 2 + 1)
- draw line [0,0] calc(($width / 5) + ($height / 6))

Version: 0.1-alpha and higher

Description: In order to do maths inside Nini programs, ‘calc’ function can be used. The ‘calc’ function cannot be nested e.g. calc(1 + calc(2 ** 3)), but nested parentheses could be used, which has the same effect.

Note: Due to some technical limitations in the way instructions are parsed in Nini, currently only one ‘calc’ function can be used in each instruction. For example ‘calc(..) calc(..)’ in a single instruction will not work. To overcome this limitation, declare multiple variables separately and use the ‘calc’ function on each, then use the variables name in the instruction.

Global Variables

Paper Dimensions

$PW: Paper width in pixels e.g. 800
$PH: Paper height in pixels e.g. 400

Top-left Coordinates

$TLX: X value of the top-left corner of the paper e.g. -400
$TLY: Y value of the top-left corner of the paper e.g. 200
$TL: Top-left coordinates of the paper e.g. [$TLX, $TLY]

Top-centre Coordinates

$TCX: X value of the top-centre of the paper e.g. 0
$TCY: Y value of the top-centre of the paper e.g. 200
$TC: Top-centre coordinates of the paper e.g. [$TCX, $TCY]

Top-right Coordinates

$TRX: X value of the top-right corner of the paper e.g. 400
$TRY: Y value of the top-right corner of the paper e.g. 200
$TR: Top-right coordinates of the paper e.g. [$TRX, $TRY]

Bottom-left Coordinates

$BLX: X value of the bottom-left corner of the paper e.g. -400
$BLY: Y value of the bottom-left corner of the paper e.g. -200
$BL: Bottom-left coordinates of the paper e.g. [$BLX, $BLY]

Bottom-centre Coordinates

$BCX: X value of the bottom-centre of the paper e.g. 0
$BCY: Y value of the bottom-centre of the paper e.g. -200
$BC: Bottom-centre coordinates of the paper e.g. [$BCX, $BCY]

Bottom-right Coordinates

$BRX: X value of the bottom-right corner of the paper e.g. -400
$BRY: Y value of the bottom-right corner of the paper e.g. -200
$BR: Bottom-right coordinates of the paper e.g. [$BRX, $BRY]

Left-centre Coordinates

$LCX: X value of the left-centre side of the paper e.g. -400
$LCY: Y value of the left-centre side of the paper e.g. 0
$LC: Left-centre coordinates of the paper e.g. [$LCX, $LCY]

Right-centre Coordinates

$RCX: X value of the right-centre side of the paper e.g. 400
$RCY: Y value of the right-centre side of the paper e.g. 0
$RC: Right-centre coordinates of the paper e.g. [$RCX, $RCY]