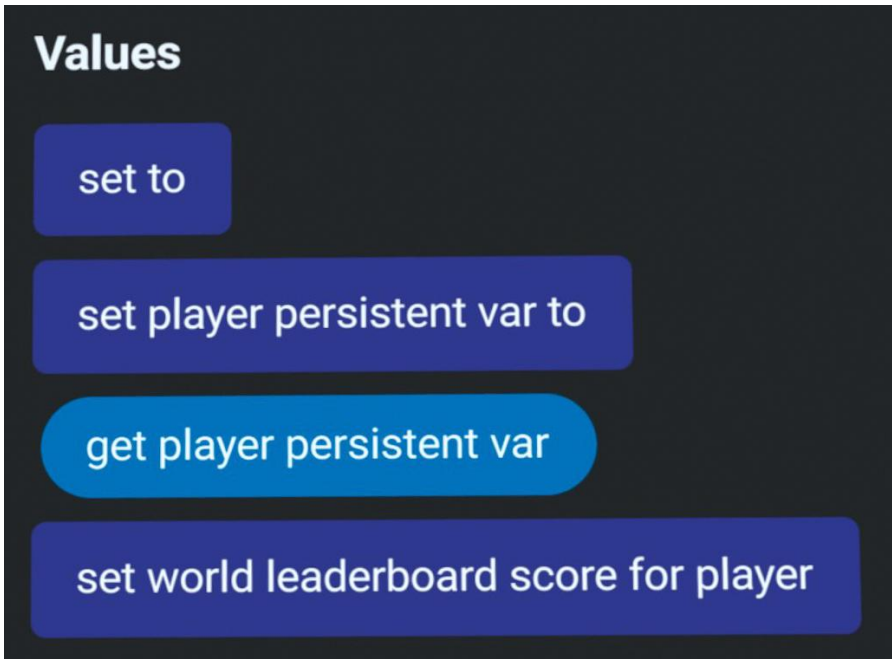
Values

Values

Values > Values



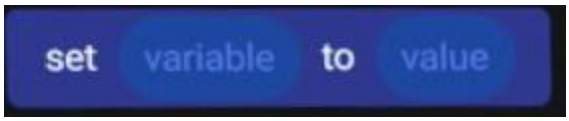
Codeblocks that set and get variable values.

set to

Values > Values > set to

Set the value of a variable.

Appearance in Composition Pane

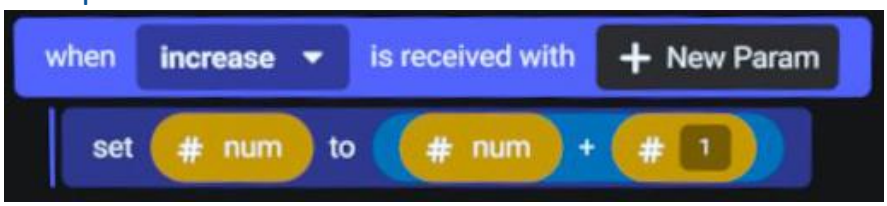


Description

set to is used to change the value of a variable. Any variable type may be passed in as the first parameter, but its value in the second parameter must match the variable type of the first parameter. For example, attempting to set a string to a boolean will throw an error unless the boolean is first typecast into a string with the **variable as string** codeblock.

It is also possible to set variables to equal other variables of the same type. Please note that variables cannot share the same name inside of the script. This is true even if the variables have different types.

Example 1: Increment a number



When the event *increase* is received, **set** the number *num* to equal *num* plus one.

set player persistent variable to

Values > Values > set player persistent variable to

Sets a persistent player variable.

Appearance in Composition Pane

Description

A Persistent Player Variable is a number variable that is saved between sessions of your world and is tied to a player ID. A great example is saving a player's highscore, for when they return to the world.

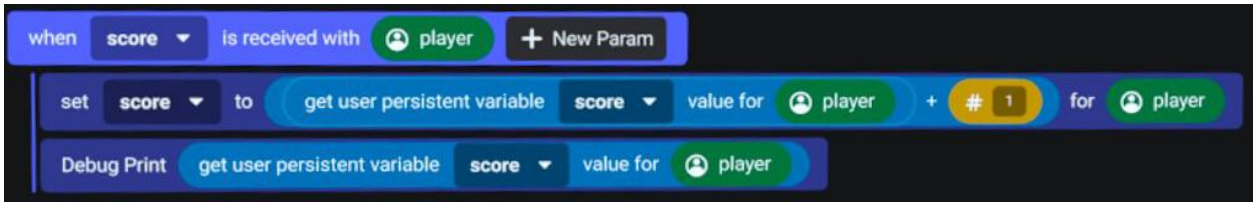
set player persistent variable to saves the number value for it to be read later. They can be created from the player settings tab in your build menu. Has an initial value of zero. Once you have created the variable, you can select it by clicking on the drop down arrow next to changeMe.

Parameters

persistent player variable, number, player

Sets the persistent player variable to the number for that player.

Example 1: Increment a player's score



When score is received with player, we increase the player's score by one, and Debug Print the new value.

See Also

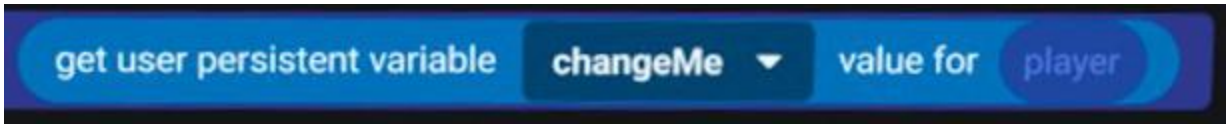
- Values > Values > get player persistent var

get player persistent var

Values > Values > get player persistent var

Retrieves a persistent player variable.

Appearance in Composition Pane



Description

A Persistent Player Variable is a number variable that is saved between sessions of your world and is tied to a player ID.

get player persistent var, is used in the pill slot of other codeblocks, and will then return the specified player variable for use.

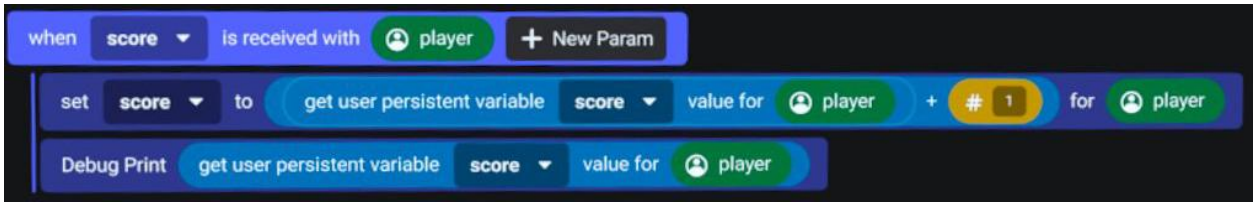
The default is set to *changeMe*. From the drop down, you can choose the variable you would like to use. See “set player persistent variable” for how to create player variables.

Parameters

persistent player variable, player

Returns the value of the selected player’s persistent variable.

Example 1: Increment a player’s score



When score is received with player, we increase the player’s score by one, and Debug Print the new value.

See Also

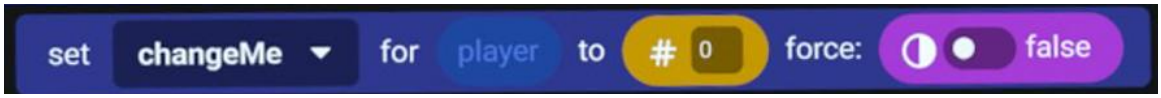
- Values > Values > set player persistent variable to

set world leaderboard score for player

Values > Values > set world leaderboard score for player

Used for updating a player's leaderboard variable.

Appearance in Composition Pane



Description

You can have up to three leaderboard scores in your world, they can only be created by the owner of the world. And can be created from your build menu > settings > leaderboard.

When creating the variable, you can set it to be either descending or ascending. Descending means that higher values are preferred, and ascending is the inverse. This also determines how the scores are displayed on the leaderboard gizmo.

Leaderboards are a great way to give users a look at their score compared to others and compete for first place.

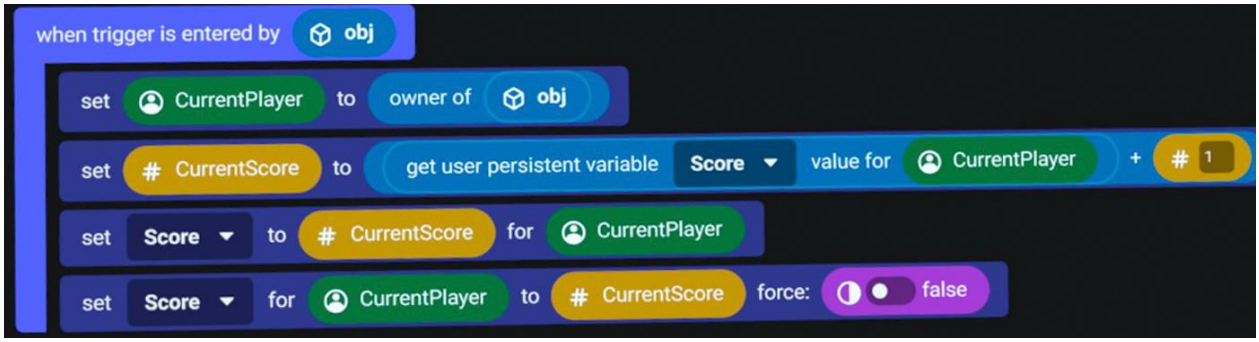
When setting the leaderboard variable, you will need to specify the variable that is being set, as well as the player, and a number. The boolean for Force, determines whether the score is forced to update, or if it must be a higher or lower score to be updated.

Parameters

leaderboard variable, player, number, boolean

Sets the leaderboard variable of the player to the number if true. And if boolean false, only updates the number when the value is higher or lower, depending on whether it is set to descending or ascending respectively.

Example 1: Track All Time Baskets



When the hoop trigger is entered by an obj, we set the CurrentPlayer to Owner of object. Then we set the *CurrentScore* number variable to get player var Score + 1. We then update the player variable and the leaderboard. Because the leaderboard is set to descending, we do not need to force the score to update as we know it will always be higher than the last time we set it.

See Also

- Values > Values > get player persistent var
- Values > Values > set player persistent variable to

Debugging

Values > Debugging

Codeblocks that aid in debugging.

debug print

Values > Debugging > Debug Print

Outputs a string message to the debug console in the script tab of your build menu.

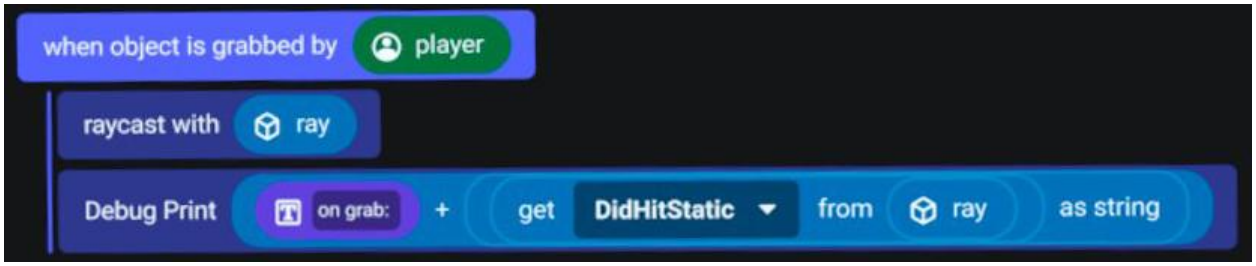
Appearance in Composition Panel

Description

Debug Print is a great way to see what is happening in your script. If you are experiencing a bug, it can allow you to check variables, or see if an event is being received.

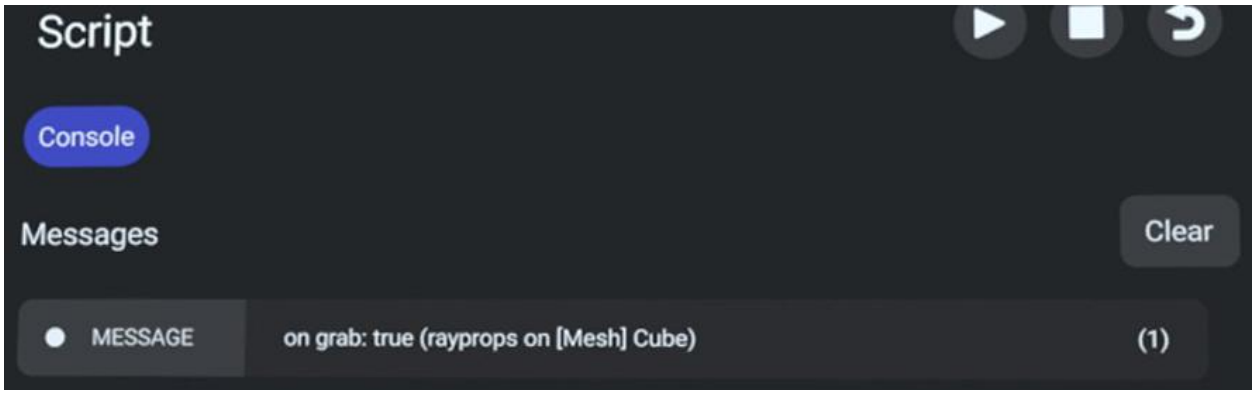
When an action isn't going off, consider following your chain of events to determine where the issue is originating. In Example 1, you can see that we debug print the raycast's "DidHitStatic." This will return either true or false. If we expected it to return true, and it is false, then the bug may be with our raycast gizmo's orientation or it's setup inside the gizmo's property panel. If the debug print is not received at all, then the issue may be with our object's settings.

Example 1: Debug Print a raycast



In the example above, we fire our raycast *ray*, then **debug print** if it *DidHitStatic*.

get raycast data here retrieves a boolean to determine if the raycast *ray* has hit a static entity. That value is converted to a string with **variable as string**, connected to the "on grab:" **string input** with a plus operator. The **debug print** codeblock, outputs the string to the debug console, found in your build menu, as seen below:



The output message includes the string sent by **debug print**, the script's name, and the object to which the script is attached. The number on the far right will increment if the same message outputs more than once in a row.

See Also

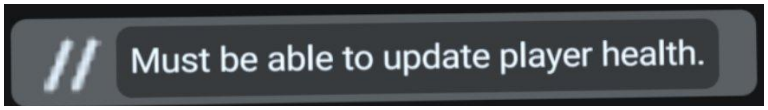
- Values > Type Casting > variable as string

comment

Values > Debugging > Comment

Stores a string message.

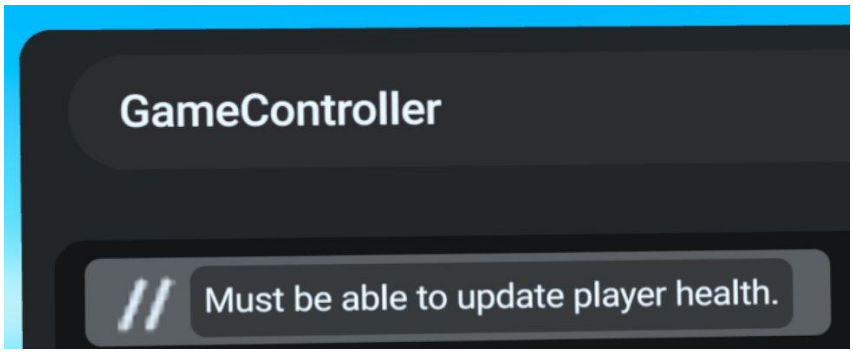
Appearance in Composition Panel



Description

Leaving comments in your scripts can be a great way to leave yourself a reminder as to why something was done, or for anyone who might look at the script later. Comments are also great for noting separate parts of a larger process. Perhaps a food script might have a section about returning the object to its original position, and another section about being able to be eaten.

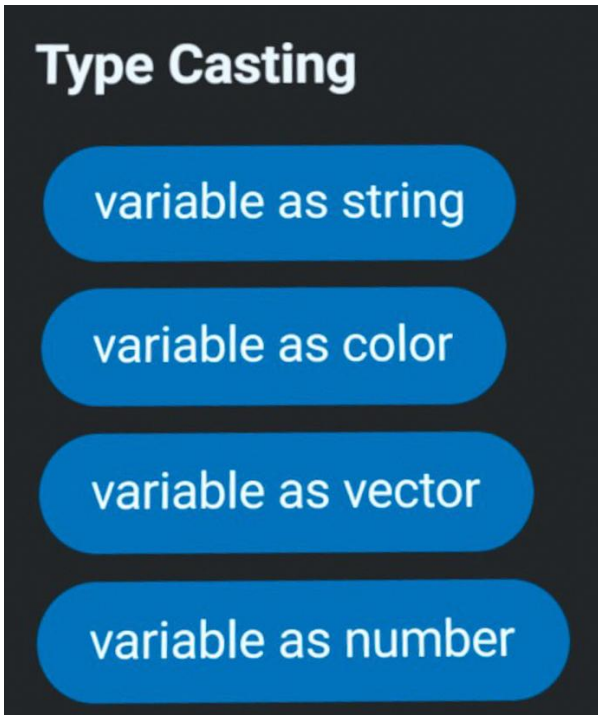
Example 1: Preparing To Script



In this example we have left ourselves a reminder that we will be using our controller to track player health.

Type Casting

Values > Type Casting



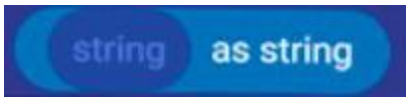
Codeblocks that convert a variable to another value type.

variable as string

Values > Type Casting > variable as string

Changes a non-string variable into a string.

Appearance in Composition Pane



Description

Converts a variable or a value into a string for use in other codeblocks.

Variable as string is local to where it is used, meaning that it will not change the variable's type on the Variables tab.

Parameters

number, boolean, object, player, vector, rotation, color, string
: Converts the value into a string and returns the string.

list
: Converts a list into comma separated values, and returns the string.

Example 1: Convert a boolean into a string

This will Debug Print "on grab:" followed by either "true" or "false."

See Also

- Values > Debugging > debug print
- Values > Values > set player persistent variable to
- Values > Values > get player persistent var

Values > Value Input > string input

variable as color

Values > Type Casting > variable as color

Changes a vector or number into a color.

Appearance in Composition Pane

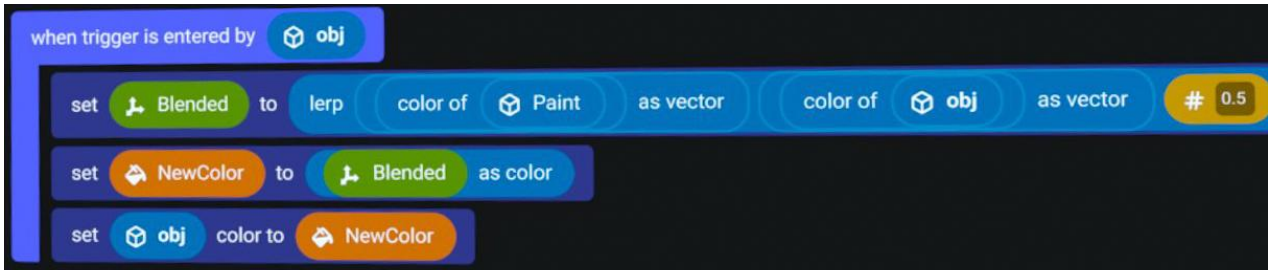
Description

Variable as color is a great way to convert a vector into a color, where (x,y,z) is turned into (r,g,b). Remember that colors range from 0 to 1. And if you use a number value rather than a vector, it will be input into all three RGB slots. Ie. 0.5 => (0.5,0.5,0.5).

Parameters

vector
number

Example 1: Blend / Mix Two Colors



When the tip of the “Brush” enters the trigger, we start by setting the vector *Blended* to be the lerp between (color of *Paint)* and the color of the obj that entered the trigger. We use 0.5 to get a color in the middle. We can then set NewColor to be Blended as a color. And apply that color to the object that entered the trigger. Adjusting 0.5 changes how intense the color change is.

See Also

- Operators > Basic Math > lerp
- Values > Type Casting > variable as vector

variable as vector

Values > Type Casting > variable as vector

Changes a color or number into a vector.

Appearance in Composition Pane

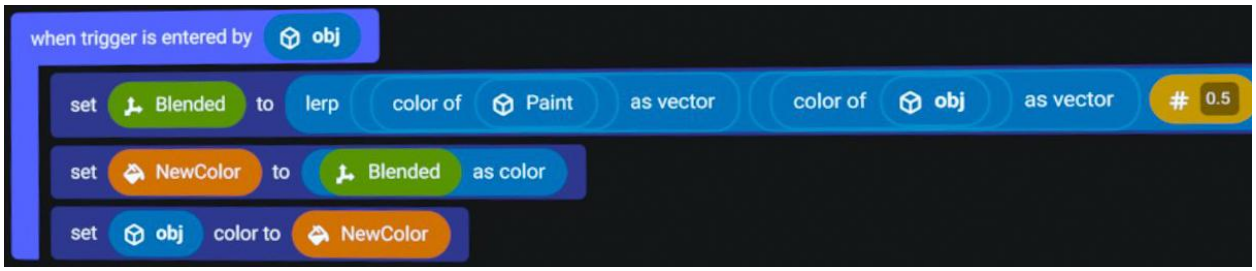
Description

Variable as vector is a great way to convert a color into a vector, where (r,g,b) is turned into (x,y,z). Remember that colors range from 0 to 1. And if you use a number value rather than a color, it will be input into all three XYZ slots. Ie. 15 => (15,15,15).

Parameters

color
number

Example 1: Blend / Mix Two Colors



When an object enters the trigger, we start by setting *Blended* to be the lerp between ((color of *Paint)* as vector) and ((color of the obj) as vector). We use 0.5 to get a color in the middle. Then set NewColor to be Blended as a color. And apply that color to the object.

See Also

- Operators > Basic Math > lerp
- Values > Type Casting > variable as color

variable as number

Values > Type Casting > variable as vector

Changes a boolean or numerical string into a number.

Appearance in Composition Pane

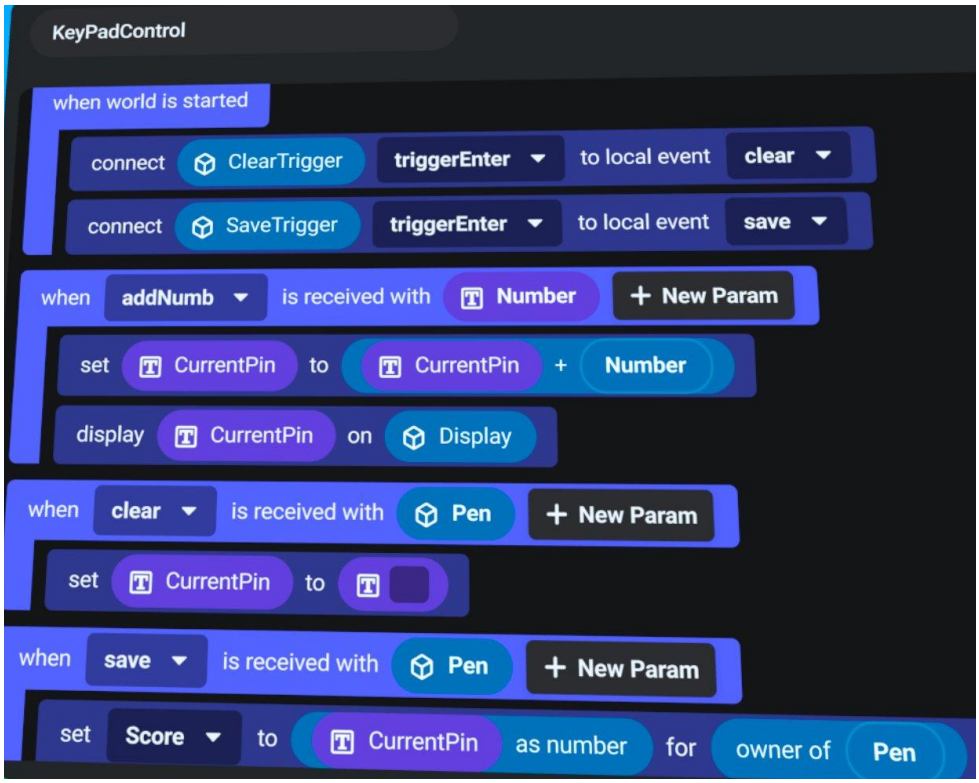
Description

Variable as number is a great way to convert a boolean (false/true) into a number (0/1). It can also be useful for converting a numerical string into a number.

Parameters

boolean, string

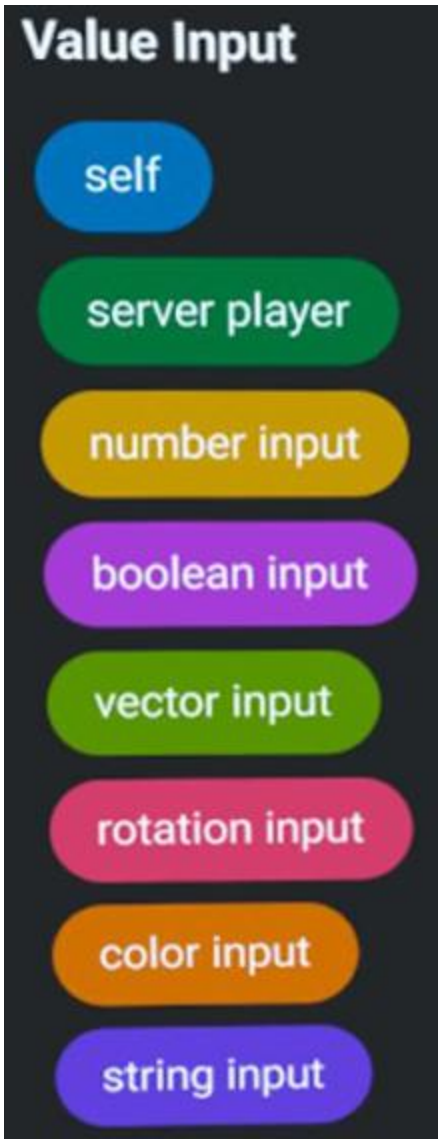
Example 1: Save A Pin



If you create a keypad that concatenates string number values to form a pin, when the user puts the pen object in the save trigger, you can then save that pin as a player variable. This would allow the player to have their own private pin saved for their next return to your world.

Value Input

Values > Value Input



Value inputs used in events and codeblocks.

self

Values > Value Input > self

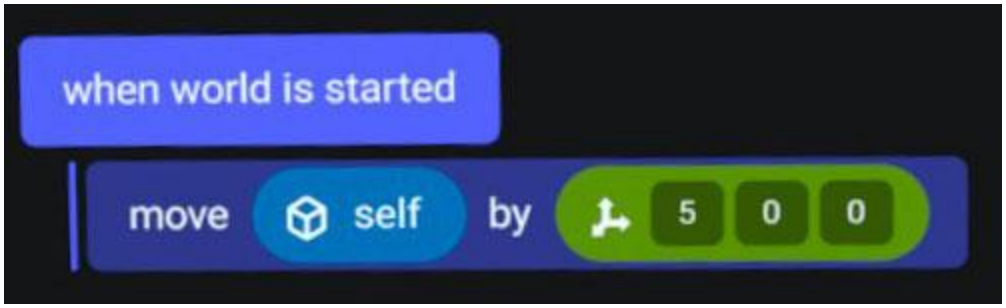
A variable representing the object to which the script is attached.

Appearance in Composition Pane

Description

A script must be attached to an object. **self** is an easy way to reference the object the script is attached to.

Example 1: Move object



When world is started, the object *self* is instantly moved by 5,0,0.

server player

Values > Value Input > server player

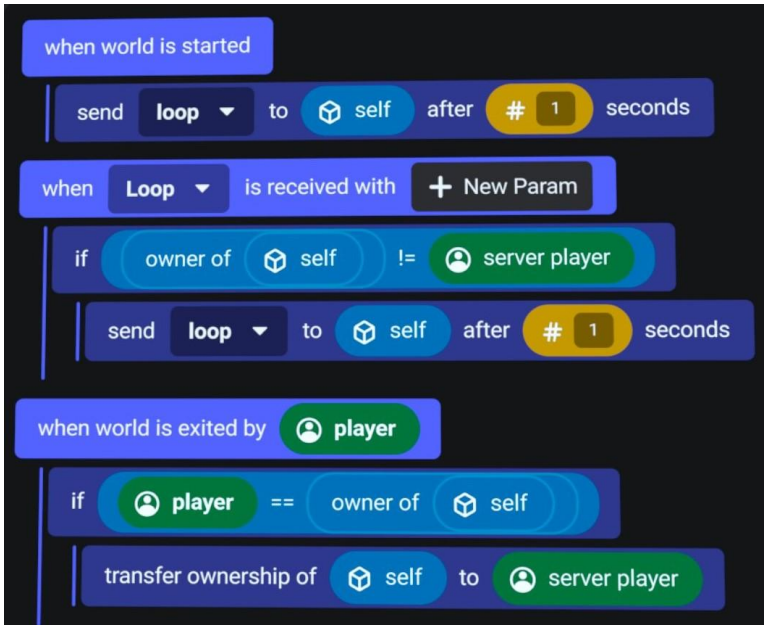
Represents the owner of any object or script that has not been assigned.

Appearance in Composition Pane

Description

Server player is used when you need to transfer ownership of an object back to the server. It's also great for use in comparisons to find out if an object is owned by the server or a player.

Example 1: Loop Comparison



Local scripts run **When World Is Started** when their object is assigned to a player. Then we loop the event when the object is no longer owned by the server player. It is useful to transfer ownership of the object back to the server player, when the player exits the world.

See Also

- Actions > Player > transfer ownership of object to player
- Actions > Player > owner of object

number input

Values > Value Input > number input

Allows you to input a number directly into a codeblock.

Appearance in Composition Pane

Description

Number Input can be used to enter a number directly into a codeblock similar to how we can use a number variable.

Example 1: Set a variable to a number value

When world is started, set the *var* number variable to 2.

See Also

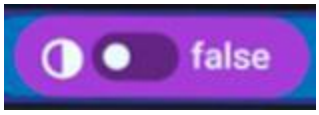
- Values > Values > set to

boolean input

Values > Value Input > boolean input

Allows you to input a boolean value directly into a codeblock.

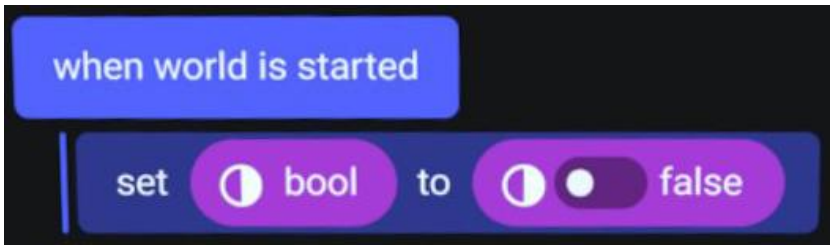
Appearance in Composition Pane



Description

Boolean Input allows you to enter a true or false value directly into a codeblocks.

Example 1: Set a bool variable to a boolean value



When world is started, set *bool* to boolean input false.

See Also

- Values > Values > set to

vector input

Values > Value Input > vector input

Allows you to input a vector value directly into a codeblock.

Appearance in Composition Pane

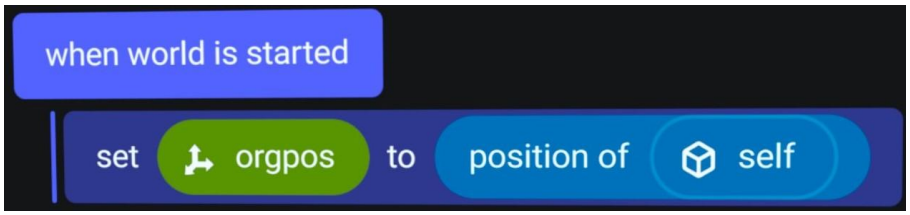
Description

Vector Input can be used to enter an X, Y, and Z value directly into a codeblock.

Their use depends on the context, a few common vector types are:

Position: the coordinates of a position relative to the world origin.
Direction: A direction in space with the values representing its length in each direction.
Scale: The size of an object in each direction relative to a different size, typically a meter.
Velocity: The change in position of an object over one second.
Angular velocity: the change in rotation of an object over one second.

Example 1: Save the objects original position



When world is started, we set the OrgPos vector variable to the position of self.

See Also

- Values > Values > set to

rotation input

Values > Value Input > rotation input

Allows you to input a rotation value directly into a codeblock.

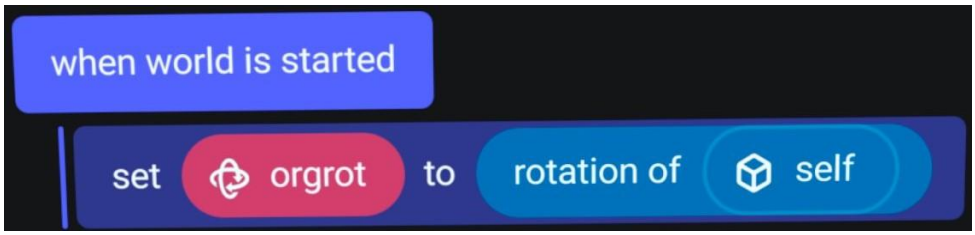
Appearance in Composition Pane

Description

Rotation Input can be used to enter a rotation directly into a codeblock.

Rotations are based on the direction each axis is facing. The values represent the pitch, yaw, and roll of an object. Rotating the x-axis of an object changes its pitch, rotating the y-axis changes its yaw, and rotating its z-axis changes its roll. Rotation is measured in degrees, from 0 to 360. Any negative value input automatically changes to a positive value that is 360 minus the absolute value.

Example 1: Save the objects original rotation



When world is started, we set the OrgRot rotation variable to the rotation of self.

See Also

- Values > Values > set to

color input

Values > Value Input > color input

Allows you to input a color value directly into a codeblock.

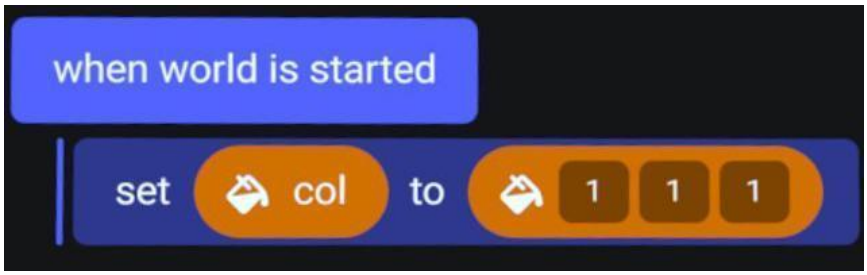
Appearance in Composition Pane

Description

Color input can be used to enter an RGB value directly into a codeblock.

The values in a color variable represent how much red, green, and blue (RGB) are in a color. Values are between 0 and 1, with (0,0,0) being black and (1,1,1) being white. Different combinations of red, green, and blue can create many possible colors.

Example 1: Set a variable to a color value



When world is started, we set the color variable *col* to white or (1,1,1).

See Also

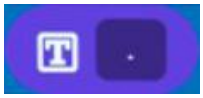
- Values > Values > set to

string input

Values > Value Input > string input

Allows you to input a string value directly into a codeblock.

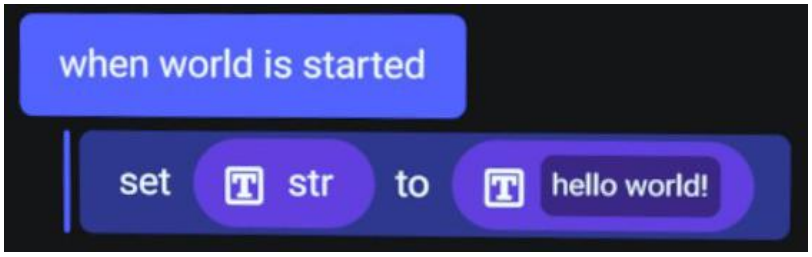
Appearance in Composition Pane



Description

string input can be used to enter a string value directly into a codeblock.

Example 1: Set a variable to a string value



When world is started, set the *str* string variable to “hello world*!*”

See Also

- Values > Values > set to

Constants

Values > Constants

Constant mathematical values, great for use in calculations and equations.

pi

Values > Constants > pi

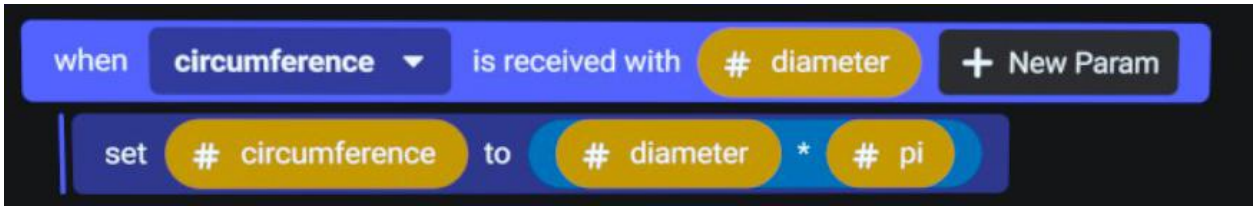
A number value set to the number pi.

Appearance in Composition Pane

Description

Pi is the constant which can be calculated from the ratio of the circumference of a circle to its diameter, which in this case is as accurate as 3.14159.

Example 1: Calculate a circle's circumference



When the event *circumference* is received with a *diameter* as the parameter, **set** the variable *circumference* to *diameter* times **pi**.