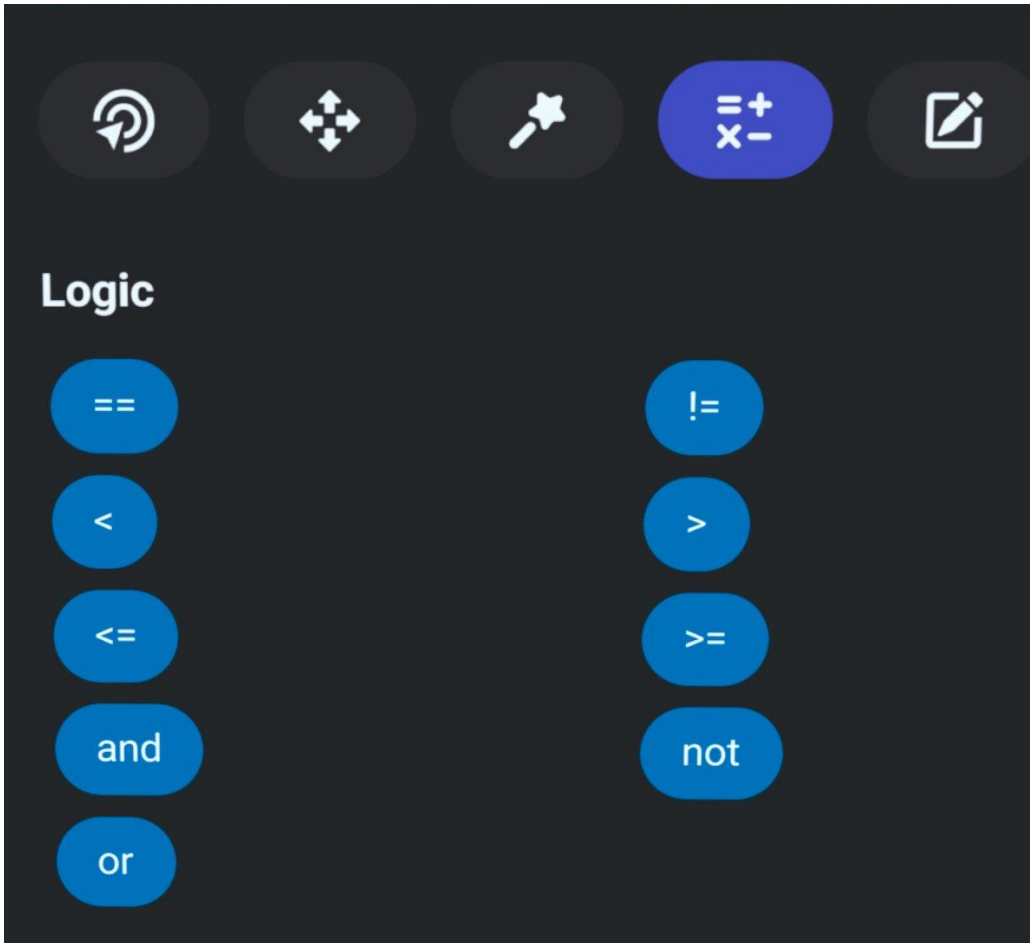
Operators

Logic

Operators > Logic



Used to compare values, returns true or false. Often used in Control Events; If, Else, and While.

==

Operators > Logic > ==

Equal To

Evaluates true if values A and B are equivalent.

Appearance in Composition Pane

Description

Compares the values in A and B and returns true if they are equivalent and false if they are not..

Parameters

Number, **boolean**, **object**, **player**, **vector**, **rotation**, **color**, **string**

Returns boolean true if the values in A and B are equivalent. Will return an error if there is a type mismatch in the A and B slots.

List variables

Returns the boolean true if the lists in A and B are equivalent.

Example 1: Show trophy

When *flagcapture* is received, *n* is increased by 1. Then, if *n* equals 3, the *trophy* is shown.

See Also

- Events > Events > If

!=

Operators > Logic > !=

Not Equal To

Evaluates to true if values A and B are not equivalent.

Appearance in Composition Pane

Description

Compares the values in A and B and returns true if they are not equivalent.

Parameters

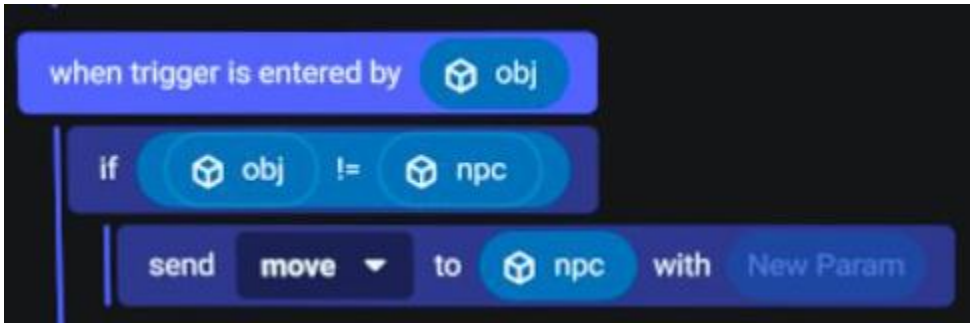
Number, **boolean**, **object**, **player**, **vector**, **rotation**, **color**, **string**

Returns boolean true if the values in A and B are not equivalent. Will return an error if there is a type mismatch in the A and B slots.

List variables

Returns the boolean true if the lists in A and B are not equivalent.

Example 1: Others Move Me



When an *obj* that is not the *npc* enters the trigger, it sends the *move* event to *npc*.

See Also

- Events > Events > If

<

Operators > Logic > <

Less Than

Evaluates to true if the number A is less than the number B.

Appearance in Composition Pane

Description

Returns the boolean true if the number in the A slot is less than the number in B.

Parameters

Number, number

Example 1: Hide all of the objects in a list

Our while loop iterates through the list of objects by adding one to the iterator. When the iterator equals the length of the list, the while loop stops running.

See Also

- Events > Events > While

>

Operators > Logic > >

Greater Than

Evaluates to true if the number A is greater than the number B.

Appearance in Composition Pane

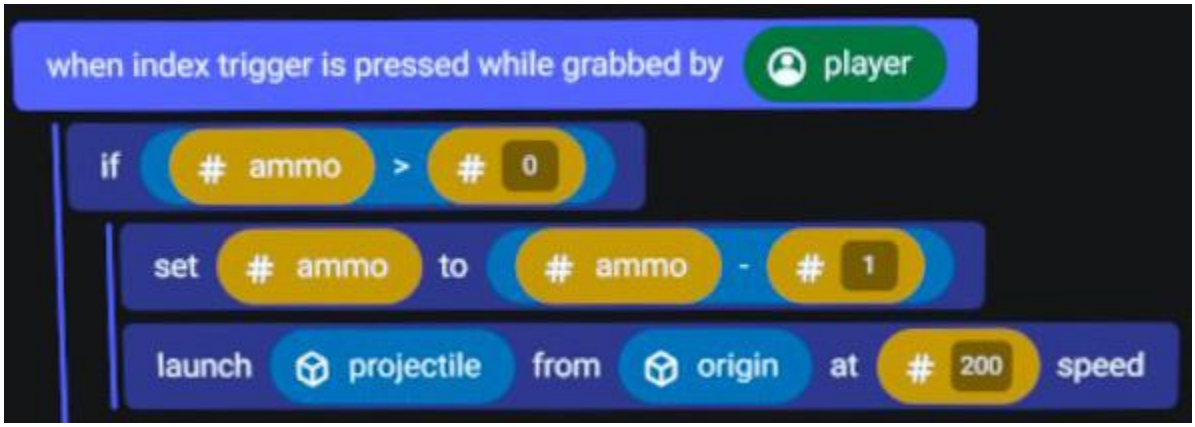
Description

Returns true if the number in A is greater than the number in B.

Parameters

number, number

Example 1: Launch a projectile if there is ammo available



When a player presses their index trigger, if *ammo* is greater than 0, it launches the *projectile* and decreases the *ammo* by 1. *Ammo* can be reset by another event.

See Also

- Events > Events > If

<=

Operators > Logic > <=

Less Than Or Equal To

Evaluates to true if the number A is less than or equal to the number B.

Appearance in Composition Pane

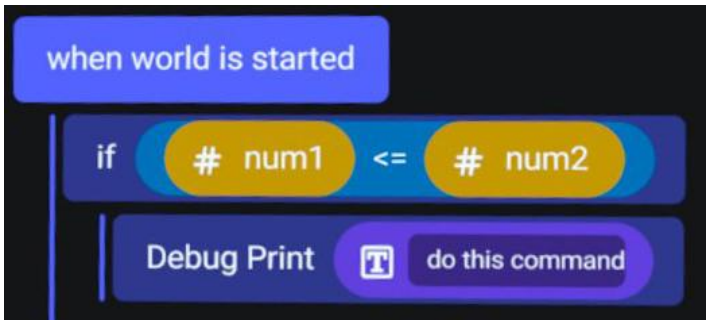
Description

Returns true if the number in A is less than or equal to the number in B.

Parameters

number, number

Example 1: Compare two numbers



The Debug Print codeblock will run if *num1* is less than or equal to *num2*.

>=

Operators > Logic > >=

Greater Than Or Equal To

Evaluates to true if the number A is greater than or equal to the number B.

Appearance in Composition Pane

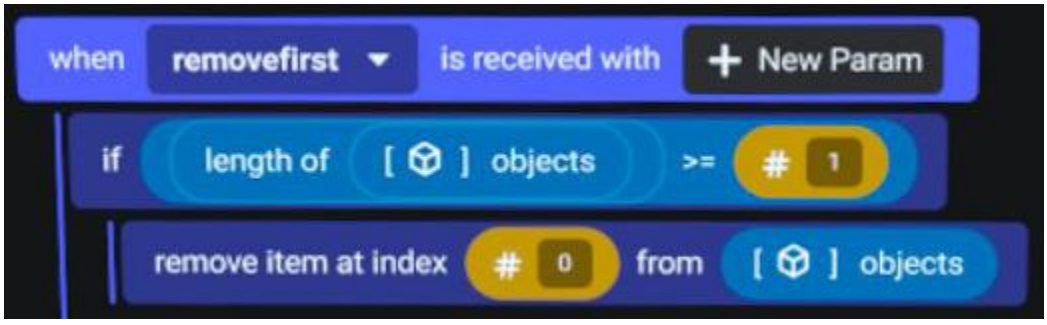
Description

Returns true if the number in A is greater than or equal to the number in B.

Parameters

number, number

Example 1: Remove Item Zero



When the *removefirst* event is received, if the length of the *objects* list is greater than or equal to 1, the first item in the list is removed. Please note that when a list has length of one, the index of that item is zero. The highest item index is always one less than the list length.

See Also

- Events > Events > If

and

Operators > Logic > and

Evaluates to true if the statements in A and B are both true.

Appearance in Composition Pane

Description

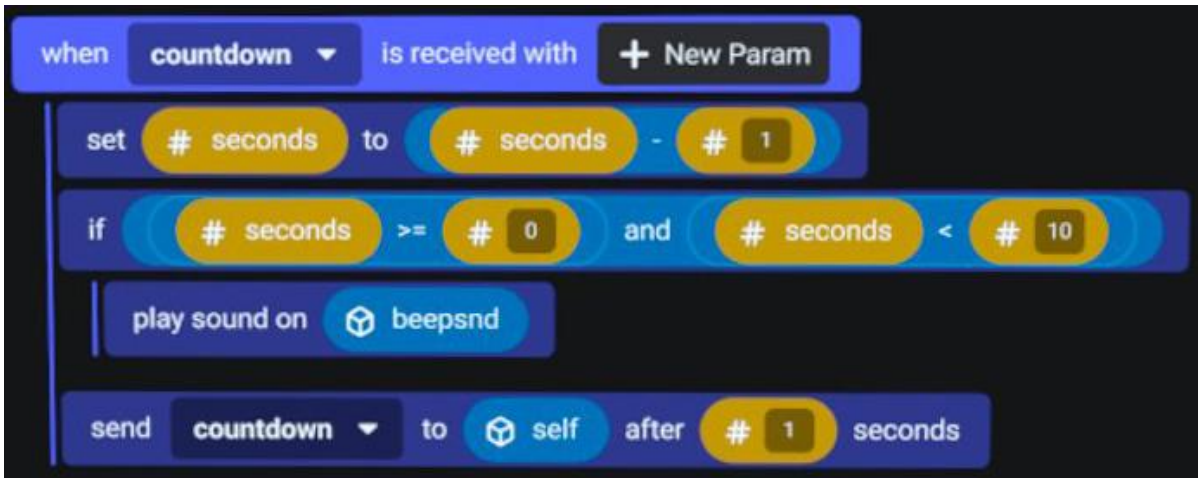
Returns true if the values and or logic in both A and B are true.

Parameters

boolean, boolean

Returns the boolean true if the boolean in A is true and the boolean in B is true.

Example 1: Final Countdown Beeping



When the *countdown* event is received, it decreases *seconds* by 1. Then, if *seconds* is greater than or equal to 0 and less than 10, the statement is true and it plays a beep sound.

not

Operators > Logic > not

Reverses a boolean value.

Appearance in Composition Pane

Description

The **not** operator toggles a boolean variable, either from true to false or from false to true.

Parameters

boolean

Example 1: Run if not true

Not (*boolean***)** is used within an **if** to test a condition. When the *boolean* is false, **if** will run.

Example 2: Flip a boolean

Setting a boolean variable to not bool will change a true bool to false or a false bool to true.

See Also

- Events > Events > If

or

Operators > Logic > or

Evaluates to true if either A or B is true.

Appearance in Composition Pane

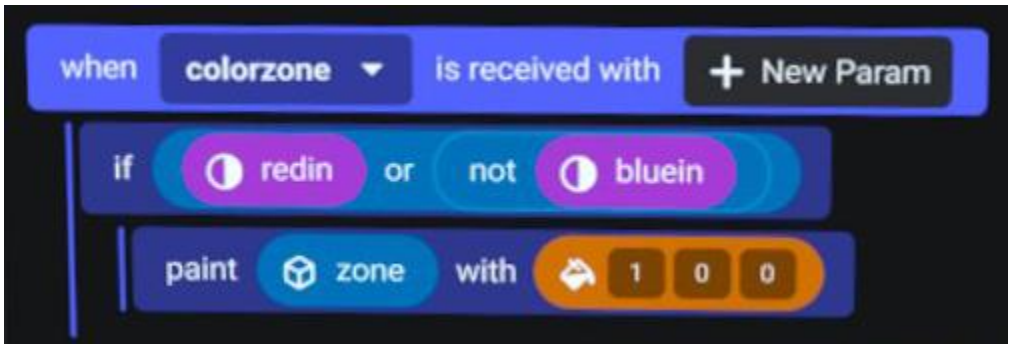
Description

Returns true if either A or B is true.

Parameters

boolean, boolean

Example 1: Paint Red



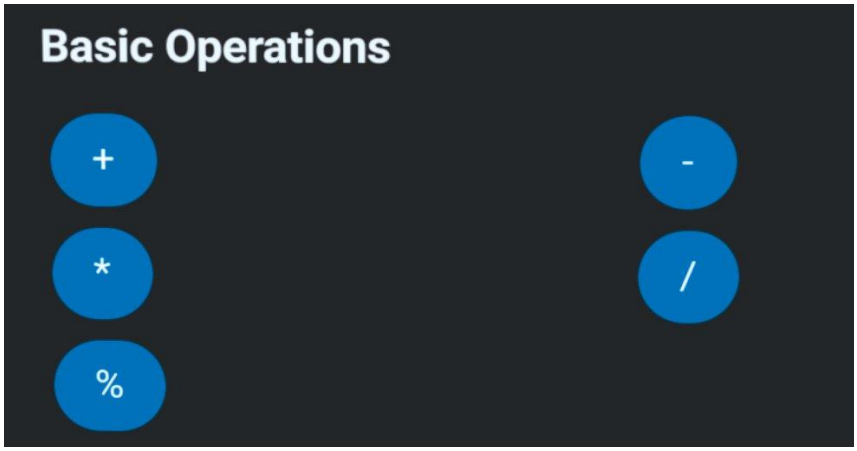
When the *colorzone* event is received, if *redin* is true, or *bluein* is false, the *zone* is painted red.

See Also

- Events > Events > If

Basic Operations

Operators > Basic Operations



Basic mathematical operators used in various expressions.

+

Operators > Basic Operations > +

Plus

Adds two values together.

Appearance in Composition Pane

Description

Typically returns the sum of A and B. The return value type depends on what values are used.

Parameters

number, number

Adds number A and number B to return the sum of A and B.

vector, vector

Adds vectors A and B. The output of (Ax,Ay,Az) + (Bx,By,Bz) is (Ax+Bx,Ay+By,Az+Bz).

rotation, rotation

Rotations are applied from right to left, this means order does matter. If you want to rotate an object along its local axis, consider example 2, where the *OrgRot* comes first. Notice when OrgRot is in the second slot it creates a rotation along the world's axis.

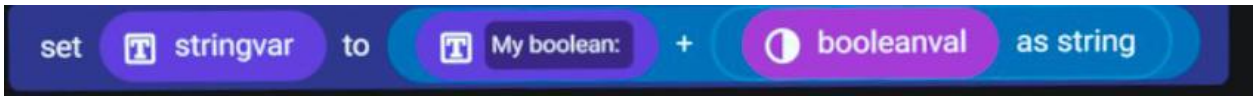
color, color

Sums the R, G, and B values from colors A and B to return a new color.

string, string

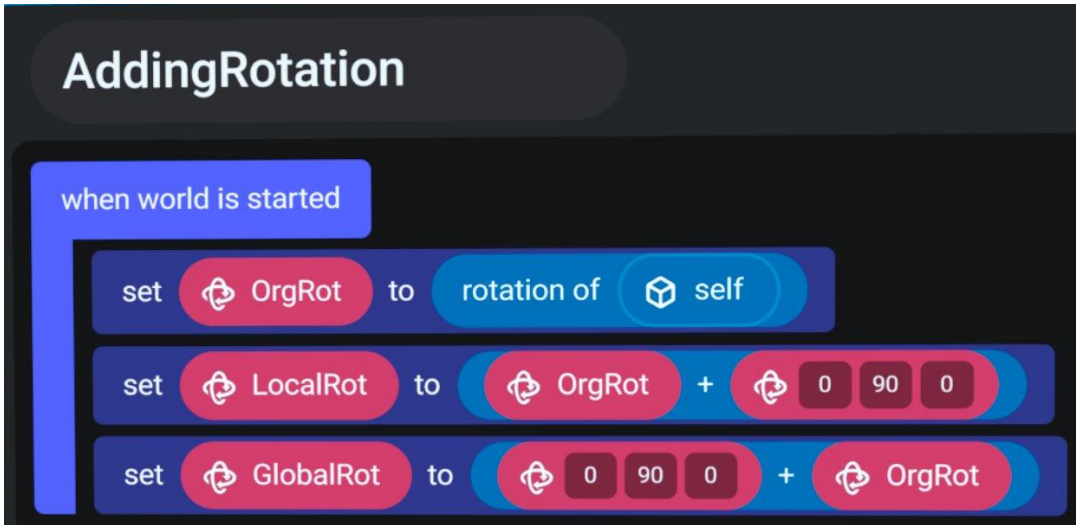
Combines strings A and B to return a new string.

Example 1: Concatenate strings



We set *stringvar* to "My Boolean:" plus *booleanval* as string. When the bool is true, it returns "My boolean: true."

Example 2: Adding Rotations



When the world is started we save the rotation of self as *OrgRot*. We can then calculate a 90 degree rotation along the y axis of the object using set *LocalRot* to *OrgRot* plus 0,90,0. If we reverse the order, we get a rotation along the world's y axis.

See Also

- Values > Debugging > debug print
- Values > Value Input > string input

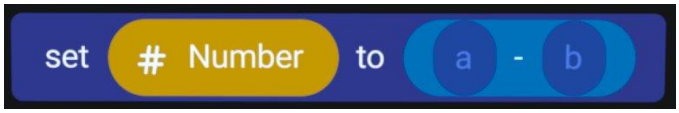
-

Operators > Basic Operations > -

Minus

Subtracts the value B from A.

Appearance in Composition Pane



Description

Returns the subtraction calculation of A minus B. Return value type depends on values used.

Parameters

number, number

Subtracts number B from number A.

vector, vector

Subtracts vector B from A. The output of (Ax,Ay,Az) - (Bx,By,Bz) is (Ax-Bx,Ay-By,Az-Bz).

rotation, rotation

Rotations are applied from right to left: meaning order matters. See Plus for an example.

color, color

Subtracts the R, G, and B values of B from A to return a new color.

Example 1: Countdown

S*econds is* decreased by 1 as 1 is subtracted from its current value.

See Also

- Operators > Basic Operations > +

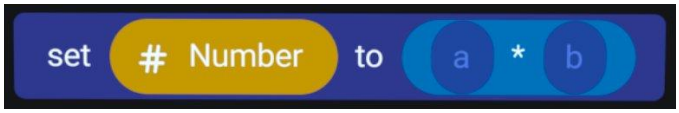
*

Operators > Basic Operations > *

Multiply

Multiplies A and B.

Appearance in Composition Pane



Description

Returns A times B. The returned value type depends on the values used.

Parameters

number, number

Returns number A times number B.

vector, vector

Multiples vector A times B. The result of (Ax,Ay,Az) * (Bx,By,Bz) is (Ax*Bx,Ay*By,Az*Bz). Multiplying vectors can be used to adjust individual values of the vector. To double the y of a vector, multiply the by (1,2,1). To zero out the y-coordinate, multiply by (1,0,1).

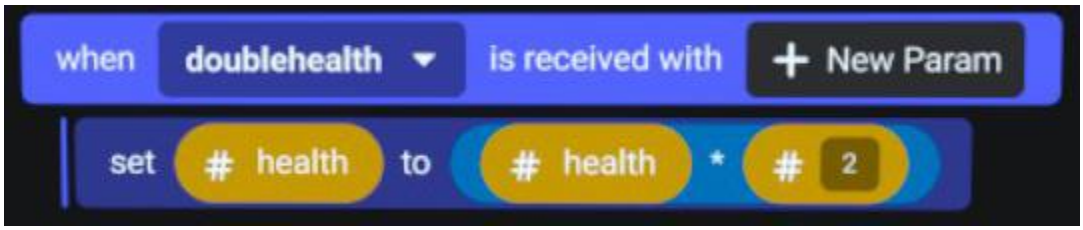
rotation, vector

If a rotation is multiplied by a vector, it rotates the vector, returning a new vector.

vector, number | number, vector

A vector multiplied by a number, returns a vector with each value scaled by the number.

Example 1: Double health



When the *doublehealth* event is received, the *health* variable is multiplied by 2.

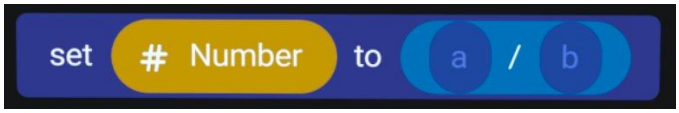
/

Operators > Basic Operations > /

Divide

Divide A by B.

Appearance in Composition Pane



Description

Returns A divided by B. Return value type depends on values used.

Parameters

number, number

Returns a number that is number A divided by number B.

vector, vector

Dividing vector A by B. The result of (Ax,Ay,Az) / (Bx,By,Bz) is (Ax/Bx,Ay/By,Az/Bz). Dividing vectors can be used to adjust individual values of the vector. If any of the values in B are 0, a divide by zero error will occur.

vector, number

A vector divided by a number, returns a vector with each value divided by the number.

Example 1: Half health

When the *halfhealth* event is received, *health* is divided by 2.

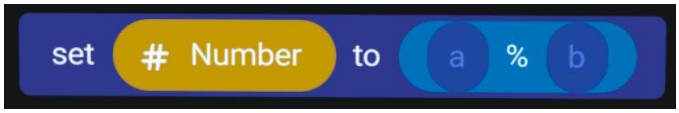
%

Operators > Basic Operations > %

Modulus

The remainder after dividing A by B.

Appearance in Composition Pane



Description

Modulus returns the remainder of dividing A by B. Modulus is often confused with percentages, but they are unrelated.

Parameters

number, number

Returns a number that is the remainder after dividing number A by number B.

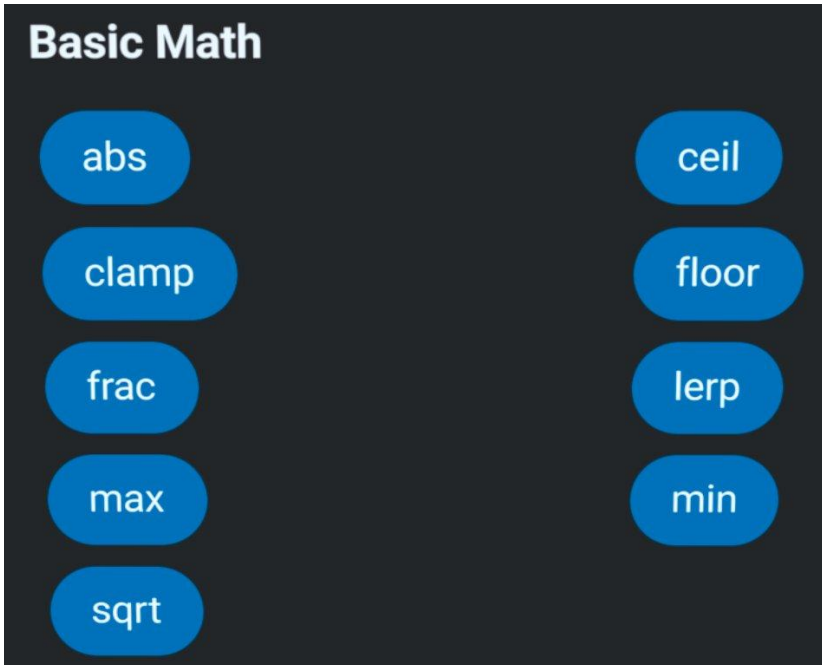
Example 1: Calculate Seconds

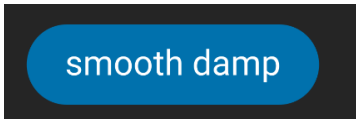
Imagine we have *curtime* which is a large number of seconds, we can then set a clocks individual *seconds* to *curtime* modulus 60. Here are a couple examples

- *Seconds* = *curtime* % 60
- 35 = 35 % 60
- 59 = 119 % 60
- 0 = 120 % 60
- 1 = 121 % 60
- 15 = 255 % 60

Basic Math

Operators > Basic Math





Codeblocks used to make calculations easier.

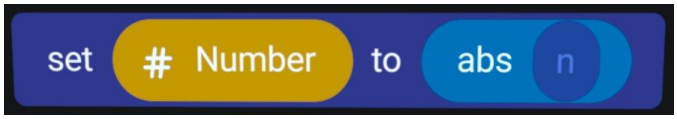
abs

Operators > Basic Math > abs

Absolute Value

Converts negative values into positive values.

Appearance in Composition Pane



Description

If a value is negative, the value returns positive. If a value is positive, nothing changes.

Parameters

Number, **vector**

Negative values are made positive.

Example 1: Calculate the difference

By using **abs** when subtracting *myheight* from *theirheight* we return the absolute value. This way, even when *myheight* is larger we still receive a positive value.

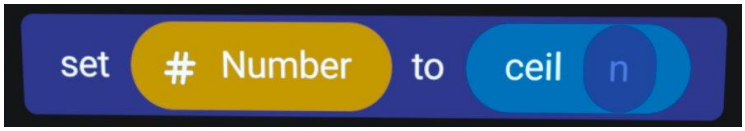
ceil

Operators > Basic Math > ceil

Ceiling

Rounds a value up to the nearest whole number.

Appearance in Composition Pane



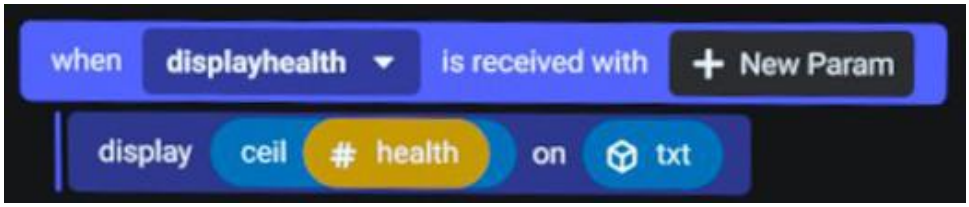
Description

Rounds the number value up to a whole number.

Parameters

number

Example 1: Whole number health



Imagine you subtracted partial health points, using ceiling makes the value look cleaner by rounding it up to a whole number. This removes any decimal points.

clamp

Operators > Basic Math > clamp

Keeps a value within a defined range.

Appearance in Composition Pane

Description

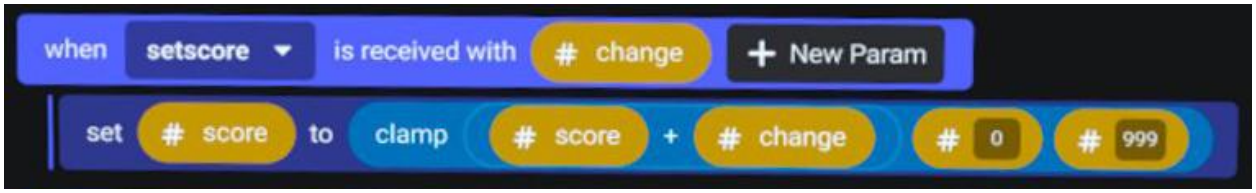
If the value is less than the minimum, clamp sets the value to the min. If the value is greater than the maximum, clamp set the value to the max.

Parameters

number, number, number
vector, vector, vector

Each value in the vector is clamped. Imagine the vectors as positions, and you can see a cuboid space that the vector value provided is clamped within.

Example 1: Keep the score between 0 and 999



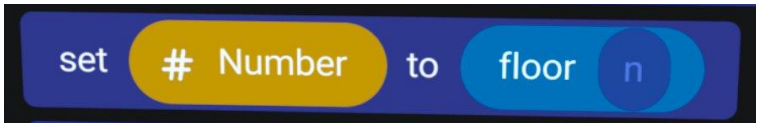
In this scenario we are increasing *score* and using **clamp** to keep (*score* + *change*) within a range of 0 and 999. Whether the change is negative or positive, the value cannot leave these bounds.

floor

Operators > Basic Math > floor

Rounds down to the nearest whole number.

Appearance in Composition Pane



Description

Rounds a number down to the nearest whole number, removing any decimal value.

Parameters

number

Example 1: Whole number division

Consider a changing variable like *distance*. If we want to give more points for being closer, we can start with a value like 10, and divide it by *distance*. Depending on *distance* this may return a number with decimals. Using **floor**, we can remove the remainder, rounding down.

frac

Operators > Basic Math > frac

Fraction

Removes the whole number.

Appearance in Composition Pane

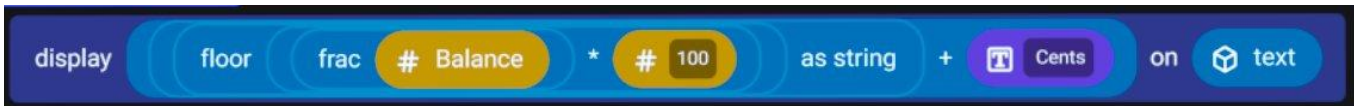
Description

Removes the whole number, and returns the decimals, as an example 12.42 becomes 0.42.

Parameters

Number

Example 1: Display cents



In this example we want to display the decimal of *Balance* as cents, using **frac** we can remove the whole number. We then multiply by 100 to move the decimal over two places, and use **floor** to remove any extraneous decimals.

lerp

Operators > Basic Math > lerp

Linear Interpolation

A position between two values.

Appearance in Composition Pane

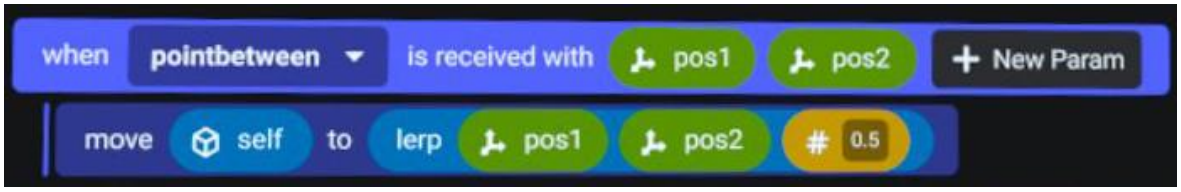
Description

Linear interpolation is useful for finding a value between two other values. From is the minimum, to is the maximum, and alpha indicates the position of the value to return. Alpha accepts numbers between 0 and 1. **Lerp** runs the calculation: min + ((max-min)*alpha).

Parameters

number, number, number
vector, vector, number
Lerps each value of the vector.
rotation, rotation, number
Lerps each value of the rotation.

Example 1: Position Between Vectors



Imagine two position vectors, if we wanted to find a point along a line between the two, **lerp** works perfectly. In this example we are finding the midpoint between *pos1* and *pos2*.

smooth damp

Operators > Basic Math > smooth damp

Smoothly change a number or vector to another number or vector over time.

Appearance in Composition Pane

Description

The MoveTo and MoveBY codeblocks allow moving an object from one place to another at a constant velocity. This code block allows you to move an object smoothly from one location to another such that it slows down naturally as it arrives. Note that it does not behave like a spring and the object will *not* pass the destination and overshoot. The object will move to the destination and slowly come to a stop.

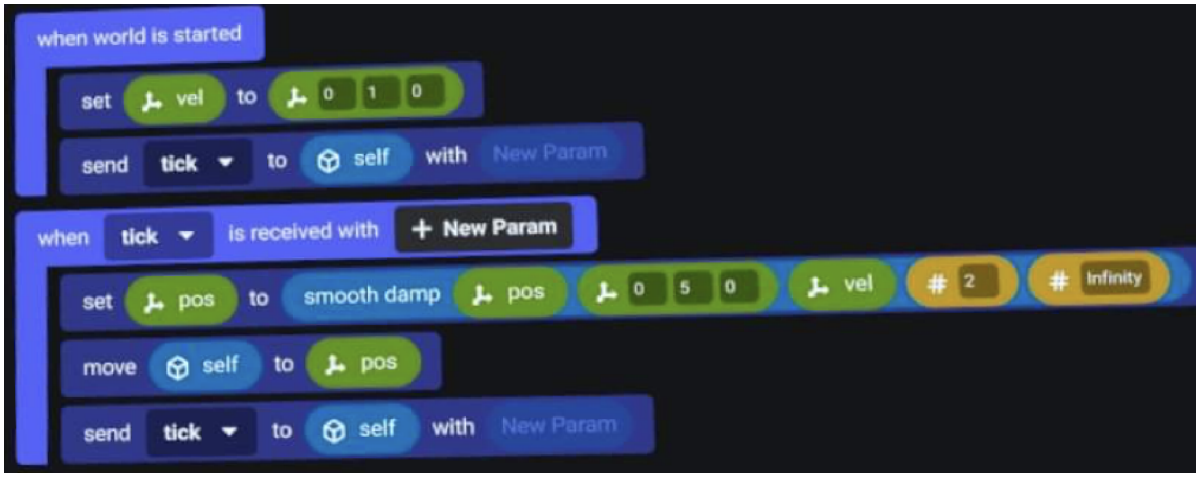
Just like the spring-push and spring-spin codeblocks, this codeblock only calculates *one frame* worth of data. This means you need to call it in a loop if you want to make an object move all the way to the destination.

Be careful: the parameter “velocity var” is *modified* each time you call this codeblock!

Parameters

from - a vector (or number) for the current location you want to move from
to - a vector (or number, though it must have the same type as “from”) representing where you want get to over time
velocity var - a script variable that contains the current velocity (a vector or number, matching the types of “from” and “to”). Note that this codeblock will *change* the value of this variable after you call the codeblock!
speed - this is the speed of the calculation (and not the object), determining how long the calculation smoothes the motion over. A higher speed value results in more damping and thus the object moves slower and takes longer to come to a stop.
max speed - a number representing the fastest speed that “velocity var” can take on. You can set this to “Infinity” if you don’t want to impose any limits.

Example: Smooth damp an object's movement



What it does: When the world starts, the object will move toward the position (0, 5, 0) with a starting velocity of (0, 1, 0) which is 1m/s up. It will be a slower movement that gradually comes to a stop.

How it works: We use the "smooth damp" code block every frame to calculate the new position of the object. Passing in "2" for the speed produces a slower, more relaxed movement.

See Also

- Motion > Physical Motion > spring spin object toward rotation
- Motion > Physical Motion > spring push object toward position

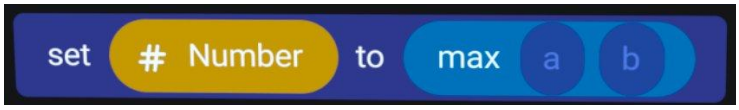
max

Operators > Basic Math > max

Maximum

Returns the greater value.

Appearance in Composition Pane



Description

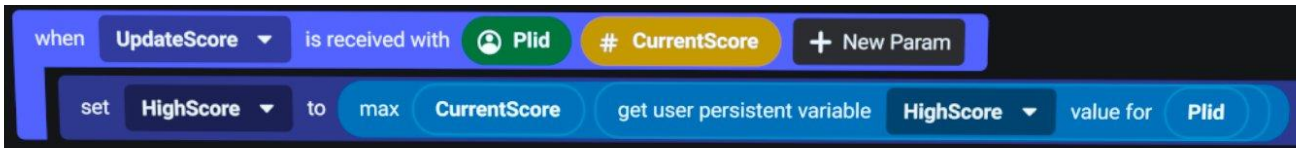
Returns the maximum value when comparing A and B.

Parameters

number, number
vector, vector

The max of vectors A and B returns (max(Ax,Bx),max(Ay,By),max(Az,Bz)).

Example 1: Update highscore



To update *HighScore*, we can use **max** to compare *CurrentScore* and *HighScore*.

See Also

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

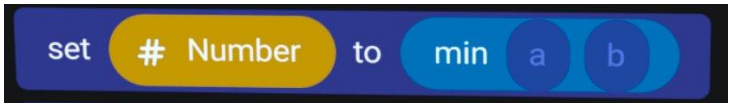
min

Operators > Basic Math > min

Minimum

Returns the lower value.

Appearance in Composition Pane



Description

Returns the minimum value when comparing A and B.

Parameters

number, number
vector, vector

The min of vectors A and B returns (min(Ax,Bx),min(Ay,By),min(Az,Bz)).

Example 1: Shortest distance

We can find the *shortestdistance* by using **min** on *distance1* and *distance2*.

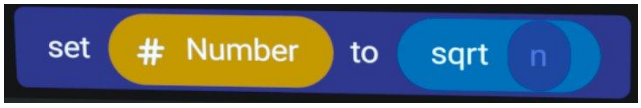
sqrt

Operators > Basic Math > sqrt

Square Root

Returns the square root of a number.

Appearance in Composition Pane



Description

Returns the square root of a number. If a negative number is given in the **sqrt** codeblock, it returns 0. The result of **sqrt** is always greater than or equal to 0.

Parameters

number

Example 1: Calculate the hypotenuse

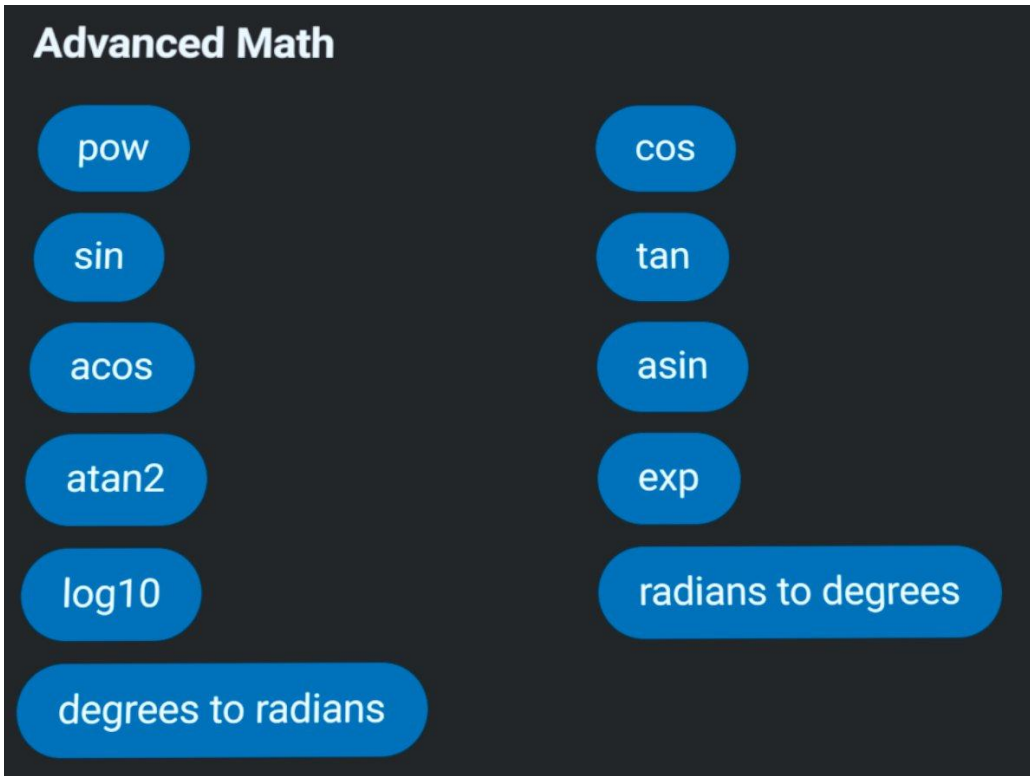
You might recall that $A^2 + B^2 = C^2$. A & B representing lengths of a 90^o triangle, and C representing the hypotenuse. Given A & B, we can calculate C by using **sqrt** of $A^2 + B^2$.

See Also

- Operators > Advanced Math > pow

Advanced Math

Operators > Advanced Math



Advanced math codeblocks used to make calculations easier.

pow

Operators > Advanced Math > pow

Power

Performs the calculation A^B.

Appearance in Composition Pane

Description

Performs the calculation A to the power of B. Here are a couple examples:

Input: A = 5 , B = 3
$5^3 = 5*5*5 = 125$

Input: A = 4 , B = -2
$4^{-2} = 1 / (4^2) = 1 / (4*4) = 1 / 16 = 0.0625$

Parameters

Number, Number

Example 1: Calculate the hypotenuse

You might recall that $A^2 + B^2 = C^2$. A & B representing lengths of a 90^o triangle, and C representing the hypotenuse. Given A & B, we can then calculate C by using **sqrt** of $A^2 + B^2$.

See Also

- Operators > Basic Math > sqrt

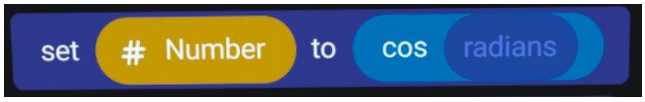
COS

Operators > Advanced Math > cos

Cosine

Performs the cosine trigonometry function.

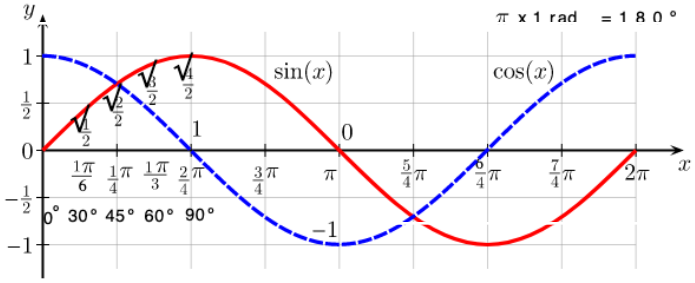
Appearance in Composition Pane



Description

Runs the cosine trigonometry function. Radians can be calculated using pi (π), where 2π = 360°.

cos(0π) = 1
cos(0.5π) = 0
cos(1π) = -1
cos(1.5π) = 0
cos(2π) = 1



Parameters

number

Example 1: Down and up motion

You may consider using the waveform created by cosine to create smooth motion down and up.

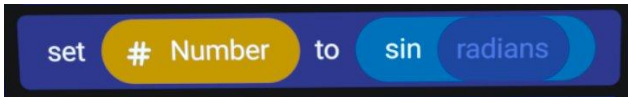
sin

Operators > Advanced Math > sin

Sine

Performs the sine trigonometry function.

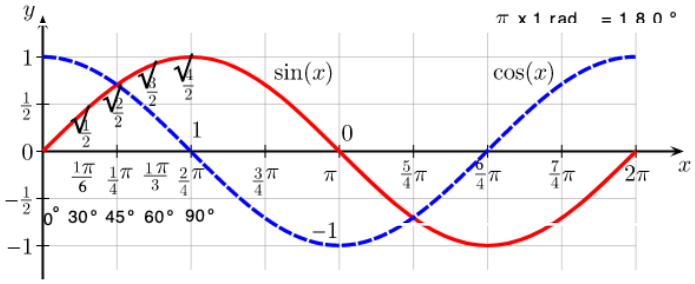
Appearance in Composition Pane



Description

Runs the sine trigonometry function. Radians can be calculated using pi (π), where 2π = 360°.

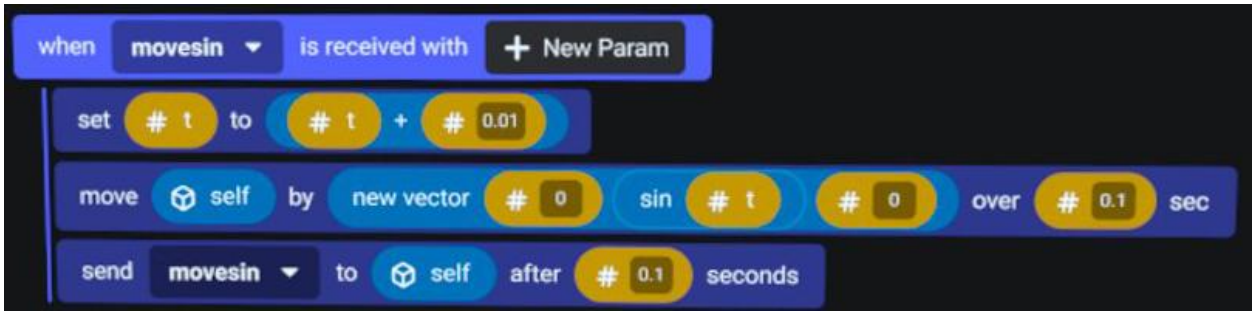
sin(0π) = 0
sin(0.5π) = 1
sin(1π) = 0
sin(1.5π) = -1
sin(2π) = 0



Parameters

number

Example 1: Up and down motion



You may consider using the waveform created by sine to create smooth motion up and down.

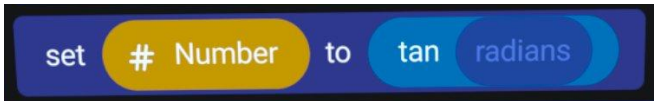
tan

Operators > Advanced Math > tan

Tangent

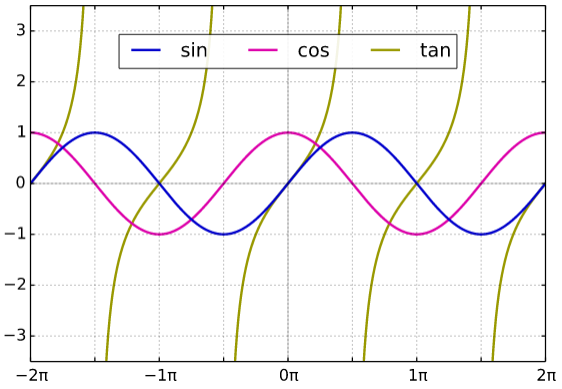
Performs the tangent trigonometry function.

Appearance in Composition Pane



Description

Runs the tangent trig function. Radians can be calculated using pi (π), where 2π = 360°.

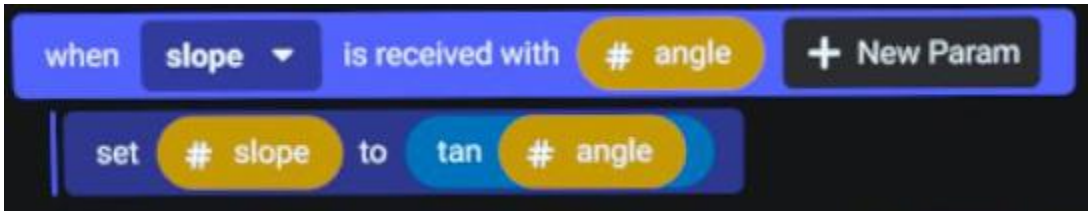


tan(x) = sin(x)/cos(x).
tan(0π) = 0
tan(0.25π) = 1
tan(0.5π) = ∞
tan(0.75π) = -1
tan(1π) = 0

Parameters

number

Example 1: Find the slope



On a right triangle, we can calculate the upward *slope* by plugging an *angle* into the **tan** function. A higher *slope* correlates to a steeper incline.

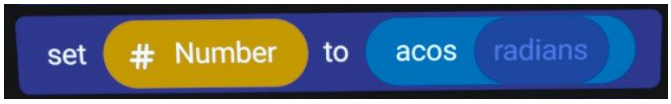
acos

Operators > Advanced Math > acos

Arccosine

Performs the arccosine trigonometry function, which is the inverse of cosine.

Appearance in Composition Pane



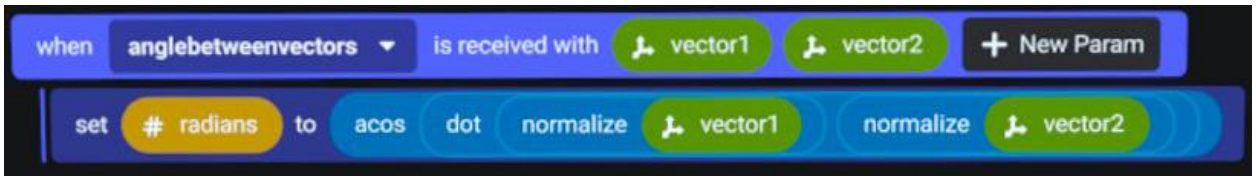
Description

Arccosine is the inverse of cosine. Using the ratio of the length of the adjacent over the length of the hypotenuse of a right triangle, the arccosine is the angle in radians. Numbers less than -1 or greater than 1 are not accepted.

Parameters

number

Example 1: Find the angle in radians between two vectors



When the *anglebetweenvectors* event is received with *vector1* and *vector2* as parameters, it sets *radians* to the arccosine of the **dot** product of the two normalized vectors. Because the dot product of two normalized vectors is the cosine of the angle between the vectors, taking the arccosine of the result converts it into the angle in radians.

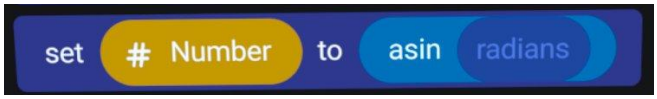
asin

Operators > Advanced Math > asin

Arcsine

Performs the arcsine trigonometry function, which is the inverse of sine.

Appearance in Composition Pane



Description

Arcsine is the inverse of sine. If the ratio of the length of the opposite from an angle over the length of the hypotenuse of a right triangle is given, the arcsine is the angle in radians. Numbers less than -1 or greater than 1 are not accepted.

Parameters

number

Example 1: Find the angle in radians between two vectors

When the *anglefromplane* event is received with *vector1* and *vector2* as parameters, it sets *radians* to the arcsine of the y of the **cross** product of the two vectors normalized. The y of the cross product of two normalized vectors is the sine of the angle between vector2 and the plane containing vector1 and the y-axis. Therefore, taking the arcsine of the result converts it into an angle in radians.

atan2

Operators > Advanced Math > atan2

Arctangent2

Performs the arctangent trigonometry function.

Appearance in Composition Pane

Description

Arctangent2 takes an x and y coordinate, and returns an angle in radians of a line drawn from the origin point to that position.

Parameters

number

Example 1: Find the angle of a slope

When the *anglefromslope* event is received with *rise* and *run* as a parameter, it calculates the angle of the slope in radians based on the rise and the run. The *rise* represents the change in y while the *run* represents the change in x across a slope.

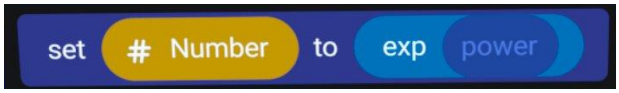
exp

Operators > Advanced Math > exp

Exponent

The constant E, Euler's Number, to a power.

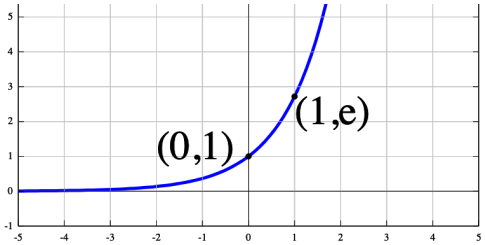
Appearance in Composition Pane



Description

The constant E, Euler's Number, to a power.
E is roughly equal to 2.71828.

To the right is a graph of E^x



Parameters

number

Example 1: Compound interest

One common use of E, is calculating compound interest. If a bank account paid 100% annual percentage yield, the account balance would be E^t times the original balance. Where *t* is years.

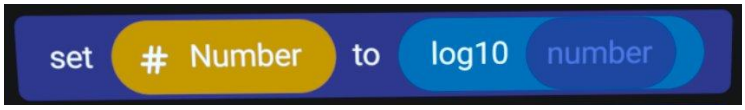
log10

Operators > Advanced Math > log10

Log base 10

The common logarithm or decadic logarithm function.

Appearance in Composition Pane



Description

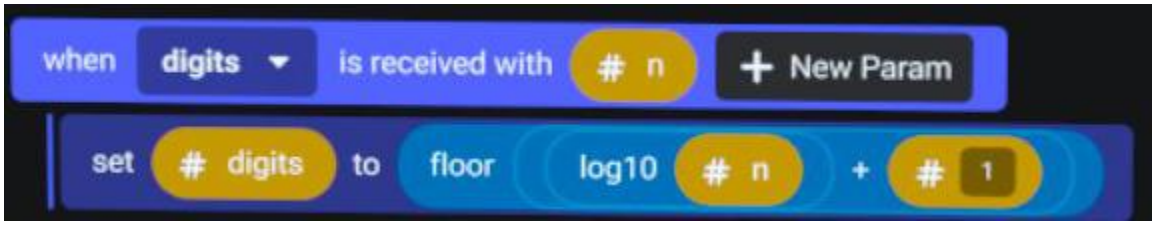
Log base 10 allows you to calculate what exponent is required to create that number. Consider 10^3 or 10*10*10, which equals 1,000. Using $\log_{10}(1000)$, we return the value 3. Therefore, log10 tells you how many times 10 is multiplied by itself to equal that number. A few examples:

$\log_{10}(10) = 1$
$\log_{10}(100) = 2$
$\log_{10}(1000) = 3$

Parameters

number

Example 1: Length of a whole number



Consider the value *n*, we can calculate the length, *digits*, by performing **log10**(*n*) plus 1, and use floor to remove any decimals. As an example, plugging 275 into n, returns 3.

digits = floor($\log_{10}(275)$ + 1)
3 = floor(2.44 + 1)

radians to degrees

Operators > Advanced Math > radians to degrees

Converts radians to degrees.

Appearance in Composition Panel

Description

Used to convert radians to degrees by multiplying by 57.29578.

Pi => 180°

Parameters

number

Example 1: Convert radians to degrees

Radians to degrees is useful anytime you need to change units.

degrees to radians

Operators > Advanced Math > degrees to radians

Converts degrees to radians.

Appearance in Composition Pane

Description

Used to convert degrees to radians by multiplying by 0.01745.

$180^{o} \approx > 3.14$

Parameters

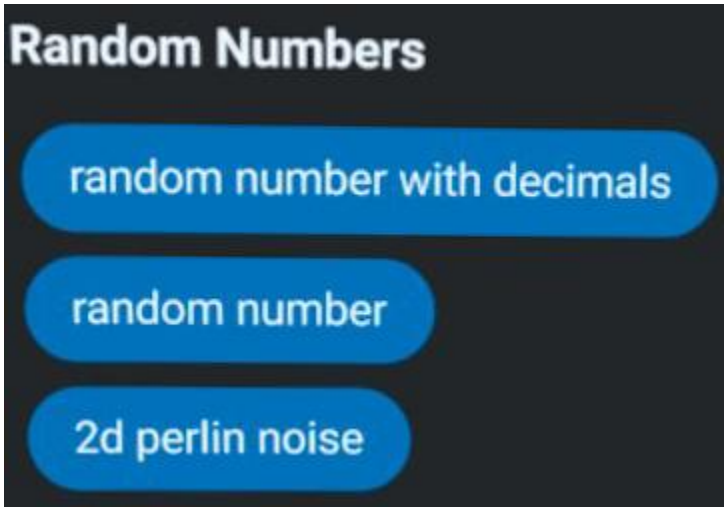
number

Example 1: Convert degrees to radians

Degrees to radians is useful anytime you need to change units.

Random Numbers

Operators > Random Numbers



Codeblocks that return random numbers.

random number with decimals

Operators > Random Numbers > random number with decimals

Returns a random number between the min and max with decimals.

Appearance in Composition Pane

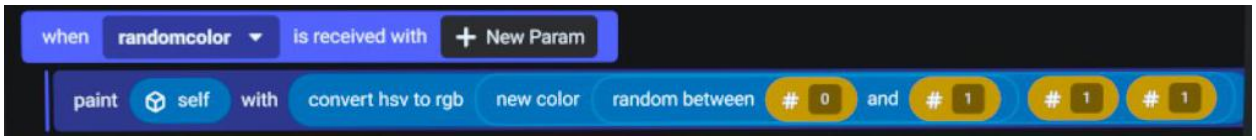
Description

Returns a random number between the min and max with decimals.

Parameters

Number, number

Example 1: Random color



When *randomcolor* is received, *self* changes to a new color created by selecting a hue randomly from 0 to 1. Saturation and value are set to 1, but could also be generated randomly.

random number

Operators > Random Numbers > random number

Random whole number between the min and max.

Appearance in Composition Pane

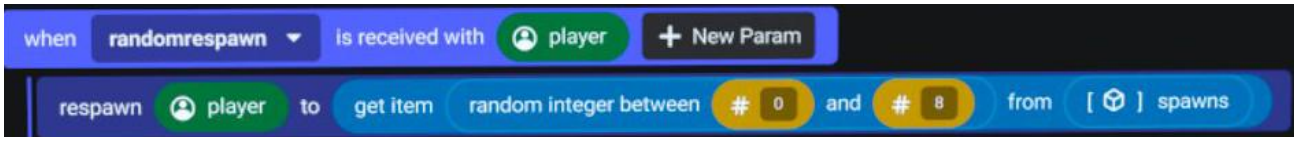
Description

Generates a random whole number between the min and max. Please note that the possible numbers include the min, but do not include the max.

Parameters

Number, number

Example 1: Spawn at a random respawn gizmo



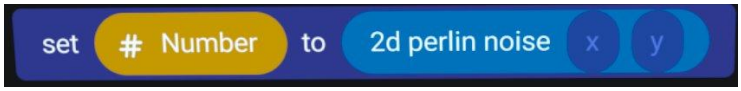
When *randomrespawn* is received with a *player* as a parameter, the *player* is respawned to a random spawn gizmo from a list of objects.

2d perlin noise

Operators > Random Numbers > 2d perlin noise

Random number generated from an x and y value.

Appearance in Composition Pane



Description

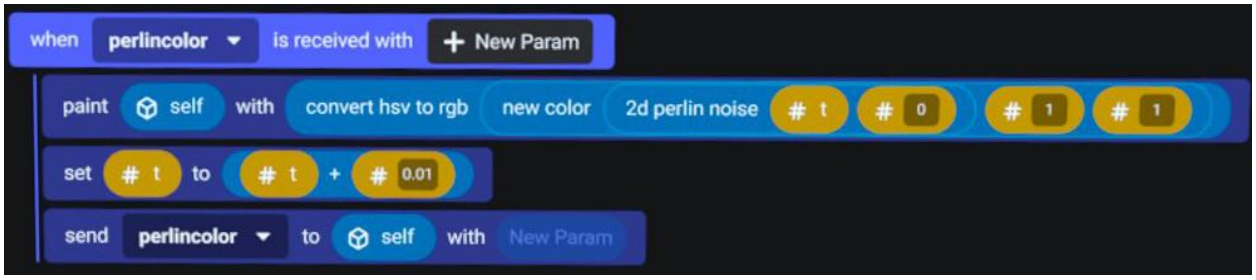
Perlin noise is often described as clouds, returning values between 0 and 1. While the numbers are relatively random, there are some patterns to look out for. First you will note that the results of two numbers will always be the same. Nearby x,y pairs are similar and change slowly.

To generate numbers between a minimum and maximum, multiply the perlin noise by the difference between the minimum and the maximum plus the minimum. As an example, for values between 6 and 10, use (2D perlin noise (X,Y) * 4) + 6.

Parameters

Number, number

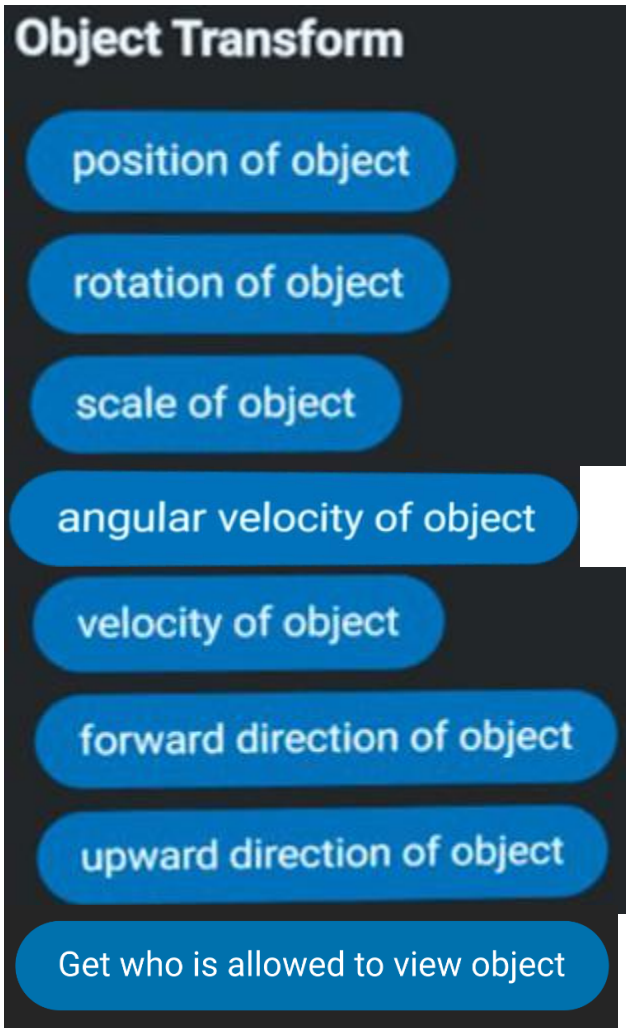
Example 1: Gradually change a color



An objects color is slowly adjusted using a hue value from 2D perlin noise with t and 0 as inputs. The event is looped, so that *t* increases by 0.01. Because *t* is a parameter in the **2d perlin noise** codeblock, there is slight color change with every iteration.

Object Transform

Operators > Object Transform



Codeblocks that return data about an object's orientation in space.

position of object

Operators > Object Transform > position of object

Position vector of an object.

Appearance in Composition Pane

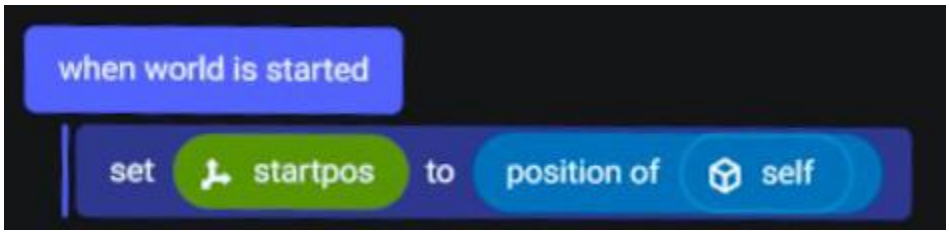
Description

Returns the position vector of an object, the x, y, and z coordinates. This is the center of the object, the same coordinates visible in the attributes tab of the object's property panel.

Parameters

object

Example 1: Save the start position of an object



When world is started, we save the position of self as the vector *startpos*. This allows another event to move the object back to its *startpos*.

rotation of object

Operators > Object Transform > rotation of object

Rotation of an object.

Appearance in Composition Pane

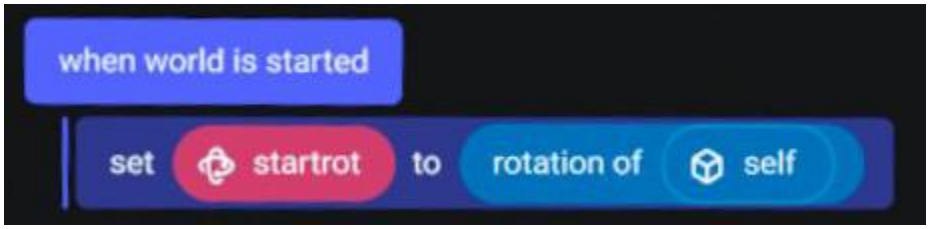
Description

Gives you the pitch, yaw, and roll in degrees of an object.

Parameters

object

Example 1: Save the start rotation of an object



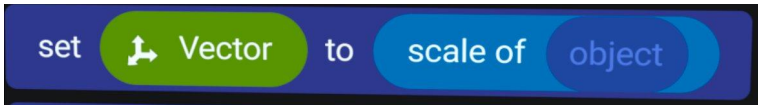
When world is started, we save the **rotation of** *self* as the rotation *startrot*. This allows another event to rotate the object to *startrot*.

scale of object

Operators > Object Transform > scale of object

Scale of an object.

Appearance in Composition Pane



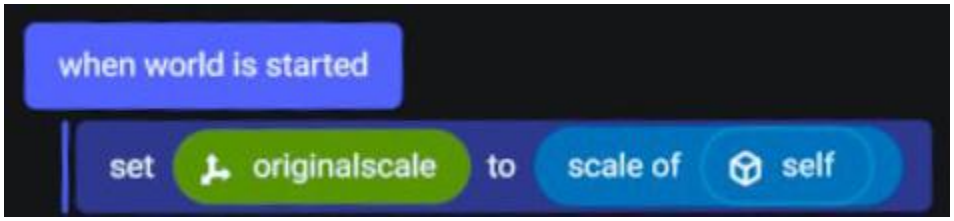
Description

Gives you the width, length, and height of an object. If the object is grouped, the scale is based on its current size relative to its size when it was grouped. If it is a single object, the scale is based on its size in meters.

Parameters

object

Example 1: Save the original scale of an object



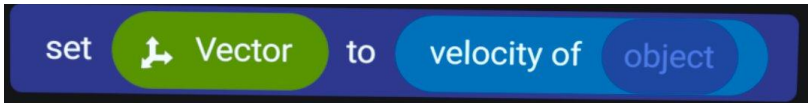
When world is started, it saves the **scale of** *self* as the vector variable *originalscale*. This allows another event to scale the object to the *originalscale*.

velocity of object

Operators > Object Transform > velocity of object

Velocity of an object.

Appearance in Composition Pane



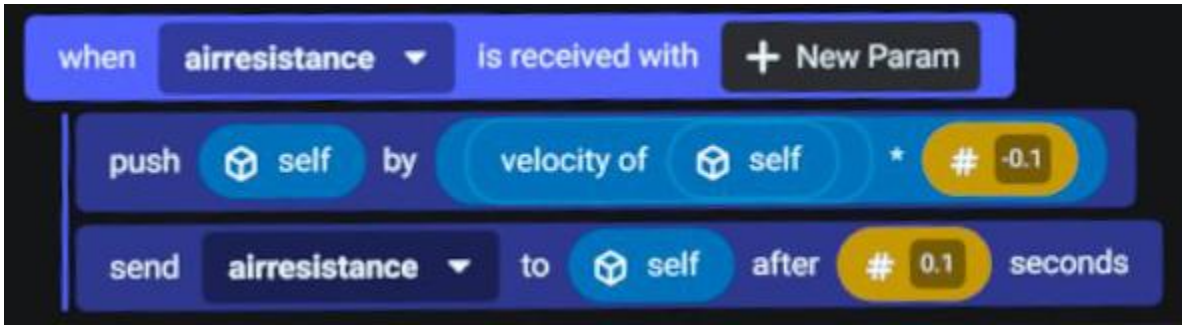
Description

Returns a velocity vector, in meters per second, using the x, y, and z direction.

Parameters

object

Example 1: Air resistance



When the *airresistance* event is received, the object is pushed with negative one tenth the **velocity of** *self*. This provides resistance to its current motion and decelerates it to a stop.

angular velocity of object

Operators > Object Transform > angular velocity of object

Returns how fast an object is spinning.

Appearance in Composition Pane

Description

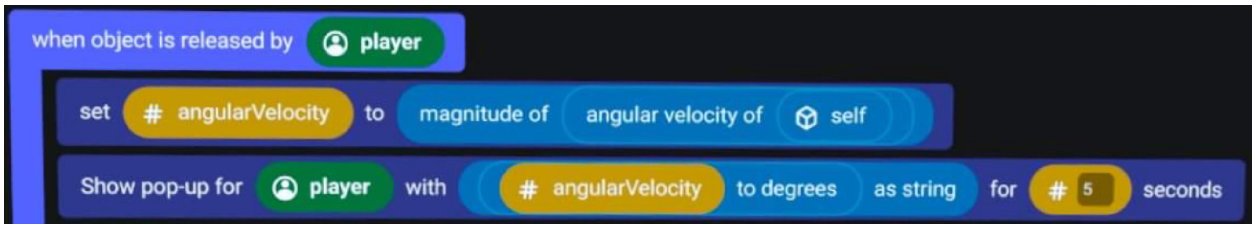
This codeblock returns a vector representing how fast an object is spinning. The direction of the vector is the axis of rotation and the length of the vector is the speed in radians/sec. If the vector is pointing right at you then it would spin clockwise around the vector. As an example: If an object were spinning around the y-axis, at 2.5 radians/sec such that it is going clockwise when viewed from above, then its angular velocity would be given by the vector (0, 2.5, 0).

Parameters

object - the object you want to get the angular velocity of

Example: Spin an object and see how fast it is

What it does: Grab an object, flick your wrist, and let go; the object flies away spinning. You then get a message that appears to tell you how fast the object is spinning (in degrees per second).



How it works: When the object is released we get the angular velocity vector, We take its magnitude to get the speed and then convert to degrees (since radians are less familiar to most people). Then we use the popup codeblock to show the person how fast they spun it.

forward direction of object

Operators > Object Transform > forward direction of object

The direction of the positive z axis of an object.

Appearance in Composition Pane

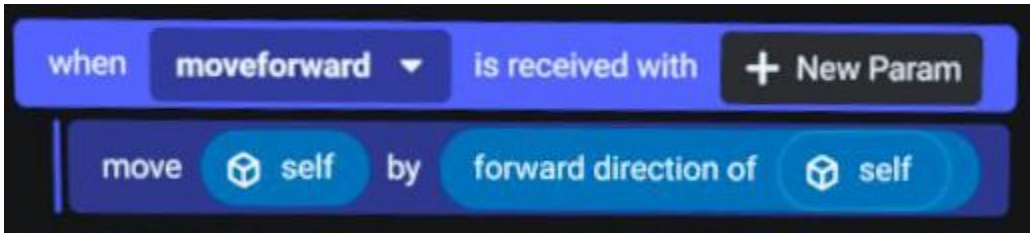
Description

Gives you the objects' direction vector along the positive z-axis with a length of one meter.

Parameters

object

Example 1: Air resistance



When the *moveforward* event is received, the object moves one meter in its forward direction.

upward direction of object

Operators > Object Transform > upward direction of object

The direction of the positive y-axis of an object.

Appearance in Composition Pane

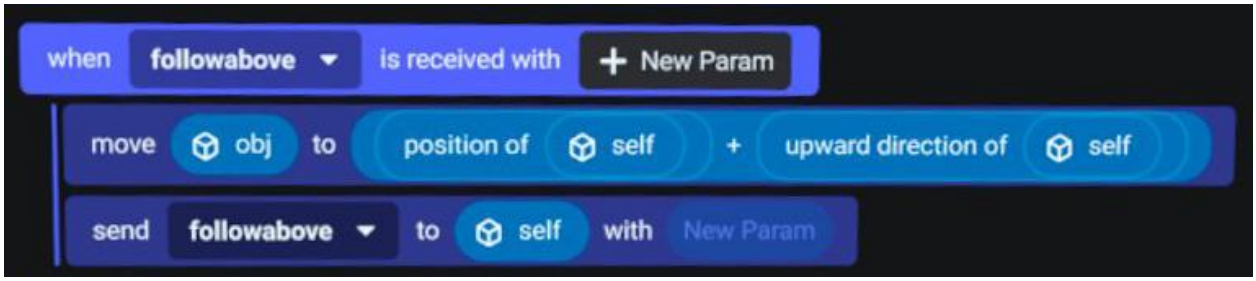
Description

Gives you the object's direction vector of the positive y-axis with a length of one meter.

Parameters

object

Example 1: Follow above an object



When the *followabove* event is received, **move** *obj* to the position one meter above *self*. The event is in a loop and the object continues to follow the position above *self*.

get who is allowed to view object

Operators > Object Transform > get who is allowed to view object

Returns a boolean of true or false depending on if the player has permission to view the object.

Appearance in Composition Pane

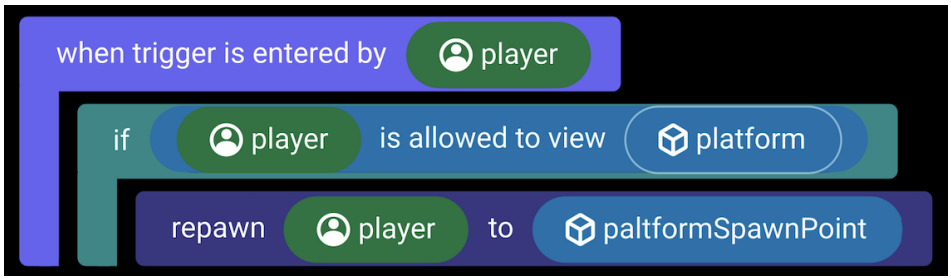
Description

The collection of “allowed to view” codeblocks enable control over which players have permission to see an object. This codeblock returns if a player currently has permission to view the object. This does not mean they can see it at the exact moment though, since it may have had its visibility set to false.

Parameters

player - the player that you want to check for.
object - the object you want to check the visibility status of.

Example: A platform that spawns you to a new place - if you can see it



What it does: There is a platform that you can only see after you solve a puzzle. If you stand on the platform it spawns you into a secret room. But if a person tries to follow you and also stand on the platform, then nothing will happen if they can’t also see the platform!

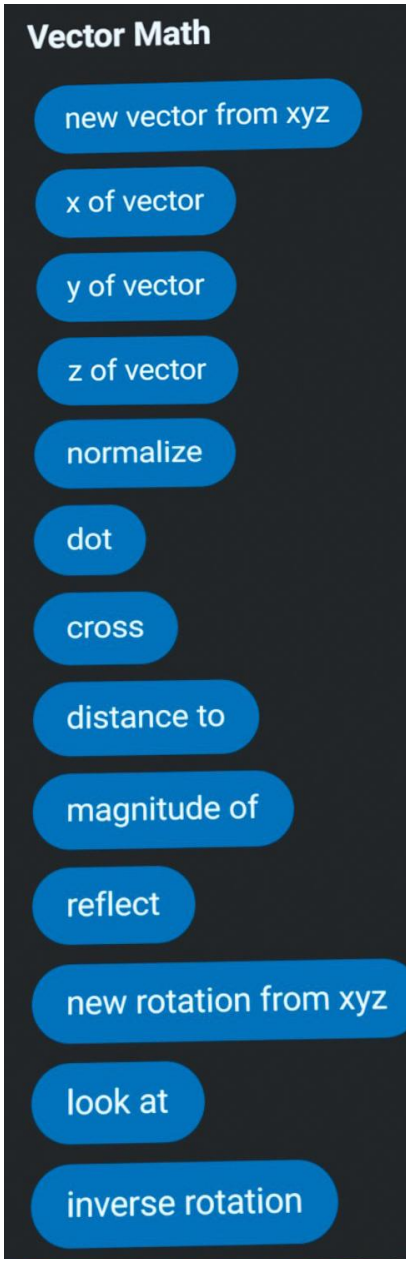
How it works: There is a trigger around the platform. When a player enters the trigger, we check to see if they can see the platform. If they can, we spawn them to a spawn point.

See Also

- **Actions > Object > set who is allowed to view object**
- **Actions > Object > reset who is allowed to view object**

Vector Math

Operators > Vector Math



Codeblocks used in vector & rotation math calculations.

new vector from xyz

Operators > Vector Math > new vector from xyz

New vector based on number inputs.

Appearance in Composition Pane

Description

Creates a new vector from three numbers x, y, and z.

Parameters

number, number, number

Example 1: Move to 0,0,0

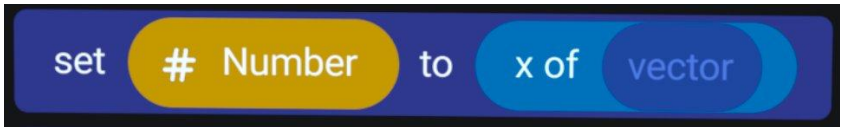
When object is released by *player*, it moves to the origin of the world at (0,0,0).

x of vector

Operators > Vector Math > x of vector

The x value of a vector.

Appearance in Composition Pane



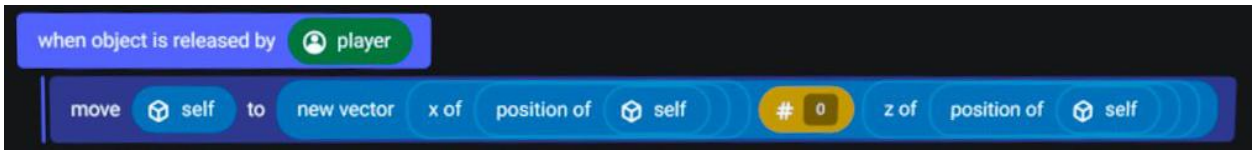
Description

Returns the x value of a vector.

Parameters

vector

Example 1: Move Along Y



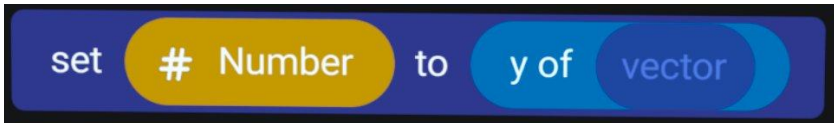
When the object is released, it moves along the world's Y axis to 0. This uses X and Z of position of *self.* And a number input for Y.

y of vector

Operators > Vector Math > y of vector

The y value of a vector.

Appearance in Composition Pane



Description

Returns the y value of a vector.

Parameters

vector

Example 1: Is the object above a player

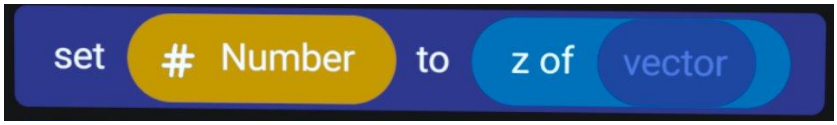
When the *abovebelow* event is received with *player* as a parameter, it sets the boolean above to true If the y of the object's position is above the y of the player's position.

z of vector

Operators > Vector Math > z of vector

The z value of a vector.

Appearance in Composition Pane



Description

Returns the z value of a vector.

Parameters

vector

Example 1: Rotate to the nearest horizontal direction

When the object is released, remove the y-value from the current forward direction so that it is parallel to the grid. This can also be done by multiplying (forward direction of self) * vect(1,0,1).

normalize

Operators > Vector Math > normalize

Normalizes a vector's length to 1 meter.

Appearance in Composition Pane

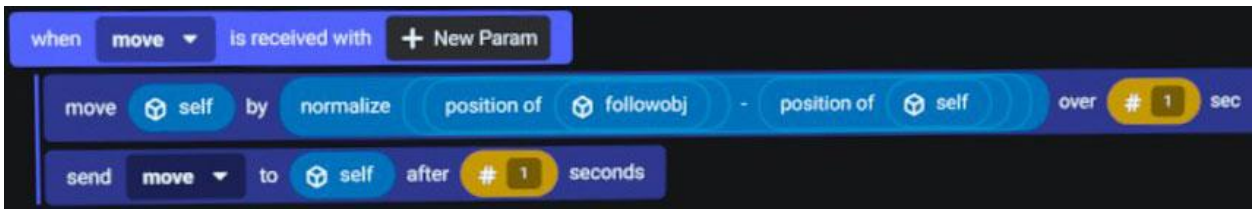
Description

A normalized vector is calculated by dividing a vector by its length. Which returns a vector with a length of 1 in the same direction. **Normalize** does this calculation for you.

Parameters

vector

Example 1: Move towards another object with a constant speed.



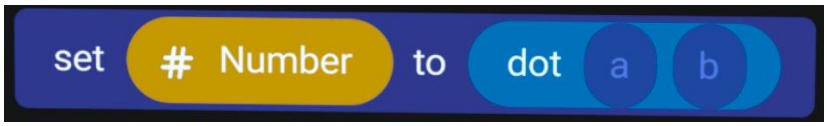
When the *move* event is received, move *self* in the direction of the object it's following. The direction vector from the object to the *followobj* can be calculated by subtracting the object's position from the *followobj* position. This direction vector will have the same length as the distance from the object to the *followobject*. If the vector wasn't normalized, the object would move the full distance from the object to the *followobj* in one second. Normalizing the vector makes the object move one meter closer to the *followobj* every second.

dot

Operators > Vector Math > dot

Returns the dot product of two vectors.

Appearance in Composition Pane



Description

Returns a number which is the dot product of two vectors, calculated using the cosine of the angle between the two vectors times the length of A, times the length of B.

Cos(angle)*magnitude(A)*magnitude(B)

If both vectors are normalized, meaning their length is 1, then the dot product of the two vectors will be between -1 and 1. Two normal vectors of the same direction have a dot product of 1. If they are perpendicular to each other, their dot product is 0. And in the opposite direction of each other, their dot product is -1.

To find the angle between two vectors, take the arccosine of the dot product of two normal vectors. You can then convert the result from radians to degrees.

Parameters

vector, vector

Example 1: Move a tank with a joystick

When the *movetank* event is received, the *tank* will move in its forward direction, and its speed will be based on the dot product of the upward direction of the joystick and the forward direction of the tank. As the player points the joystick towards the forwards direction of the tank, the dot product increases to 1. If the joystick points straight up, the dot product will be 0 because it's perpendicular to the forwards direction of the tank. As the player points the joystick towards the back end of the tank, the dot product decreases to -1 and the tank moves backwards. The speed of the tank depends on the dot product because the forward direction is multiplied by the dot product. This changes the length of the forward direction that it is moving.

cross

Operators > Vector Math > cross

Cross product of two vectors.

Appearance in Composition Pane

Description

The cross product of two vectors is a vector that is perpendicular to both A and B. For any two vectors, there are always two vectors that are perpendicular to both. One to the right, and one to the left. If you were to point the top of your head in the direction of vector A and look in the direction of vector B, the cross product would be the perpendicular vector to the right of you. Therefore, to find the vector pointing to the right of an object, take the cross product of the upwards direction and the forwards direction of the object.

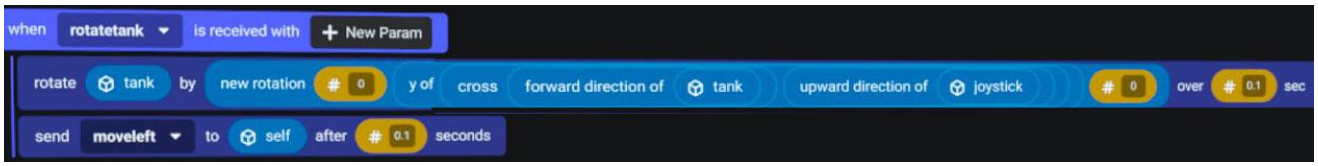
Switching the order of A and B in a cross product returns the opposite direction.

Parameters

vector, vector

Returns the cross product of two vectors as a vector.

Example 1: Rotate a tank with a joystick



When the *rotatetank* event is received, rotate the y-axis of *tank* based on the y of the cross product of the forward direction of the tank and upwards direction of the joystick. If the upwards direction of the joystick is to the right of the plane that makes up the forwards direction of the tank and the y-axis of the world, then the tank will rotate clockwise. If the joystick points to the left, the tank will rotate counterclockwise.

distance to

Operators > Vector Math > distance to

Distance between two positions.

Appearance in Composition Pane

Description

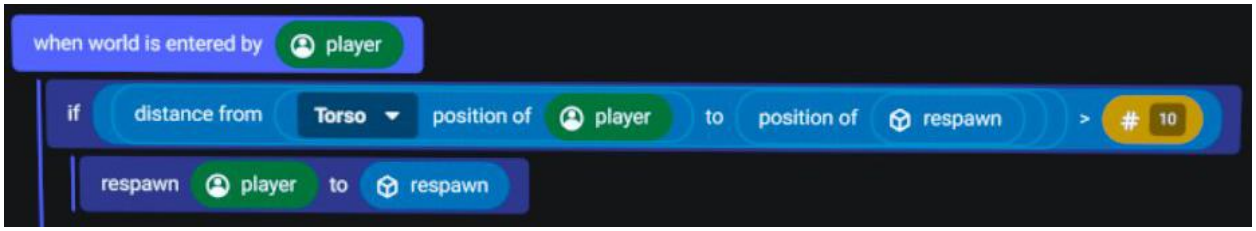
Distance to calculates the distance between two vector positions. Distance can be calculated by sqrt((Ax-Bx)^2+(Ay-By)^2+(Az-Bz)^2). Or using magnitude of (VectorA - VectorB). This codeblock does this calculation for you.

Parameters

vector, vector

Returns a number that is the distance between two vectors in meters.

Example 1: Respawn players that are more than 10 meters from the start spawn



When world is entered by player runs after the world is reset, we can then use distance to, to check if the distance from the position of the *player* to the **position of** the *respawn* gizmo. If the distance is greater than 10 meters, we can respawn the player to the respawn gizmo. This prevents players from respawning if they are already near the respawn location.

magnitude of

Operators > Vector Math > magnitude of

The magnitude of a vector.

Appearance in Composition Pane

Description

The magnitude of a vector is its length. It can be calculated by sqrt(x^2+y^2+z^2). This codeblock does the calculation for you. Magnitude is quite useful for direction vectors. The magnitude of a position vector is the distance from the origin to that position.

Parameters

vector

Returns a number that is the length of the vector in meters.

Example 1: Convert Velocity To Speed

When the *getspeed* event is received, *speed* is set to the **magnitude of** the **velocity of** *self*. The velocity codeblock returns a vector of the speed in each direction.

reflect

Operators > Vector Math > reflect

Reflect a vector across a normal

Appearance in Composition Pane

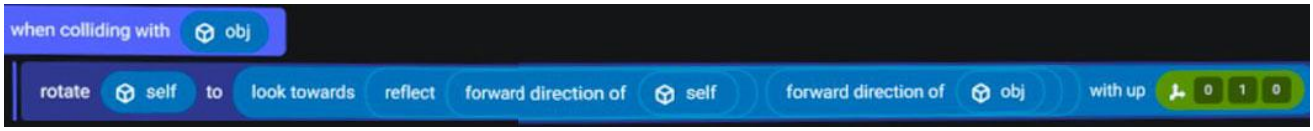
Description

Returns a vector that is the reflection of vector across a normal. You can think of this as similar to a beam of light traveling and being reflected off a mirror. The first vector is the initial direction of the light, and the normal vector is the direction that the mirror is facing.

Parameters

vector, vector

Example 1: Rotate away from wall



When an NPC collides with a wall object, it reflects its forward direction off the wall and rotates the NPC to the new forward direction. Please note that the z direction of the wall must be the facing direction of the wall for this to work.

new rotation from xyz

Operators > Vector Math > new rotation from xyz

Create a new rotation from a pitch, yaw, and roll.

Appearance in Composition Pane

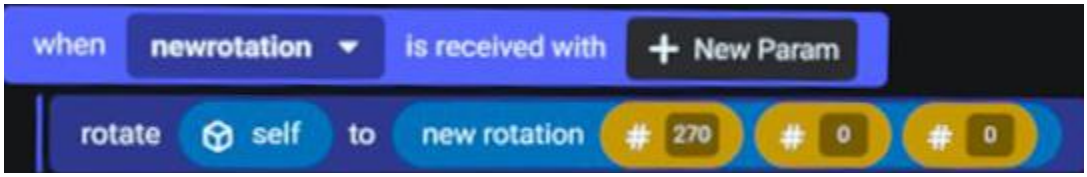
Description

Creates a new rotation value from three numbers, representing pitch, yaw, and roll in degrees.

Parameters

number, number, number

Example 1: Rotate the forward direction upwards



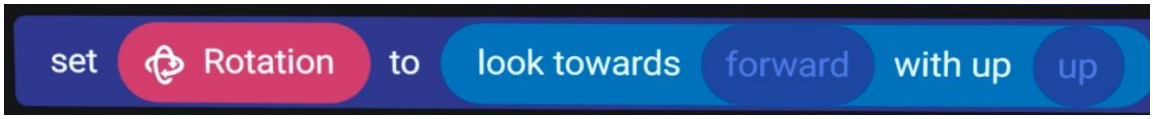
When the *newrotation* event is received, *self* rotates to the rotation (270,0,0). This will point the forward direction upwards.

look at

Operators > Vector Math > look at

Convert a forward direction vector and an upward direction vector into a rotation.

Appearance in Composition Pane



Description

The **look at** codeblock uses the forward direction and upward direction vectors to create a new rotation. The new rotation is the rotation that points the positive z-axis towards the forward direction and the positive y-axis as close to the upward direction as possible. Looking in a direction is not the same as looking at a position. See the example below.

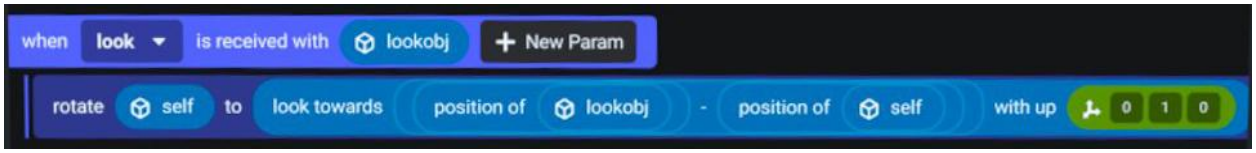
Please note, you want to avoid having the forward direction and upward direction parallel.

Parameters

vector, vector

Returns a rotation from forward and upward direction vectors.

Example 1: Look Towards an object



When the *look* event is received with *lookobj* as a parameter, the object rotates to the direction that points itself towards *lookobj*. The direction from the object to *lookobj* can be found by subtracting the object's position from the position of *lookobj*. The object's forward direction rotates towards this direction, and the upwards direction rotates as close to (0,1,0) as possible.

inverse rotation

Operators > Vector Math > Inverse Rotation

Flip a rotations pitch, yaw, and roll to the inverse.

Appearance in Composition Pane

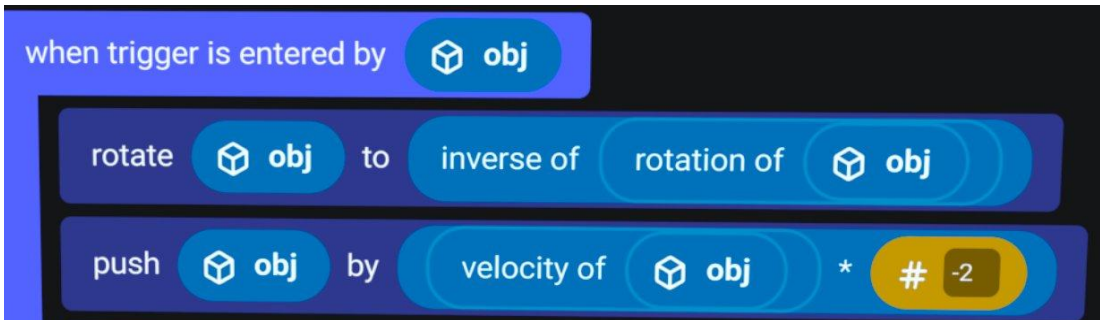
Description

Creates a new rotation that is opposite the input rotation. For instance: (0,90,0) => (0,270,0).

Parameters

rotation

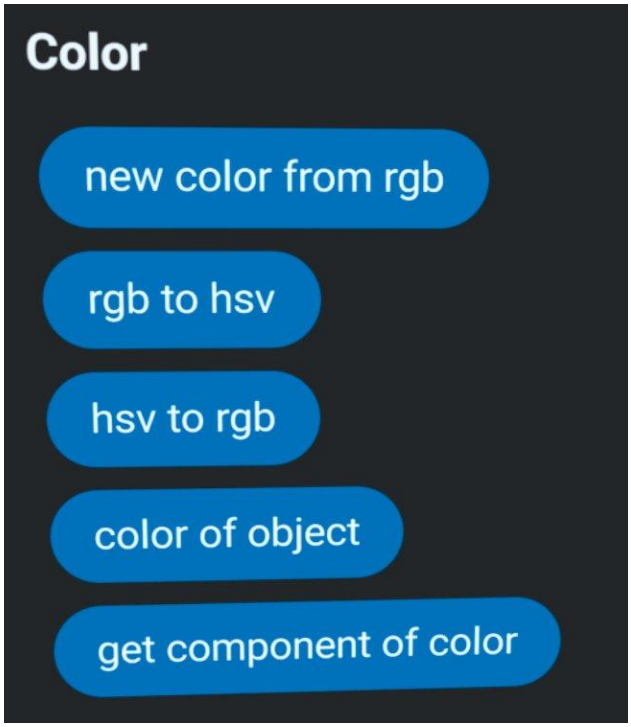
Example 1: Reverse Direction (Bouncy Net)



When the net trigger is entered by an object tagged ball, we rotate the *obj* to the inverse rotation of its current rotation. We also push the object by its current velocity multiplied by -2. This is twice the force of its current direction, which offsets its current velocity, and sends it flying in the inverse direction at the same speed at which it entered the net.

Color

Operators > Color



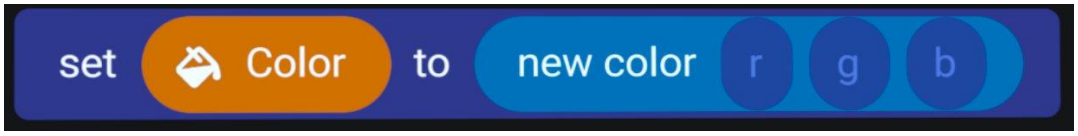
Codeblocks that get and interact with colors.

new color from rgb

Operators > Color > new color from rgb

Create a new color value.

Appearance in Composition Pane



Description

The number inputs represent how much red, green, and blue is in the new color. The numbers are from 0 to 1, with (0,0,0) being black, and (1,1,1) being white. Different combinations of red, green, and blue can create millions of colors. When placed in the HSV to RGB codeblock, the values will still be between 0 and 1, and then represent Hue, Saturation, and Value.

Parameters

number, number, number

Example 1: Darken an object

Each time the *darken* event is received, the number variables *red*, *green*, and *blue* decrease by 0.01. Then, the object is painted with the new color from the 3 variables. Because the event is in a loop, the object will gradually darken over time.

rgb to hsv

Operators > Color > rgb to hsv

Converts a color value from red, green, and blue to hue, saturation, and value.

Appearance in Composition Pane

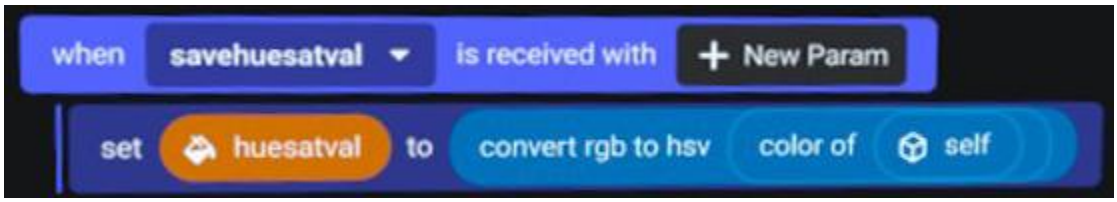
Description

Converts the red, green, and blue values into hue, saturation, and value. Hue is where in the color spectrum, ranging from red to red. Saturation is how vivid the color is. Value, is how bright the color is. Different combinations can create millions of colors.

Parameters

color

Example 1: Set a color variable to the HSV of an object



When the *savehuesatval* event is received, the hsv of the object's color is saved as a variable.

hsv to rgb

Operators > Color > hsv to rgb

Converts a color value from hue, saturation, value to red, green, and blue.

Appearance in Composition Pane

Description

Converts hue, saturation, and value to red, green, and blue. Hue is where in the color spectrum, ranging from red to red. Saturation is how vivid the color is. Value, is how bright the color is. Different combinations can create millions of colors.

Parameters

color

Example 1: Increase the Saturation of an object

When the *increasesat* event is received, it will increase *saturation* by 0.05. We can then **paint** *self* by converting the hsv values to rgb.

color of object

Operators > Color > color of object

The color of an object in red, green, and blue values.

Appearance in Composition Pane

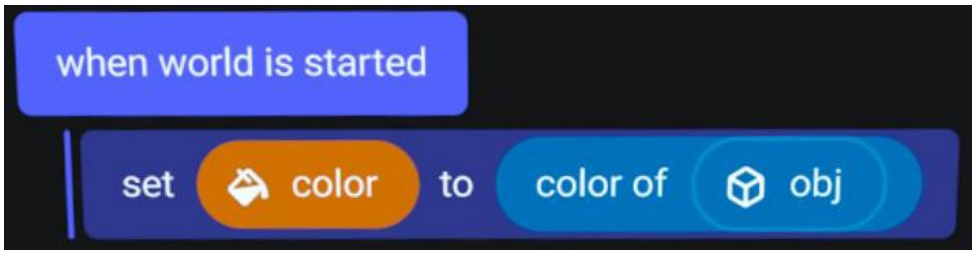
Description

Gives you the red, green, and blue color value of an object. These values are represented as numbers from 0 to 1, with 0 being the darkest and 1 being the lightest. The **color of object** codeblock does not work on grouped objects.

Parameters

object

Example 1: Save a color value as a variable



When world is started, we **set** *color* to the **color of** *obj.*

get component of color

Operators > Color > Get Component of Color

Returns a number, representing either the red, green or blue of an object's color.

Appearance in Composition Pane

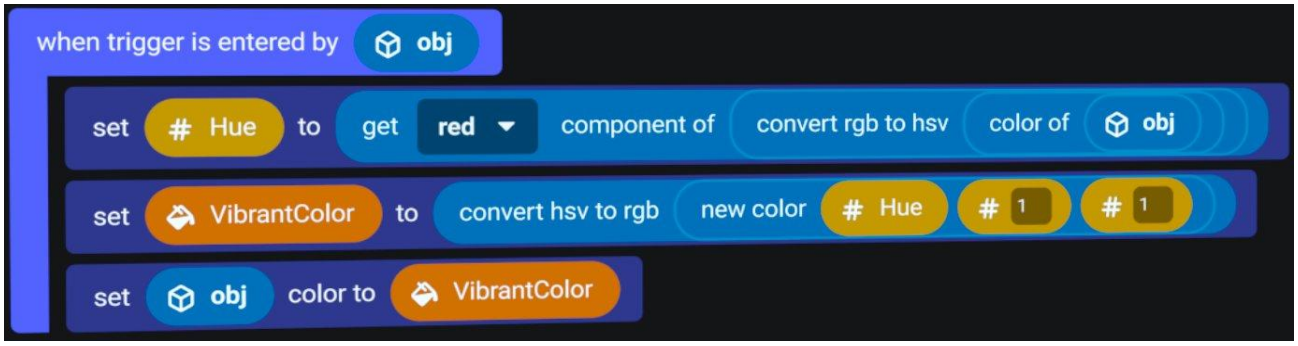
Description

Returns a number between 0 and 1, representing either the red, green or blue of an object's color. When paired with *Convert RGB to HSV*, you can retrieve the Hue, Saturation, or Value.

Parameters

component, object

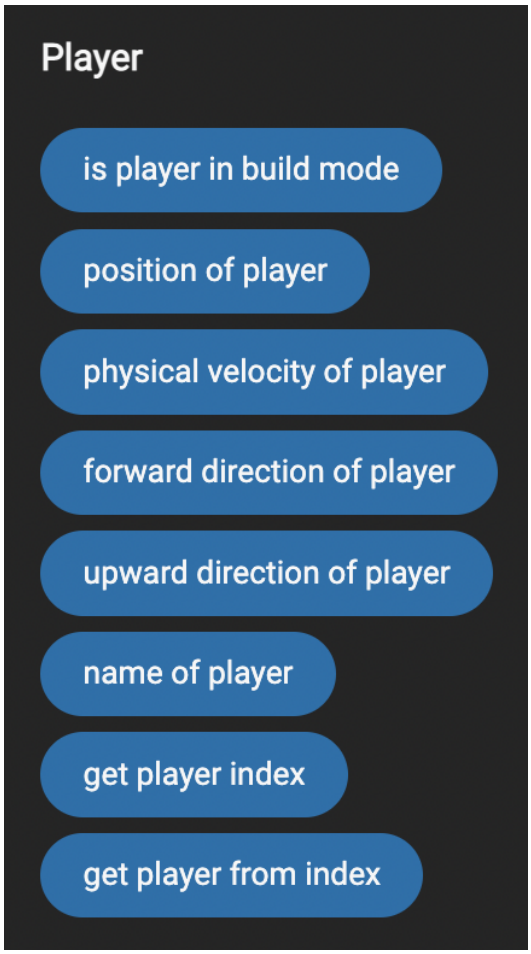
Example 1: Save a color value as a variable



When the trigger is entered by an object, we save it's *Hue* as a number variable. Then use that to create a more vibrant color by increasing its Saturation and Value to 100%. Please note that Horizon default colors are in RGB (Red, Green, Blue), so we must convert back and forth between HSV (Hue, Saturation, Value).

Player

Operators > Player



Codeblocks that get values of a player.

is player in build mode

Operators > Player > is player in build mode

Check if the given player is currently in build mode.

Appearance in Composition Pane

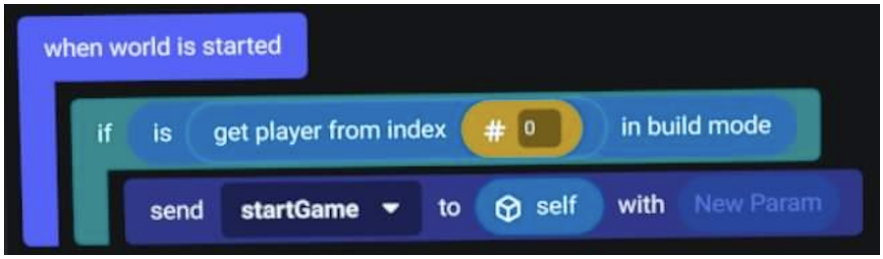
Description

This codeblock enables you to check whether a player is currently in build mode. You can use this codeblock to create debug-only code. So that you don't need to keep going in and out of play mode while building.

Parameters

player - the player to check

Example: If player at index 0 is in build mode automatically start the game



What it does: If you are building a game, it can take a lot of time to start the world, go into play mode, press a button to start the game, and then go back into build mode to then watch debug logs, for example. This example checks if the player at index 0 is in build mode and if so, starts the game immediately. When the world is published this code won't run since the codeblock will return false.

position of player

Operators > Player > position of player

The vector position of a player.

Appearance in Composition Pane

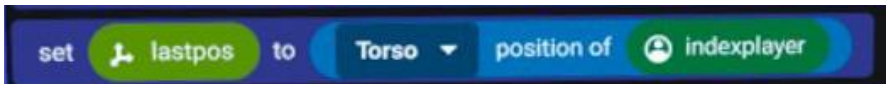
Description

Gets the position of a player. In the drop down box you can choose between the position of the head, foot, torso, left hand, and right hand to adjust which position is returned.

Parameters

Player

Example 1: Save the last position of a player



We can use **set to**, to save *lastpos* of the *indexplayer's* Torso.

physical velocity of player

Operators > Player > physical velocity of player

Returns the vector velocity of the player coming from physics-based movement. For example, when jumping or falling. This velocity does not include player movement from sliding or teleport locomotion.

Appearance in Composition Pane

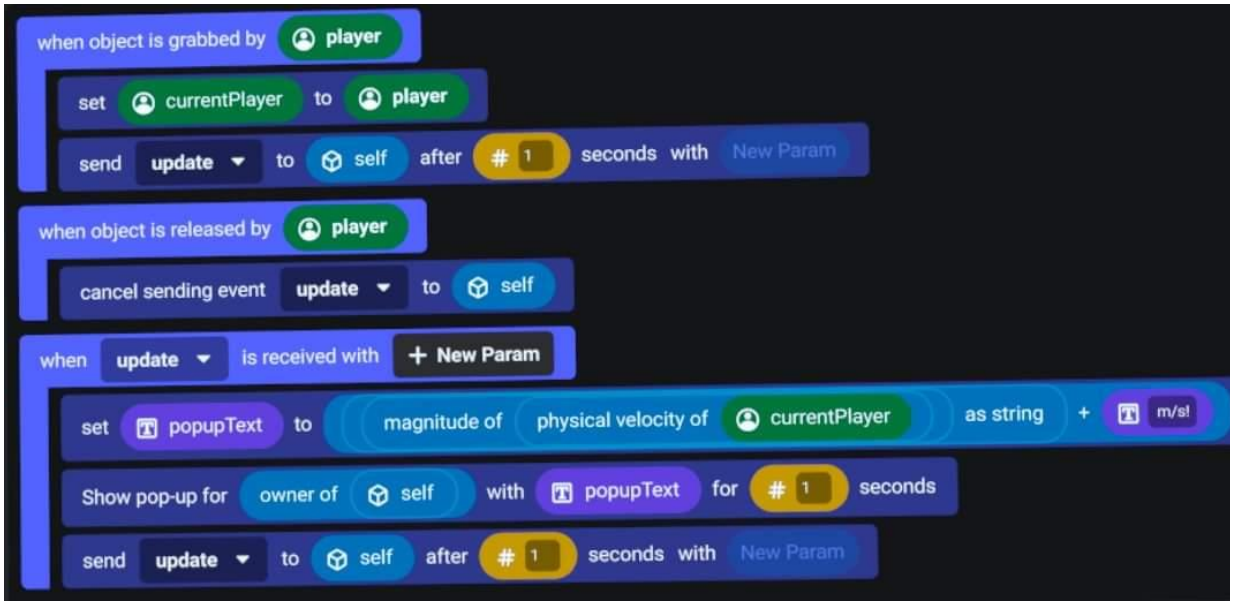
Description

This codeblock returns the velocity of the player coming only from physics. Any movement the person does in reality or avatar locomotion are not included. Additionally, this returns the value that the physics engine is trying to apply to the player, so if they are being pushed into a wall they may have a physical velocity even though the wall is pushing them back and they are not currently moving.

Parameters

player - the player to check

Example: Show a player their physical speed



What it does: Once a player grabs the object, they start getting pop ups every second telling them their physical velocity in meters per second, until they release the object.

How it works: Once the player grabs the object, we store the player and set the “update” event to run every second, and stop it when the player lets go. During the update event we get the player’s physical velocity and take the magnitude (to convert it to speed) and then show it to the player in a popup.

forward direction of player

Operators > Player > forward direction of player

A vector of the forward direction of a player.

Appearance in Composition Pane

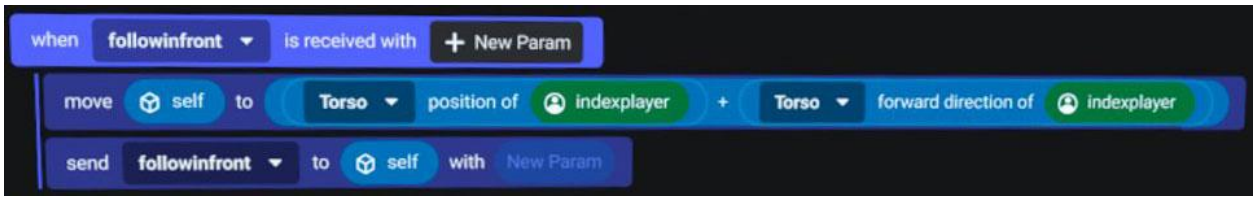
Description

Returns the forward direction of a player with a length of one meter. In the drop down box, choose between the forward direction of the head, foot, torso, left hand, or right hand. The forward direction of the foot and torso is based on the forward direction of the player's head, but removes the y from the direction making it horizontal facing.

Parameters

Player

Example 1: Follow In Front of a Player



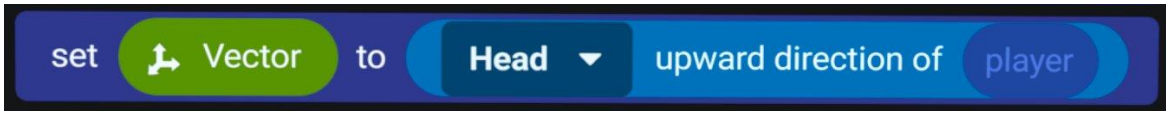
Each time the *followinfront* event is received, the object moves one meter in front of the player. The forward direction of the torso is added to the position of the player's torso, returning a new position that is 1 meter in front of the player.

upward direction of player

Operators > Player > upward direction of player

A vector of the upward direction of a player .

Appearance in Composition Pane



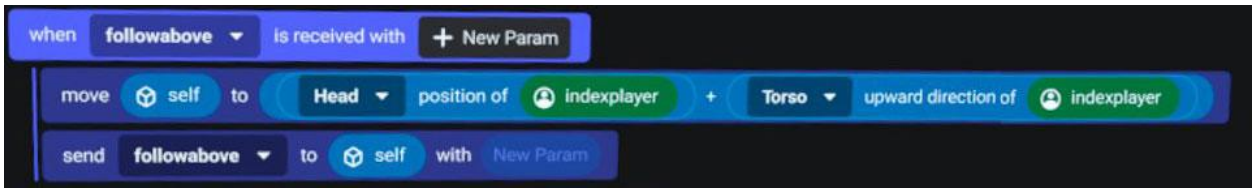
Description

Gets the upward direction of a player with a length of one meter. In the drop down box, choose between the upward direction of the head, foot, torso, left hand, or right hand.

Parameters

Player

Example 1: Follow above a player



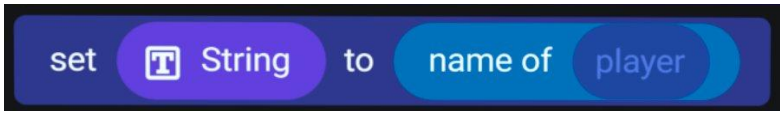
Each time the *followabove* event is received, the object moves one meter above the player's head. The upward direction of the torso is added to the position of the player's head, returning a new position that is directly above the player's head.

name of player

Operators > Player > name of player

Returns the username of a player.

Appearance in Composition Pane



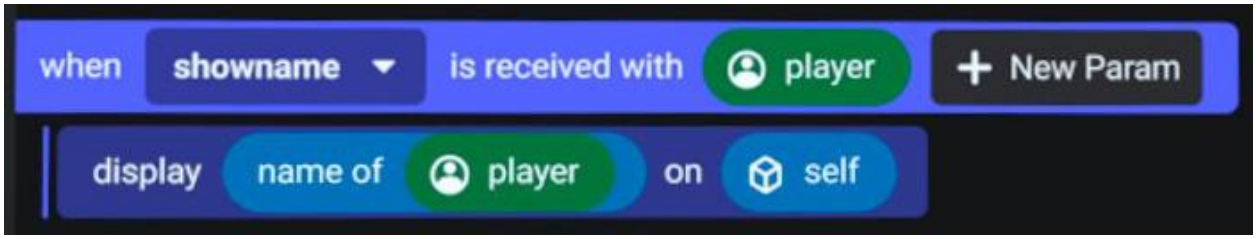
Description

Returns the username of a player as a string.

Parameters

Player

Example 1: Display the name of a player



When the *showname* event is received with *player,* as a parameter, it displays the username of the player on the text gizmo *self*. This could be used to create a welcome sign and more!

get player index

Operators > Player > get player index

The index of a player, defined when they enter the world.

Appearance in Composition Pane

Description

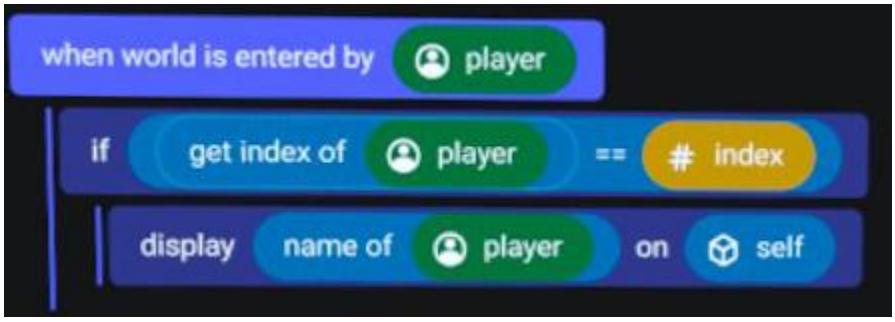
When a player enters the world, they are assigned the lowest available index. Each player will have a different index, and all player indexes will be less than the player capacity of the world. When a player leaves the world, their player index becomes available to the next player that enters the world. The index of a player does not change when another player leaves the world.

Parameters

Player

Returns a number that is the index of the player value.

Example 1: Display name of a player



When a player enters the world, if their player index equals the index variable, it displays their name on the text object the script is attached to. You can imagine this being on a jersey, locker, or even a tombstone!

get player from index

Operators > Player > get player from index

Returns the player id at the index provided.

Appearance in Composition Pane

Description

Each player has an index assigned to them based on when they enter the world. The 'get player from index' codeblock gives you the player variable that identifies the player with that index. If no player has the index given, a null player is returned.

Parameters

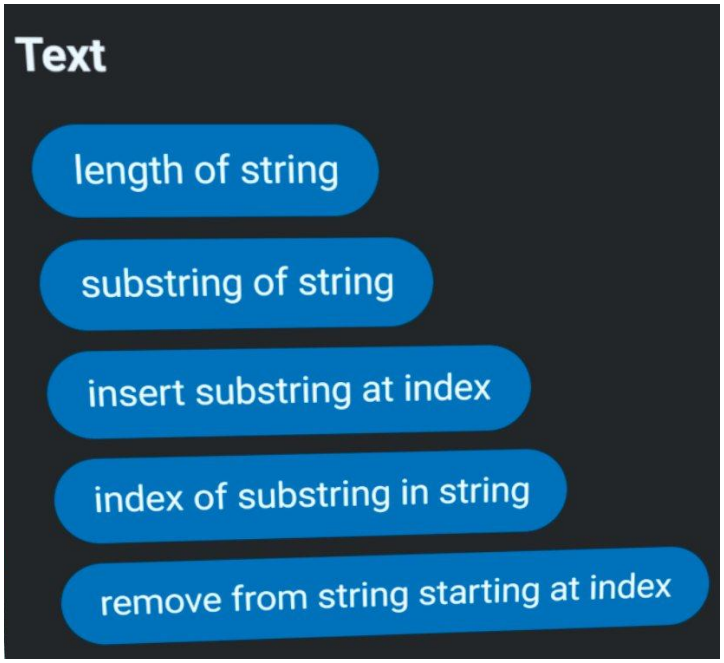
Number

Example 1: Respawn all the players in the world

When the *respawnplayers* event is received, the number *i* is incremented through a **while** loop to **respawn** each player from indexes 0 through 5. The **while** loop is written to run while *i* is less than 6 because this world has a maximum capacity of 6 players.

Text

Operators > Text



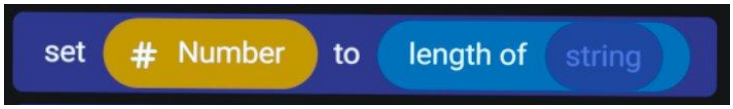
Codeblocks that can interact with strings.

length of string

Operators > Text > length of string

Length of a string

Appearance in Composition Pane



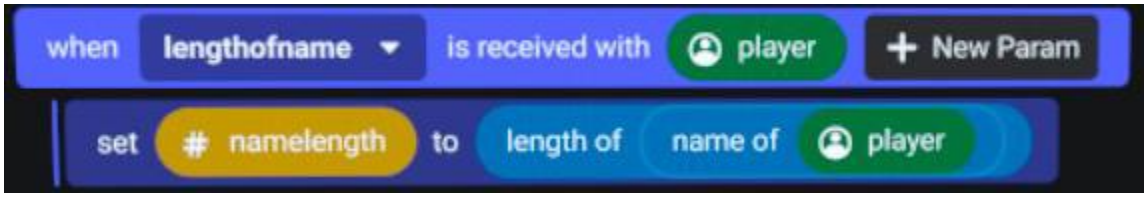
Description

Returns the number of characters in a string as a number value.

Parameters

String

Example 1: Length of a player's name



When we receive *player* as a parameter, we can set *namelength* to the **length of** player's name.

substring of string

Operators > Text > substring of string

Returns a new string from a starting index in a string with a given length.

Appearance in Composition Pane

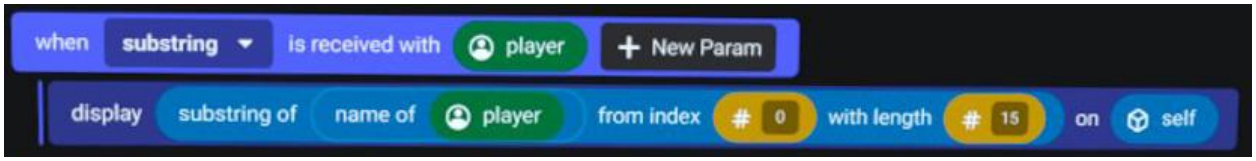
Description

Returns a substring of a string that starts at the index given and with the length given. The first letter of the string starts at index 0, and length of 1, returns a single character.

Parameters

String, number, number

Example 1: Substring from a String



When the *substring* event is received with *player* as a parameter, it displays the first 15 characters of the name of the player. This cuts the name short, if it is too long to display.

insert substring at index

Operators > Text > Insert Substring at Index

Inserts a string at a given index.

Appearance in Composition Pane

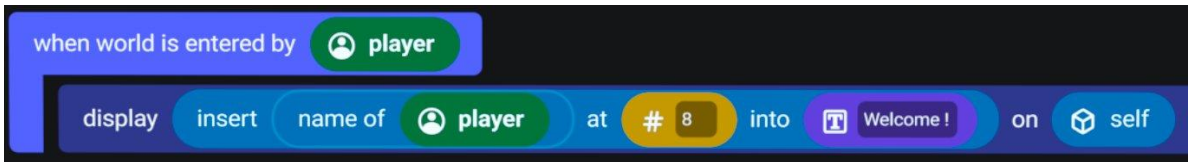
Description

Insert substring at index allows you to place a string into another string.

Parameters

String, Number, String

Example 1: Welcome Player



When a player enters the world, we insert their name into the welcome string one character from the end, placing their name after welcome, and leaving an exclamation point at the end.

index of substring in string

Operators > Text > Index of Substring in String

Returns a number, which is the index of the substring.

Appearance in Composition Pane

Description

Returns a number, which is the index of the substring. Can be used to search the string, or just to find out if it contains a value (see example 1 below). This is possible, because a -1 is returned if the substring is not found in the string.

Parameters

String, String

Example 1: Respawn players with an E in their name

When a player enters the world, if their name contains a capital or lowercase E, we respawn them to self.

remove from string starting at index

Operators > Text > Remove From String Starting at Index

Returns a string with a portion removed.

Appearance in Composition Pane

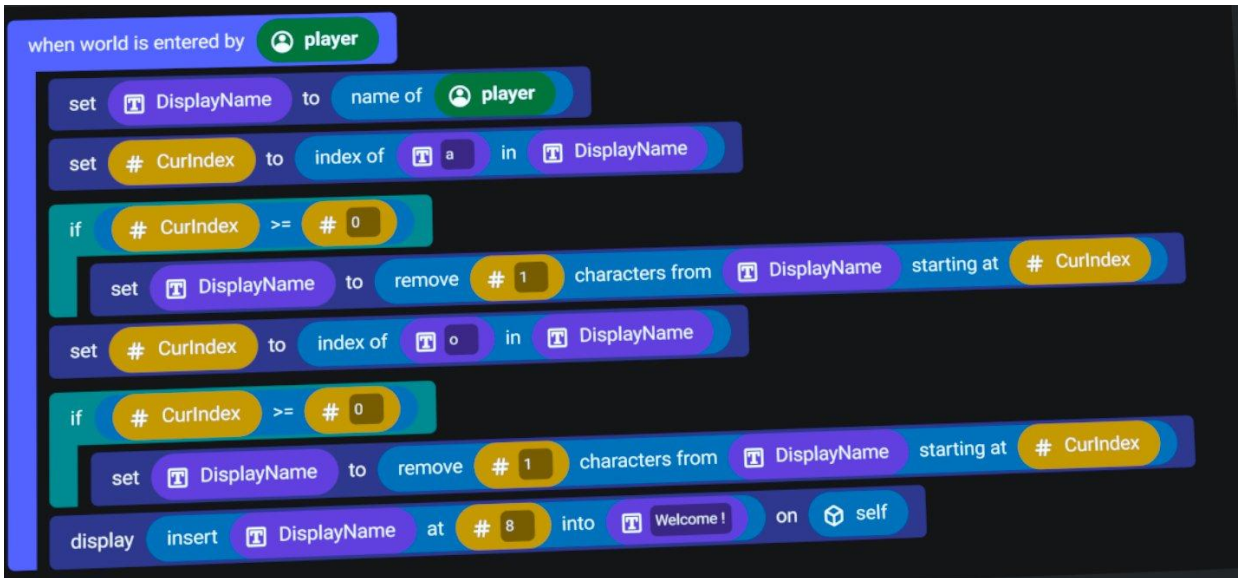
Description

Returns a string with a portion removed. Removal starts at the index provided and ends after the length given.

Parameters

String, Number, Number

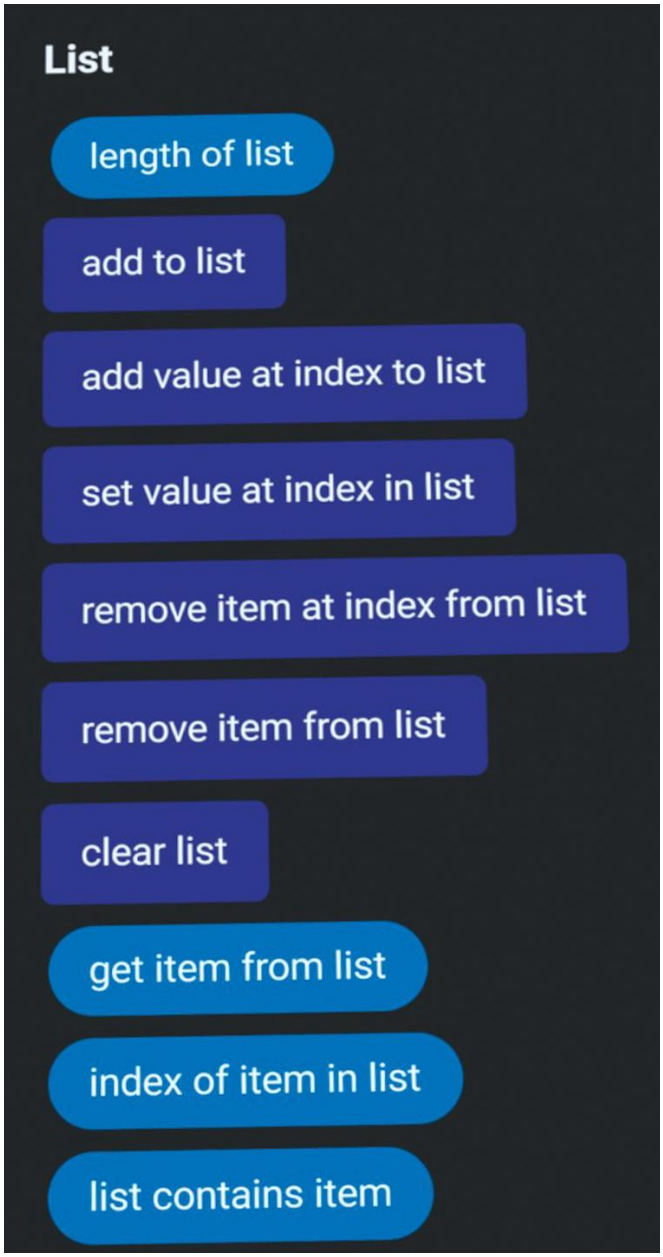
Example 1: Welcome Player, But Without Vowels



We welcome players on world enter, but remove the first lowercase a and o. Codeblocks can be duplicated for additional letters.

List

Operators > List



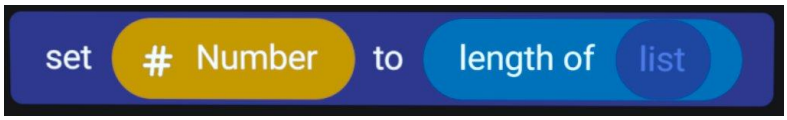
Codeblocks that interact with and manipulate lists.

length of list

Operators > List > length of list

The length of a list.

Appearance in Composition Pane



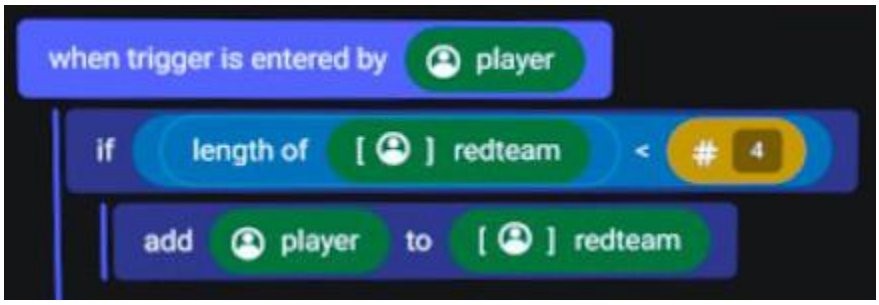
Description

The number of values in a list.

Parameters

list

Example 1: Add a player to list



When a player enters a trigger, it adds them to the *redteam* list, if there are less than 4 players in the list.

See Also

- Operators > List > add to list

add to list

Operators > List > add to list

Add a value to a list

Appearance in Composition Pane

Description

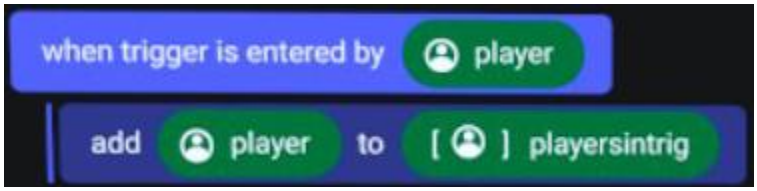
Adds a value to the end of the list. With its index being one less than the length after adding.

Parameters

Number, boolean, object, player, vector, rotation, color, or string | List

A value is added to a list of the same value type.

Example 1: Add player in trigger to a list



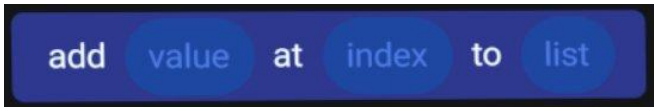
When a player enters the trigger, it adds them to the list *playersintrig*. Another event can remove the player when they leave the trigger, to prevent double entries in the list.

add value at index to list

Operators > List > add value at index to list

Add a value to a list at an index and push higher index items back one index.

Appearance in Composition Pane



Description

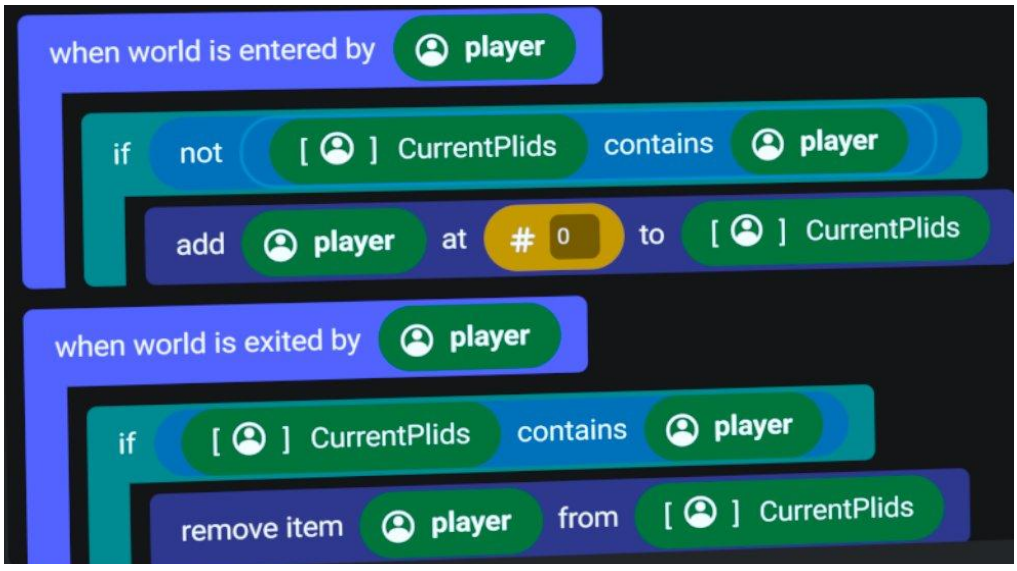
Adds a value to a specific spot in a list. Pushes remaining items back one index.

Parameters

Number, boolean, object, player, vector, rotation, color, or string | Number | List

A value is added at an index to a list of the same value type.

Example 1: Add player to the front of a list



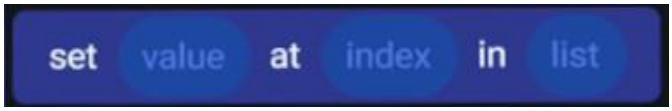
When a player enters the world, it adds them to the front of the list CurrentPlids.
*See Remove Item From List and List Contains Item.

set value at index in list

Operators > List > set value at index in list

Sets the value at the index in a list.

Appearance in Composition Pane



Description

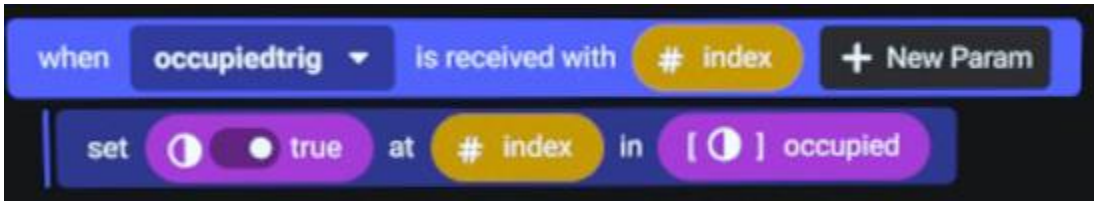
Changes the value of the item at the index in the list. This does not add more values to the list.

Parameters

value, number, list

Sets the *value* at the index *number* in a *list* of the same value type.

Example 1: Set a boolean in a list to true



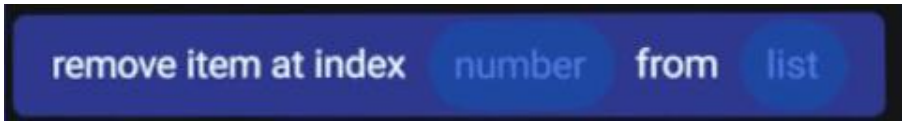
Here we have a custom event, *occupiedtrig,* when received it informs the script that a trigger at a specified index is occupied, and the list of booleans is updated.

remove item at index from list

Operators > List > remove item at index from list

Removes the item from a list at the index given.

Appearance in Composition Pane



Description

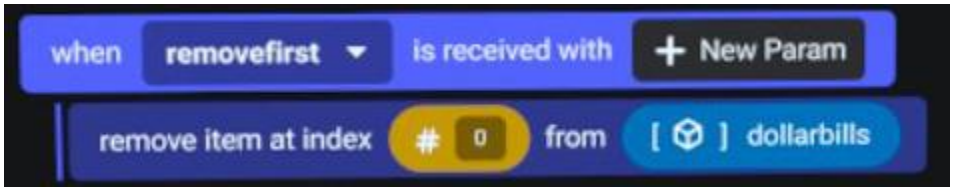
This codeblock allows us to remove an item in a list that is at a given index. When an item is removed from a list, all items with a higher index move down by one. If an index given is not in the list, an error will occur.

Parameters

number, list

Remove the item at the index number from the list.

Example 1: Remove the first item from a list



When the *removefirst* event is received, we **remove item at index** *0* from the list *dollarbills*.

remove item from list

Operators > List > Remove Item From List

Remove an item from a list.

Appearance in Composition Pane

Description

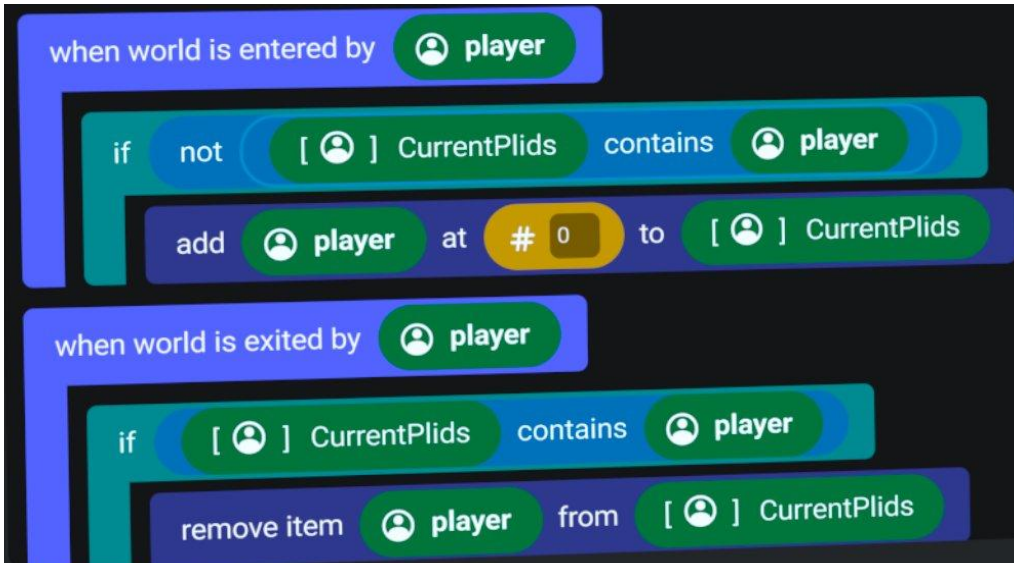
Removes an item from a list, if there are multiple of the same item, removes the item with the lowest index.

Parameters

Number, boolean, object, player, vector, rotation, color, or string | List

Remove a value from a list.

Example 1: Track current players



When a player enters the world, it adds them to the front of the list *CurrentPlids*. When they exit the world we remove that player from the list.

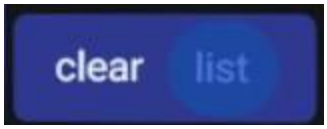
*See Add Value At Index To List and List Contains Item.

clear list

Operators > List > clear list

Removes all items in a list.

Appearance in Composition Pane



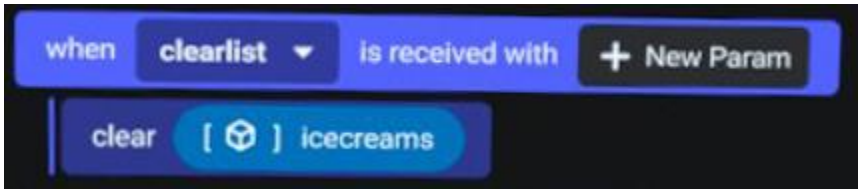
Description

Removes all the items from a list, setting its length to 0.

Parameters

list

Example 1: Clear a list of icecreams



When the *clearlist* event is received, all the objects in the *icecreams* list are removed.

get item from list

Operators > List > get item from list

Gets the item at the index from the list

Appearance in Composition Pane

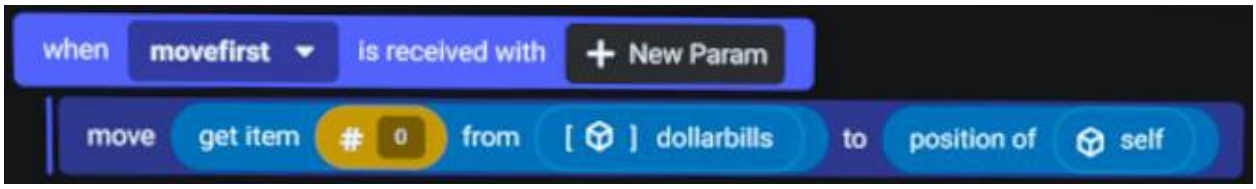
Description

Allows you to get the item at an index in the list, to be used as a value in other codeblocks.

Parameters

index, list

Example 1: move an object in a list



When the *movefirst* event is received, we **move** the object at index 0 in the *dollarbills* list.

index of item in list

Operators > List > index of item in list

The index of an item in a list.

Appearance in Composition Pane

Description

Gets the index of an item in a list. Returns -1 if the item is not in the list.

Parameters

value, list

Returns the value's index in the list as a number.

Example 1: Remove an Object From a List

When the *removeobject* event is received with *obj* as a parameter, it removes that object from the list by getting the **index of item in list**.

list contains item

Operators > List > List Contains Item

Returns a boolean (true/false), if an item is or is not in the list.

Appearance in Composition Pane

Description

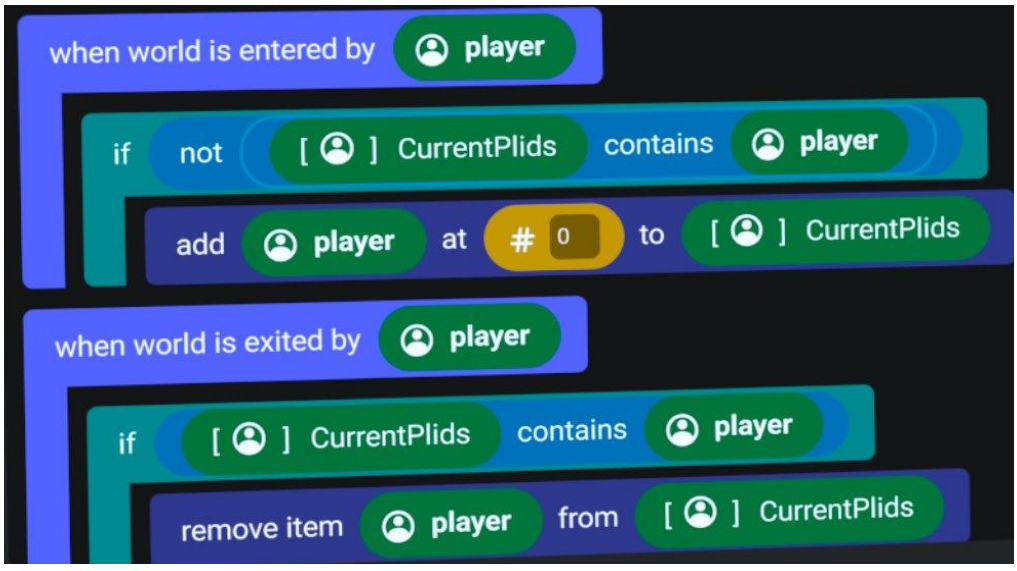
Returns a boolean (true/false), if an item is or is not in the list. Great for use in if statements.

Parameters

List | Number, boolean, object, player, vector, rotation, color, or string

Check if the list contains the value provided.

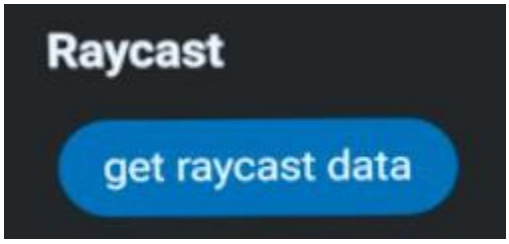
Example 1: Track current players



When a player enters the world, if they are not already in the list, we add them to the list. And when they exit the world we remove that player from the list if they are still in the list. This boolean check is a great way to make sure you don't have any errors occur in your script.
*See Add Value At Index To List and Remove Item From List.

Raycast

Operators > Raycast



Returns data from a raycast gizmo.

get raycast data

Operators > Raycast > get raycast data

Gets data from a raycast gizmo

Appearance in Composition Pane

Description

This codeblock returns data from a raycast gizmo. You will need to fire a **raycast** or **raycast with overrides** and reference the raycast gizmo object.

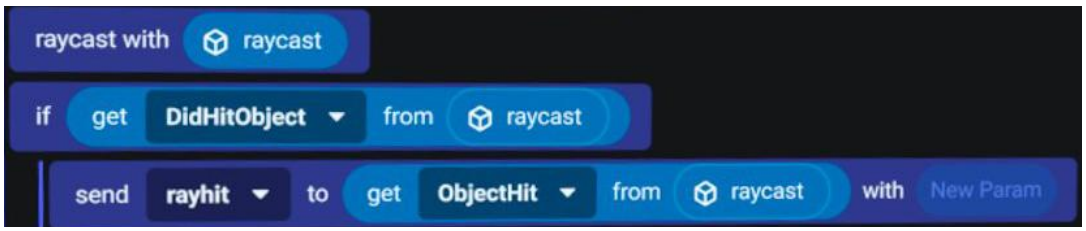
Parameters

Data, object

Returns a value based on the data requested and the raycast object specified.

DidHit: Returns the boolean true if the raycast beam hit something.
DidHitObject: Returns the boolean true if the raycast hits an object with a tag corresponding to the raycast's property panel configuration.
DidHitPlayer: Returns the boolean true if the raycast hits a player.
ObjectHit: Returns the object that the raycast hit.
PlayerHit: Returns the player that the raycast hit.
HitDistance: Returns a number that is the distance in meters between the raycast and object or player it hit.
HitPoint: Returns the vector position of the impact point on the object or player.
HitNormal: Returns the hit positions surface direction vector.
DidHitStatic: Returns false on players and true on objects.

Example 1: Send an event to a hit object



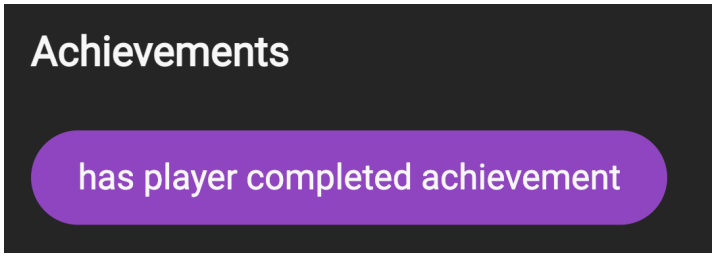
First we fire a **raycast**. We then use **get raycast data** to test **if** we **DidHitObject**. If that boolean is true, we send the *rayhit* event to the object that was hit.

See Also

- Actions > Raycast

Achievements

Operators > Achievements



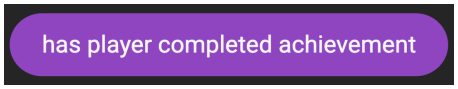
Acces achievement-related data.

has player completed achievement

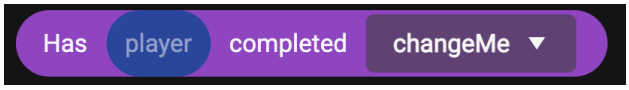
Operators > Achievements > has player completed achievement

Check if a player has completed an achievement.

Appearance in Library



Appearance in Composition Pane



Description

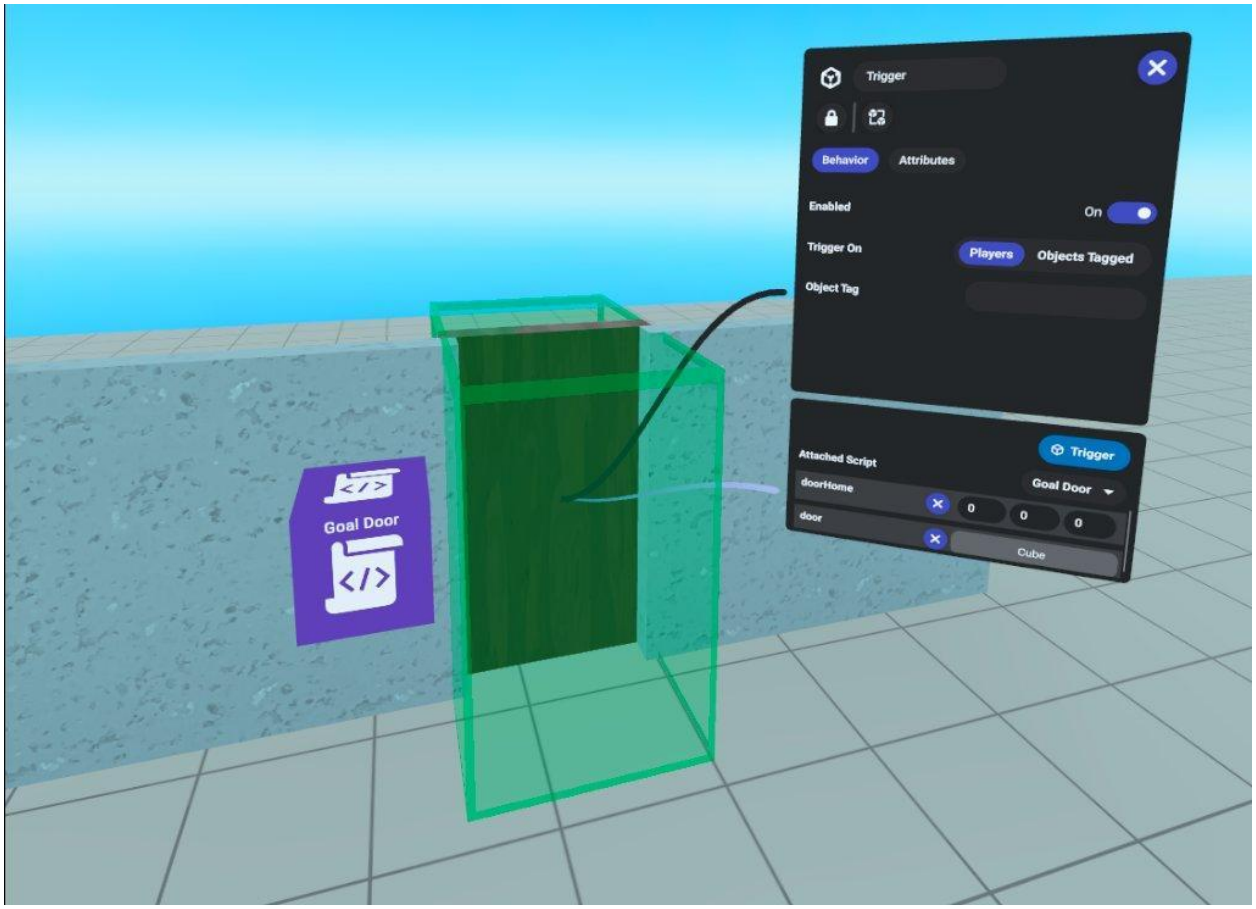
This operation returns a boolean value, and allows you to check whether an achievement has been previously completed for a particular player. This lets you decouple actions based on achievement completion from when they are actually completed.

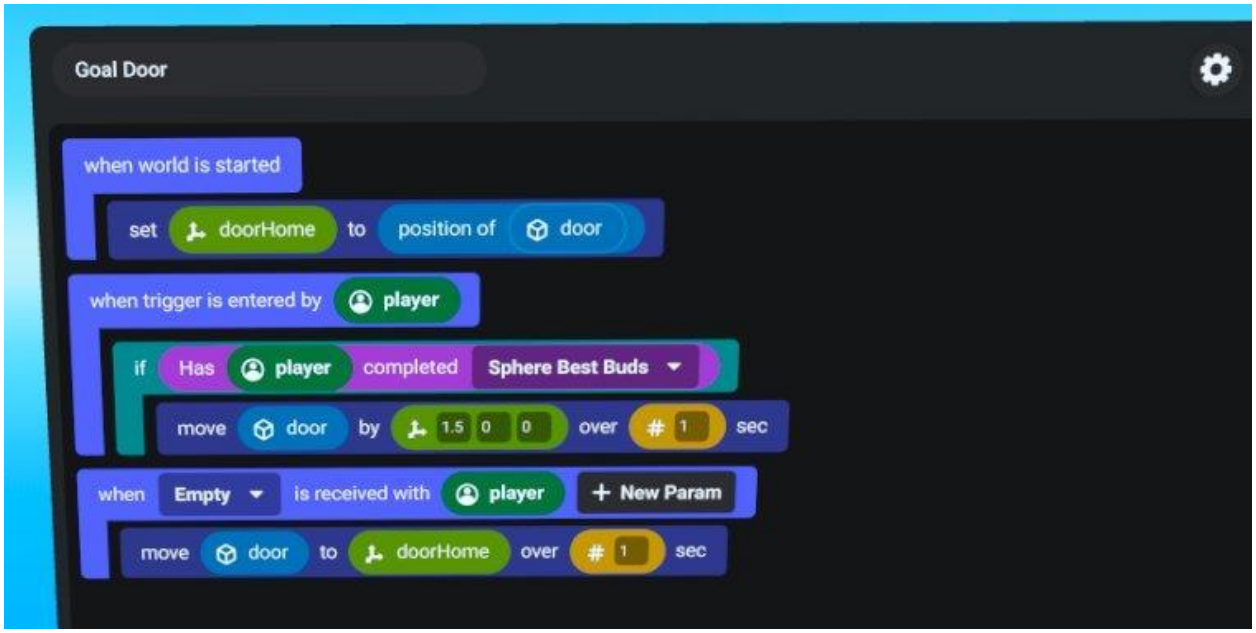
Parameters

player: the player you want to check.
achievement: the name of which achievement to check. Note that it is *not* the Script ID.

Example: Locked Door

In this example, we have a trigger in front of a door that only will open if the player has previously completed some achievement.


Trigger
Behavior
Attributes
Enabled
On
Trigger On
Players
Objects Tagged
Object Tag
Attached Script
Trigger
doorHome
Goal Door
0
0
0
door
Cube
Goal Door



When the trigger is entered by the player, we check if the player has completed the Achievement (looked up by Name, not ScriptID) at some point in the past, and if so, slide the door off to the side. When all players have left the trigger, the door is returned to its closed position.