

Math Operations

<code>x + y</code>	Adds x and y
<code>x - y</code>	Subtracts y from x
<code>x * y</code>	Multiplies the x by y
<code>x / y</code>	Divides x by y
<code>x % y</code>	Remainder of x divided by y

Math Functions

<code>math.random(x,y)</code>	Returns a random number between x and y
<code>math.floor(x)</code>	Rounds down x
<code>math.ceil(x)</code>	Rounds up x
<code>math.abs(x)</code>	Returns the absolute value of x
<code>math.sqrt(x)</code>	Returns the square of x
<code>math.pi</code>	Returns pi

Basic References

<code>.game</code>	Parent of all running game services
<code>.workspace</code>	References Workspace, circumvents <code>game.Workspace</code>
<code>.script</code>	References the script itself
<code>.parent</code>	References the parent of an object

For Loops

```
for i = start, end, count do
    --Code
end
for i = 1, 5 do
    print("Iteration : ", i)
end
local fruits = {"Apple ", "Banana ", "Cherry "}
```

For Loops (cont)

```
> for index, value in ipairs(fruits) do
    print(index, value)
end
```

While Loops

```
while condition do
    -- Code
end
local counter = 1
while counter <= 5 do
    print("Counter: ", counter)
    counter = counter + 1 -- Increment the counter to avoid an infinite loop
end
```

Wait routines

`wait()` Pauses the execution of the script for a set duration. Tied to `game framerate`

`task.wait()` Pauses the execution of the script for a set duration. More accurate, independent of `framerate`.

`task.delay()` Runs a function after a specified delay but does not block the current script.

`RunService.Heartbeat` Creates a custom wait loop. High precision, frame-dependent tasks.

`Debris.AdItem()` Used for timed object destruction but can act as a timer mechanism.

Basic Threading

`task.spawn()` Runs a function in a new thread after yielding briefly (~1/30th of a second)

`task.defer()` Schedules a function to run in a new thread without yielding at all. It is executed after the current thread completes.

`coroutine.create()` Creates a new coroutine that is paused until explicitly resumed.

`coroutine.resume()` Resumes a paused coroutine.

`coroutine.yield()` Pauses the current coroutine until it is resumed again.

`coroutine.status()` Returns the status of a coroutine ("suspended", "running", or "dead").

`coroutine.wrap()` Creates a coroutine and returns a function that, when called, resumes the coroutine.