SPACES EDU

TypeScript cheat sheet

PRO

```ts
let myColor = new Color(100, 0, 0);
// Defines a color with RGB values (100, 0, 0)

Debug.log(`The defined color is: ${myColor}`);
// Logs the defined color
```

```ts
let item1 = Scene.getItem("item1ID");
let item2 = Scene.getItem("item2ID");

Physics.physicsSpeed = 10;  // Sets the overall physics speed to 10

item2.physics.enabled = true;
item1.physics.enabled = true;

const distance = item1.center.sub(item2.center);
// Subtracts item2's position from item1's position to calculate the direction

item2.physics.applyImpulseLocal(Vector3.zero, distance);
// Pushes item2 towards item1 by applying an impulse in the direction of the distance
```

```ts
item.transform.rotate(Vector3.zero, new Vector3(0, 1, 0), -1);
// Rotates the object around its origin point
// along the Y-axis by -1 radian (clockwise)
```

```ts
let item1 = Scene.getItem("Item1");
let item1_slot = item1.getSlot('Top');

let item2 = Scene.getItem("Item2");
let item2_slot = item2.getSlot('Bottom');

item2_slot.attachTo(item1_slot);
// Attaches item2 to item1 using their respective slots
```

Last updated: October 2024

TypeScript code simply described and represented

```ts
const item = Scene.getItem("itemID") as AnimatedItem;

item.animation.play("Dance fun");
// Plays the animation named 'Dance fun'
```

Table of contents

Transform 3

Transitions 3

Position 5

Rotation 5

Scale 6

Actions 7

Generic 7

Sound 10

Video 11

Events 12

Input 12

Collision 12

Web 13

Other 13

Control 14

Loops 14

If 15

Other 16

Operators 17

Logic 17

Math 18

Items 20

Modify 20

Get 21

Data 23

Variables 23

Values 24

Physics

Simple 26
Advanced 27
Properties 27

Functions **29**

Simple 29

MERGE Cube **30**

Actions 30
Events 30
Simple 31

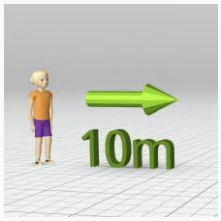
Notes **32**

Alternative code 32

Transform

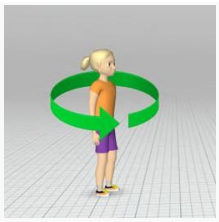
Transitions

BASIC



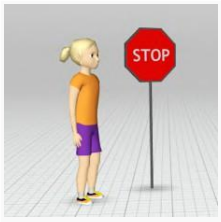
Make an object **move a certain distance** in a certain degrees on the z-axis in 1

```
const item = Scene.getItem("myItem");

item.transition.moveBy(new Vector3(10, 0, 0), 2);
// Moves the item by 10 meters along the X-axis over 2 seconds
```



Make an object or a character **turn** over time

```
const item = Scene.getItem("myItem");

item.transition.rotateLocal(new Vector3(0, 0, 1), 3, 1);
// Rotates the item locally around the Z-axis by 3 degrees over 1 second
```



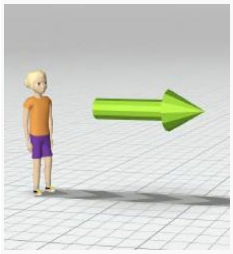
Stop an object or a character

```
const item = Scene.getItem("myItem");
item.transition.stop();  // Stops the object's current movement or transition
```

PRO

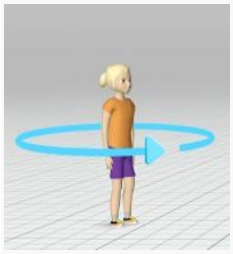
Make an object **move to a certain point** over time

```
const item = Scene.getItem("myItem");

item.transition.moveTo(new Vector3(5, 10, 3), 3);
// Moves the object to position (5, 10, 3) over 3 seconds
```



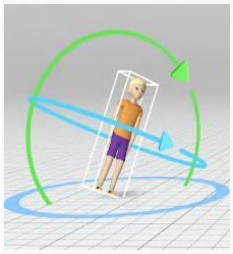
Make an object **move on a path** over time

```
const myItem = Scene.getItem("myItem");
const myPath = Scene.getItem("myPath") as PathItem;

myItem.transition.moveOnPath({
    path: myPath,
    speed: 1.5,
    turnWithPath: true,
    repeat: true
});
// Moves the item along the path with a speed of 1.5, following the path indefinitely
```



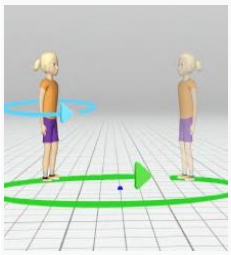
Make an object **turn of a certain angle**

```
const item = Scene.getItem("objectID");

item.transform.rotate(Vector3.zero, new Vector3(0, 0, 1), Math.PI);
// Rotates the object around its origin point
// along the Z-axis by 180 degrees (Math.PI radians)
```



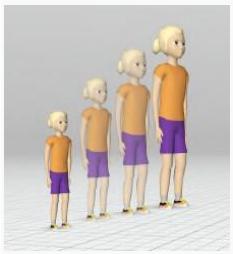
Make an object **turn around an axis**

```
item.transform.rotate(Vector3.zero, new Vector3(0, 1, 0), -1);
// Rotates the object around its origin point
// along the Y-axis by -1 radian (clockwise)
```



Make an object **turn relative to a point** in a certain direction

```
const item = Scene.getItem("ObjectID");

item.transform.rotateLocal(new Vector3(0, 2, 0), new Vector3(0, 0, 1), -180);
// Rotates the object with an offset of (0, 2, 0) along the Z-axis by -180 degrees
```

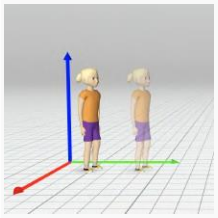


Change the **size** of an object over time

```
const item = Scene.getItem("objectID");

Time.scheduleRepeating(() => {
    item.transform.multScale(1.2);
}, 1);
// Increases the size of the item by 20% each second
```

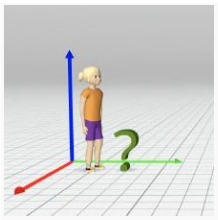
Position

PRO



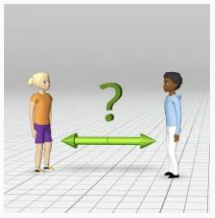
Change the **position** of an object

```
const item = Scene.getItem("myItem");

item.transform.position = new Vector3(0, 0, 0);
// Sets the object's position to (0, 0, 0)
```



Get the **position** of an object

```
const item = Scene.getItem("objectID");

Debug.log(`Item's position is: ${item.transform.position}`);
// Logs the position of the item to the Debug console
```

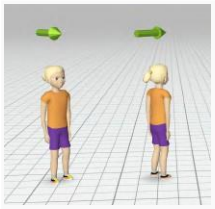


Get the **distance** between 2 objects

```
const item1 = Scene.getItem("item1ID");
const item2 = Scene.getItem("item2ID");

let distance = item1.center.dist(item2.center);
Debug.log("The distance between the items is " + distance);
// Calculates and logs the distance between item1 and item2
```

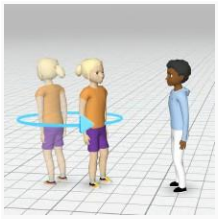
Rotation

PRO



Change the **direction** of an object

```
const item = Scene.getItem("objectID");
let newDirection = new Vector3(0, -1, 0);

item.transform.setDirection(newDirection);
// Sets the object's direction to (0, -1, 0) along the Y-axis
```

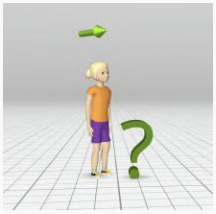


Make an object **turn towards another object**

```
const item1 = Scene.getItem("item1ID");
const item2 = Scene.getItem("item2ID");

item1.transform.lookAt(item2.transform.position);
// Makes item1 rotate to face item2's position
```

Make an object **turn towards certain position**

```
const item = Scene.getItem("itemID");

item.transform.lookAt(new Vector3(0, 0, 0));
// Makes the item rotate to face the point (0, 0, 0)
```



Get the **direction** of an object

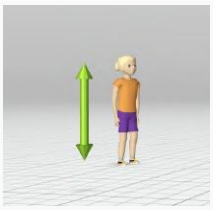
```
const item = Scene.getItem("itemID");
let originalDirection = item.transform.axisY;

Debug.log("Item direction is " + originalDirection);
// Open the console debugger </> to see the object's Y-axis direction
```

Scale

PRO

Set the **size** of an object

```
const object = Scene.getItem("ObjectID");

object.transform.scale = 0.5;
// Scales the object down to 50% of its original size
```



Get the **size** of an object

```
let item = Scene.getItem("itemID");

Debug.log(`Item's size is: ${item.transform.scale}`);
// Logs the current scale (size) of the item to the Debug console
```

Actions

Generic

BASIC

Make a character **say** or **think** something

```
const character = Scene.getItem("myCharacter");
character.speech = "Hi!";  // Character says "Hi!"

character.thought = "Hmm";  // Character thinks "Hmm"
```

Change the **color** of an item

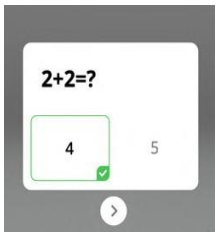
```
const item = Scene.getItem("objectID");

item.color = Color.blue;
// Changes the item's color to blue
```

Change the **opacity** of an item

```
const item = Scene.getItem("myItem");
item.opacity = 0.5;  // Sets the opacity of the item to 50%
```

Show an **info panel** with a title, text (and an image)

```
GUI.HUD.showInfoPanel({
    title: "Title",
    text: "Text",
    image: "imageUrl"  // Optional: Add image URL or leave it out for no image
});
```



Show a **quiz panel** with a question and answers that can be clicked and will trigger an action when the selected answer is correct or incorrect.

```
GUI.HUD.showQuizPanel({
    question: 'Your question here',
    answer1: 'Answer 1',
    answer2: 'Answer 2',
    answer3: 'Answer 3',
    answer4: 'Answer 4',
    correctAnswer: 1,  // Change to the correct answer number
    onCorrect: () => {
        // Action to perform on correct answer
    },
    onWrong: () => {
        // Action to perform on wrong answer
    }
});
```

Show a **choice panel** with a question and options that can be clicked and that trigger different actions.

```
let choice = 0;
GUI.HUD.showChoicePanel({
    question: 'This is your question.',
    answer1: 'First answer.',
    answer2: 'Second answer.',
    onAnswer: answer => {
        choice = answer;
        if (choice === 1) {
            // Action for Option1
        }
        else if (choice === 2) {
            // Action for Option2
        }
    }
});
```

PRO

Play or stop the **animation** of an object

```
const item = Scene.getItem("itemID") as AnimatedItem;

item.animation.play("None");
// Plays the animation named 'None'
```

Make an object **say** something for a certain duration

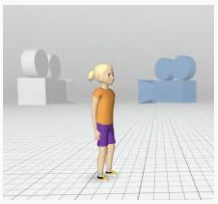
```
function timedSpeech(char, sentence: string, duration: number) {
    char.speech = sentence;
    Time.schedule(() => {
        char.speech = null;
    }, duration);
}

timedSpeech(item, "Hi", 2);
// Makes the item say "Hi" for 2 seconds
```

Make an object **think** something for a certain duration

```
function timedThought(char, sentence: string, duration: number) {
    char.thought = sentence;
    Time.schedule(() => {
        char.thought = null;
    }, duration);
}

timedThought(item, "Hmm", 2);
// Makes the item think "Hmm" for 2 seconds
```

Define the **text** of a text object

```
let textObject = Scene.getItem("objectID") as Text3DItem;

textObject.text = "my text";
// Sets the text of the 3D text object to "my text"
```

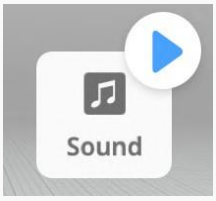


Switch to a different **camera** in your scene

```
let cam2 = Scene.getItem("cameraID") as CameraItem;

Camera.focusedItem = cam2;
// Defines the CameraItem that the client is currently looking through
```

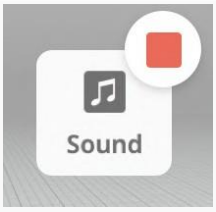
Sound

BASIC



Play a **sound** file

```
const music = Sound.load("soundID");
music.play();  // Play the sound
```

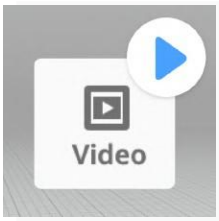


Stop playing the **sound** file

```
music.stop();  // Stop the sound
```

Video

PRO

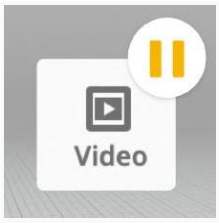


Start playing a video and choose to wait for the video to end or not before next actions

```
let item = Scene.getItem("clipID") as VideoItem;

item.video.play();
// Plays the video associated with the VideoItem
```

Stop playing a certain video

```
let item = Scene.getItem("clipID") as VideoItem;

item.video.stop();
// Stops the video associated with the VideoItem
```



Pause a certain video

```
let item = Scene.getItem("clipID") as VideoItem;

item.video.pause();
// Pauses the video associated with the VideoItem
```

Events

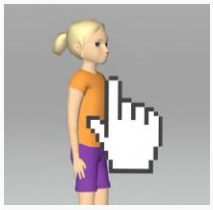
Input

BASIC

Make something happen **when** an item is **clicked**

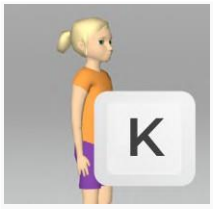
```
const item = Scene.getItem("myItem");

item.input.onClick(() => {
    // Action to perform when clicked on
});
// Executes the action when the item is clicked
```

PRO



Define what happens when a certain **object** is **hovered over**

```
const item = Scene.getItem("myItem");

item.input.onHover(() => {
    // Action to perform when hovered over
});
// Executes the action when the item is hovered over
```

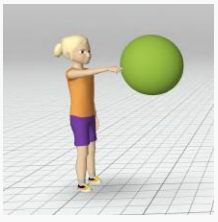


Define what happens when a certain **key** is **pressed**

```
Input.onKeyPressed(() => {
    // Action or function to perform
}, "k");
// Executes the action when the "k" key is pressed
```

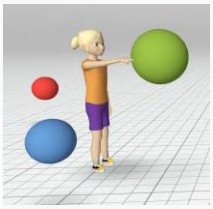
Collision

PRO



Define what happens when a certain **object collides with another object**

```
const item = Scene.getItem("itemID");

item.onCollisionEnter(() => {
    // Action or function to perform
});
// Executes the action when the item collides with another object
```

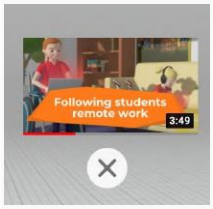


Define what happens when a certain **object collides with any other object** and no longer collides with it

```
const item = Scene.getItem("itemID");

item.onCollisionExit(() => {
    // Action or function to perform
});
// Executes the action when the item is no longer colliding with another object
```

Web

PRO



Show a YouTube video when the object is **clicked.**

Currently not supported in Typescript - only via Debugger (ctrl+click)

```
const item = Scene.getItem("itemID");

item.input.onClick(() => {
    Debug.log("Follow the link: https://youtu.be/15Vlqe22_x0?si=oyXvOYrqFYjihSjD");
});
// Logs a message with a YouTube video link in the debugger when the item is clicked
```

Open a website when the object is **clicked**

Currently not supported in Typescript - only via Debugger (ctrl+click)

```
const item = Scene.getItem("myItem");

item.input.onClick(() => {
    Debug.log("Follow the link: https://edu.cospaces.io");
});
// Logs the message with the link when the item is clicked
```

Removes existing **events** on an object (e.g. when this object is **clicked or hovered**)

```
const item = Scene.getItem("myItem");

item.input.onClick(() => {
    // Action to perform when clicked on

    item.input.onClick(null);
    // Disables further onClick events by setting the event handler to null
});
// Executes the action and disables further onClick events
```

```
const item = Scene.getItem("myItem");

item.input.onHover(() => {
    // Action to perform when hovered over

    item.input.onHover(null);
    // Disables further onHover events by setting the event handler to null
});
// Executes the action and disables further onHover events
```

Control

Loops

BASIC

Make one or more actions **repeat** a certain amount of times

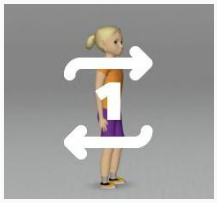
```
for (let i = 0; i < X; i++) {
    // Repeated action
}
```

Repeats the action **x** times

PRO

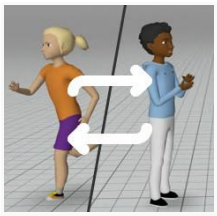
Make an action happen **forever** as a **loop**

```
Time.scheduleRepeating(() => {
    // Action or function to perform repeatedly
}, 1);
// Repeats forever, each second
```



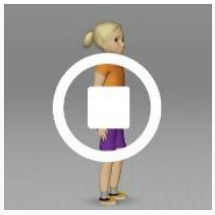
Make an action **repeat** as a loop for a **certain amount** of times. Use every repetition step as a **variable**.

```
let step = 1;

for (let i = 0; i < 100; i = i + step) {
    // Action or function to perform
}
// Repeats the action 100 times, increasing by 'step' each time
```



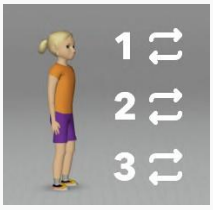
Make an action **repeat** as a loop **while** a condition is **true**

```
let condition = true;
if (condition = true) {
    // Action or function to perform
}
// Performs the action while 'condition' is true
```



Stop a loop from **repeating**

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break;  // Stops the loop when i equals 5
    }
    // Action or function to perform
}
```



Make an action **repeat for each entry** in a list

```
const item1 = Scene.getItem("item1ID");
const item2 = Scene.getItem("item2ID");
const item3 = Scene.getItem("item3ID");

let list = [item1, item2, item3];
list.forEach(element => {
    // Action or function to perform on each item
});
// Iterates through the list and performs the action on each item
```

If

PRO

Make an action **happen only if** a certain condition is true

```
if (1 < 2) {
    // Action or function to perform if 1 is less than 2
}
// Executes the action because 1 is less than 2
```

Make an action **happen only if** a certain condition is true. **Else**, make **another action** happen

```
if (1 < 2) {
    // Action or function to perform if 1 is less than 2
} else {
    // Action or function to perform if the condition is false
}
// If 1 is less than 2, the first action is executed, otherwise the second
```

Other

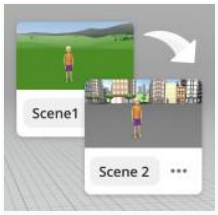
BASIC

Make my program **wait** for some time

```
Time.schedule(() => {
    // Perform action or function after 3 seconds
}, 3);
// Makes the program wait for 3 seconds
```

Go to specific scene

```
Space.goToScene(4);
// Switches to the scene with ID 4
```



Switch to a different **scene**

```
Application.onSceneSwitch(() => {
    // Action or function to perform
});
// Executes the action when the scene is switched
```

Quit the CoSpace and optionally show an image

```
Application.quit();
// Quit the CoSpace
```

Hide or show arrows to **switch scenes**

```
GUI.HUD.sceneNavigationVisible = false;
// Hide arrows to switch scenes
```

Operators

Logic

PRO

Enter an **arithmetic condition**

```
if (1 === 10) {
    // Action or function to perform if 1 equals 10
}
// This condition will evaluate as false since 1 does not equal 10
```

Enter a **logical condition**

```
if (1 < 2 && true) {
    // Action or function to perform if both conditions are true
}
// The action will be performed because both conditions are true
```

Check if something **isn't true**

```
if (!false) {
    // Action or function to perform if the condition is NOT true
}
// The action will be performed because 'false' is NOT true
```

Check if something is **true** and **return a certain value if it is or isn't**

```
let result = false ? 1 : 2;
// Returns 1 if true, otherwise returns 2

Debug.log(result);
// Logs the result
```

Math

PRO

Get a **random number** within a certain range

```
function getRandomInt(min: number, max: number): number {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

let randomNumber = getRandomInt(0, 100);
// Generates a random integer between 0 and 100
```

Get a **random integer number** within a certain range

```
function getRandomInt(min: number, max: number): number {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

let randomNumber = getRandomInt(0, 100);
// Generates a random integer between 0 and 100

let myNumber = randomNumber.toFixed(0);
// Formats the number with 0 digits after the decimal point
```

Enter an **arithmetic condition**

```
let result = 1 + 1;
// Adds 1 and 1, storing the result

Debug.log(result);
// Logs the result
```

Limit a number **within a certain range of numbers**

```
let x = 5;
function constrain(number: number, min: number, max: number): number {
    return Math.min(Math.max(number, min), max);
}

let constrainedValue = constrain(x, 1, 100);
// Constrains the value of x between 1 and 100

Debug.log(constrainedValue);
// Logs 5, since it's within the range
```

Check if a certain number is **even or odd**. Returns true for even, false for odd.

```
function isEven(number: number): boolean {
    return number % 2 === 0;
}

let result = isEven(3);
// Checks if 3 is even

Debug.log(result);
// Logs false because 3 is not even
```

Check if a certain number is **divisible by another number**. Returns true if it is.

```
function isDivisibleBy(number: number, divisor: number): boolean {
    return number % divisor === 0;
}

let result = isDivisibleBy(0, 3);
// Checks if 0 is divisible by 3

Debug.log(result);
// Logs true because 0 is divisible by any non-zero number
```

Get the **remainder of a division**

```
let remainder = 0 % 2;
// Gets the remainder of the division of 0 by 2

Debug.log(remainder);
// Logs 0 because 0 divided by 2 has no remainder
```

Get the **rounded value** of a certain number

```
let myNumber = randomNumber.toFixed(0);
// Formats the number with 0 digits after the decimal point
```

Gets the **rounded to 2 decimals value** of a certain number

```
let myNumber = 0;
let roundedNumber = myNumber.toFixed(2);
// Rounds the number to 2 decimal places

Debug.log(roundedNumber);
// Logs the rounded value
```

Get the **square root** of a certain number

```
let myNumber = 0;
let squareRoot = Math.sqrt(myNumber);
// Gets the square root of myNumber

Debug.log(`Square root is: ${squareRoot}`);
// Logs the square root value
```

Get the result of a **trigonometric operation**

```
let myNumber = 0;
let sineValue = Math.sin(myNumber);
// Gets the sine of myNumber (in radians)

Debug.log(`Sine of the number is: ${sineValue}`);
// Logs the sine value
```

Get the **sum** of a list of variables

```
let myVariable = [1, 2, 3, 4, 5];
let sum = myVariable.reduce((accumulator, currentValue) => accumulator + currentValue, 0)
// Sums all the values in the list

Debug.log(`The sum is: ${sum}`);
// Logs the sum of the list
```

Items

Modify

PRO

Add the **child** of an object to another object

```
const item1 = Scene.getItem("item1ID");
const item2 = Scene.getItem("item2ID");

item1.add(item2);
// Adds item2 as a child of item1
```

Attach an object to another object

```
let item1 = Scene.getItem("Item1");
let item1_slot = item1.getSlot('Top');

let item2 = Scene.getItem("Item2");
let item2_slot = item2.getSlot('Bottom');

item2_slot.attachTo(item1_slot);
// Attaches item2 to item1 using their respective slots
```

Detach an object from the object it's attached to

```
let item = Scene.getItem("itemID");

item.removeFromParent();
// Detaches the object from the parent it's attached to
```

Delete an object

```
let item = Scene.getItem("itemID");

item.delete();
// Deletes the object
```

Delete all children of an object

```
let item = Scene.getItem("itemID");

item.deleteChildren();
// Deletes all children of the object
```

Activate physics on an object

```
item.physics.enabled = true;
// Enables physics for the object
```

Disable physics on an object

```
let item = Scene.getItem("itemID");

item.physics.enabled = false;
// Disables physics for the object
```

Add an **object** at a certain position and with a certain name

```
let capsule = Scene.createCapsule(0, 0, 0);
// Creates a capsule at position (0, 0, 0)

capsule.name = "MyCapsule";
// Sets the name of the capsule
```

Change the **name** of an object

```
let item = Scene.getItem("itemID");

item.name = "MyItem";
// Sets the name of the item to "MyItem"
```

Get

BASIC

Get a certain **item**

```
let item = Scene.getItem("itemID");
```

Get a certain **group item**

```
const myGroup = Scene.getItem("GroupID");
// Retrieves the group item with the specified GroupID
```

PRO

Get a certain **camera** object

```
const MyCam = Scene.getItem("cameraID") as CameraItem;
// Retrieves the camera object with the specified cameraID
```

Get a certain **path** object

```
const MyPath = Scene.getItem("pathID") as PathItem;
// Retrieves the path object with the specified pathID
```

Get a certain **text** object

```
const myText = Scene.getItem("textID") as TextItem;
// Retrieves the text object with the specified textID
```

Get a certain **3D text** object

```
const myTextin3D = Scene.getItem("3dtextID") as Text3DItem;
// Retrieves the 3D text object with the specified 3dtextID
```

Get a certain **video**

```
const myVideo = Scene.getItem("videoID") as VideoItem;
// Retrieves the video object with the specified videoID
```

Get an object of a certain **name**

```
let item = Scene.getItem("Camel");
// Retrieves the object named "Camel"
```

Get the name of a certain **object**

```
let name = item.name;
// Retrieves the name of the object
```

Duplicate a certain object

```
let item = Scene.getItem("itemID");

item.copy();
// Duplicates the object
```

Get the **parent** of a certain object

```
let item = Scene.getItem("itemID");
let parent = item.parent;
// Gets the parent of the object
```

Get the **number of children** of a certain object

```
let item = Scene.getItem("itemID");
let childrenCount = item.children.length;
```

Get a certain **object's child** with index 0

```
let item = Scene.getItem("itemID");
let child = item.children[0];
```

Logs the child with index 0 of the object

```
Debug.log(`Child with index 0 is: ${child.name}`);
// Gets and logs the child with index 0 of the object
```

Check whether a certain **video is playing**

```
const myVideo = Scene.getItem("videoID") as VideoItem;
let checkVideo = myVideo.video.playing;

if (checkVideo === true) {
    Debug.log("The video is playing");
} else {
    Debug.log("The video is not playing");
}
// Checks whether the video is currently playing and logs the result
```

Data

Variables

PRO

Create a variable with a certain initial value

```
let myVar = "";
// Initializes the variable myVar with an empty string

Debug.log(`The initial value of myVar is: '${myVar}'`);
// Logs the initial value of the variable
```

Store a certain **CoSpace variable** under a certain name in order to reuse it in another scene *

```
let key = "lives";
let value = "3";
Space.setProperty(key, value);
// Stores the value "3" under the key "lives" in the CoSpace

Debug.log(`Stored property: ${key} = ${value}`);
// Logs the key-value pair stored in the CoSpace
```

Get the **stored CoSpace value** *

```
let key = "lives";
let storedValue = Space.getProperty(key);
// Retrieves the stored value associated with the key "lives"

Debug.log(`The stored value for ${key} is: ${storedValue}`);
// Logs the retrieved value for the key
```

Increase or decrease the value of a **variable**

```
let myVariable = 0;  // Define your variable with an initial value
myVariable += 1;     // Increment the variable by 1
Debug.log(myVariable);  // Logs the current value of myVariable (which is 1)
```

Values

PRO

Use **false** (or true)

```
let condition = false;
```

Use a **certain number**

```
let myNumber = 0;
```

Use a certain **mathematical constant** (like π)

```
let pi = Math.PI;
// Uses the mathematical constant Pi (π)
```

Use **no value**

```
let anything = null;
// Assigns no value (null) to the variable

Debug.log(`The value is: ${anything}`);
// Logs 'null' because the variable has no value
```

Use a **random color**

```
let myColor = Color.random();
// Generates a random color and assigns it to myColor
```

Get the **color** of a certain object

```
const item = Scene.getItem("cuboidID") as Cuboid;
let itemColor = item.color;
// Retrieves the color of the cuboid and assigns it to itemColor
```

Use a certain **color** you pick

```
let myColor = Color.white;
// Assigns the color white to myColor
```

Use a certain **color** you define with its **RGB** values

```
let myColor = new Color(100, 0, 0);
// Defines a color with RGB values (100, 0, 0)
```

Create a **text string** with text you define

```
let myString = "ABC";
// Creates a string with the value "ABC"

Debug.log(`The string is: ${myString}`);
// Logs the string
```

Use a **text** that you define

```
let myText = Scene.getItem("textID") as TextItem;
myText.text = "Lives";
// Sets the text of the TextItem to "Lives"
```

Use certain **coordinates**

```
let myCoordinates = new Vector3(0, 0, 0);
// Defines a coordinate with x: 0, y: 0, z: 0
```

Use a certain **coordinate**

```
let xValue = myCoordinates.x;
// Retrieves the x value of the coordinates
```

Use the **length** of a certain **variable**

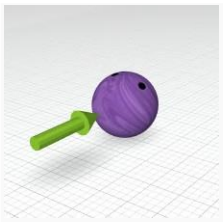
```
let myVariable = [1, 2, 3, 4, 5];
// Defines a variable as an array with 5 elements

let lengthOfVariable = myVariable.length;
// Gets the length of the array

Debug.log(`The length of the variable is: ${lengthOfVariable}`);
// Logs the length of the variable
```

Physics

Note: To use physics on an item, **`enable` physics** with the item inspector. By default, items ignore physics collisions and forces.

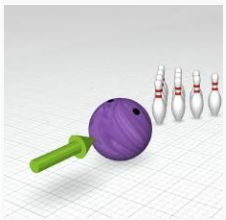
Simple

PRO



Push a certain object **in** a certain **direction**

```
let item = Scene.getItem("objectID");
item.physics.enabled = true;
item.physics.applyForce(new Vector3(0, 0, 500));
// Applies constant directional force to this item each frame.
```

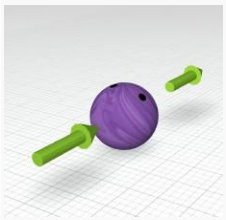


Push an object **towards** another **object**

```
const distance = item1.center.sub(item2.center);
// Subtracts item2's position from item1's position to calculate the direction

item2.physics.applyImpulseLocal(Vector3.zero, distance);
// Pushes item2 towards item1 by applying an impulse in the direction of the distance
```

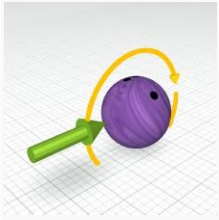
Push an object **towards** a certain **position**

```
let distance = new Vector3(10, -10, 0).sub(item.center);
// Subtracts the item's position from a specific position (Vector3(10, -10, 0))

item.physics.applyImpulseLocal(Vector3.zero, distance);
// Pushes the item towards the position, by applying an impulse in that direction
```



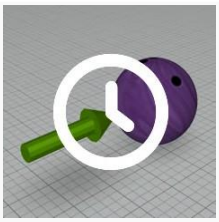
Push an object **in** a certain **direction** at a certain speed

```
item.physics.enabled = true;
// Enables physics for the item

Physics.physicsSpeed = 0.5;
// Sets the overall physics speed to 0.5, making the physics simulation slower

item.physics.applyImpulseLocal(Vector3.zero, new Vector3(0, 100, 0));
// Applies an impulse upwards along the Y-axis to the item
```



Make a certain object **spin** in a certain direction

```
item.physics.angularVelocity = new Vector3(0, 0, 2);
// Sets the angular velocity of the item to spin along the Z-axis

item.physics.restrictRotationAxis({ x: true, y: true, z: false });
// Restricts rotation, allowing the object to spin only around the Z-axis

item.physics.friction = 0;
// Removes friction, allowing the object to spin without slowing down
```



Define a **duration for physics TypeScript code** to get executed

```
Time.schedule(() => {
    Physics.paused = true;
    // Pauses the physics simulation. If 'true', physics will stop simulating
}, 2);
// Executes the physics block after 2 seconds
```

Advanced

PRO

Set the local or global speed for a certain physics object

```
const item = Scene.getItem("itemID");
item.physics.velocity = new Vector3(0, 10, 0);
// Sets the velocity of the item to (0, 10, 0) along the Y-axis
```

Set the local or global angular speed for a certain physics object

```
const item = Scene.getItem("itemID");
item.physics.angularVelocity = new Vector3(0, 0, 20);
// Sets the angular velocity of the item to rotate around the Z-axis at 20 units
```

Properties

PRO

Restrict the **movement** of an object on axes

```
const item = Scene.getItem("itemID");
item.physics.restrictRotationAxis({ x: true, y: false, z: true });
// Restricts rotation on the X and Z axes, but allows rotation on the Y axis
```

Define whether an object is **static** or not

```
const item = Scene.getItem("itemID");
item.physics.static = true;
// Makes the item static, preventing it from moving or being affected by forces
```

Define whether an object can **collide** with other objects

```
const item1 = Scene.getItem("item1ID");
const item2 = Scene.getItem("item2ID");

item2.physics.addToCollisionFilter(item1);
// Adds item2 to the collision filter of item1, allowing item2 to pass through item1
```

Define the **friction** level of a certain object

```
let item = Scene.getItem("itemID");
item.physics.friction = 0.5;
// Value of 0 resembles ice (very slippery)
// Value of 1 resembles a surface with high friction, e.g., car tyre
```

Define the **weight** (mass) of a certain object

```
let item = Scene.getItem("itemID");
item.physics.mass = 500;
// Defines the mass of the item.
// Items with higher mass more easily push away items with lower mass.
```

Define the **bounciness** level of a certain object

```
let item = Scene.getItem("itemID");
item.physics.restitution = 1;
// Defines the bounciness of the item
// Restitution of 0 does not cause the item to bounce
// Restitution of 1 causes the item to bounce without any loss of energy
```

Define the **gravity** level in your scene

```
Physics.gravityAcceleration = 0;
// Defines the gravity acceleration of the physics simulation.
// A value of 0 results in no gravity, so items will float.
// Negative values cause items to fall 'up'.
```

Change the **speed** at which **physics** happen (1 = standard speed)

```
Physics.physicsSpeed = 5;
// Defines the speed of the physics simulation.
// Low values (below 1) allow you to create slow-motion simulations.
// Higher values (above 1) speed up the simulation.
```

Functions

Simple

PRO

Stop a **function** from executing further

```
function myFunction() {
    // Some code
    if (someCondition) {
        return; // Stops the function and exits
    }
    // More code that won't execute if the condition is true
}
```

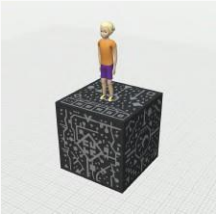
Return a certain **value from a function**

```
function myFunction(): number {
    return 1; // Returns the value 1
}

let result = myFunction();
Debug.log(result); // Logs the returned value (1) to the console
```

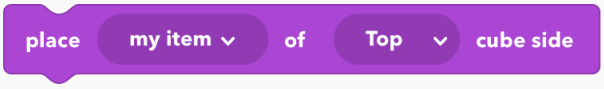
MERGE Cube

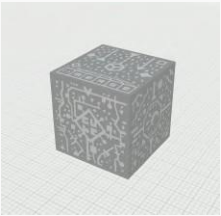
Actions – no Typescript code for Merge cube

PRO

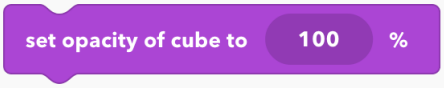


Place a certain object **on** a certain **side** of the MERGE Cube

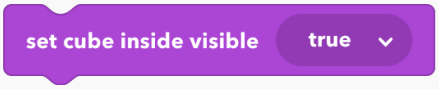




Change the **opacity** level of the MERGE Cube



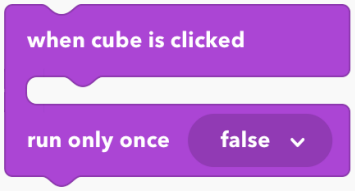
Make the **inside** of the MERGE Cube **visible or invisible**

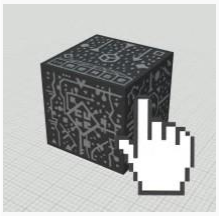


Events - no Typescript code for Merge cube

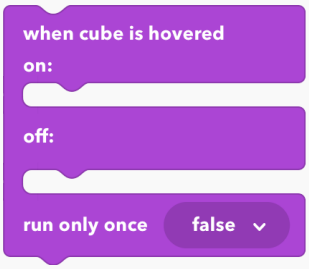
PRO

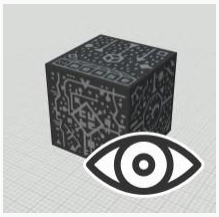
Make an action **happen when** the MERGE Cube **is clicked** and define whether it should happen only once or more



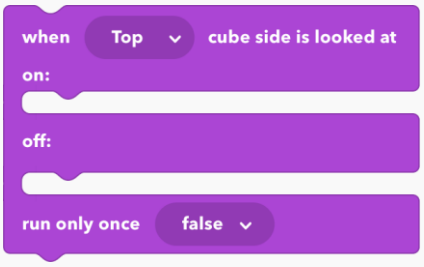


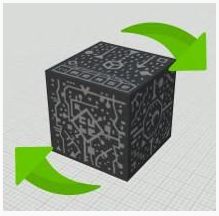
Make an action **happen when** the MERGE Cube **is hovered** and define whether it should happen only once or every time it is hovered



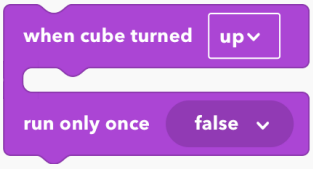


Make an action **happen** when **looking** at a certain **side** of the MERGE Cube and define whether it should happen only once or every time it is looked





Make an action **happen** when the MERGE Cube is **turned** a certain way and define whether it should happen only once or every time the MERGE Cube is turned



Stop a group of MERGE Cube **actions from happening**

Simple - no Typescript code for Merge cube

PRO

Use a certain **side** of the MERGE Cube

Use the **visible side** of the MERGE Cube (the one the camera is currently looking at)

Notes *

Storing CoSpace property values between TS scripts in ONE scene

storedLivesTS

```
// Define and export the class storedLives
export class storedLives {
    lives: number;

    constructor(lives: number) {
        this.lives = lives;
    }
}

// Set the "storedLives" property in the Scene
Scene.setProperty("storedLives", "3");
```

With this code, you can set CoSpace property, export the *stored Lives* class and manage the lives within your CoSpace project.

mainTS

```
import { storedLives } from './storedLivesTS';

// Assuming the lives are stored in Scene's property
let storedLivesValue = Scene.getProperty("storedLives")      ;
let livesInstance = new storedLives(Number(storedLivesValue));

Debug.log(`Lives: ${livesInstance.lives}`);  // Logs the value (e.g., 3)
```

With this code you can get the CoSpace property, by importing the *stored Lives* class and further manage the lives within your CoSpace project.