WORKSHOP 4 : Loop Blocks

🎯 Learning Goals

By the end of this workshop, you will be able to:

1. Understand what a loop is and why it is useful in programming.
2. Use the Repeat N times block to run instructions a fixed number of times.
3. Use the Repeat While / Until block to loop based on a condition.
4. Use the Count with block to loop with an automatic counter variable.
5. Use the For Each item in list block to iterate over a list of values.
6. Use the Break out of loop block to exit a loop early when needed.
7. Apply all loop types to real Niryo robot programs.

📦 Section 1 — What is a Loop?

The Everyday Analogy

Imagine you have to fill 10 glasses of water. You could write down "fill glass" ten separate times — or you could simply write "repeat 10 times: fill glass". That second version is exactly what a loop does in programming.

💡 Definition

A loop is a programming instruction that makes the computer (or robot!) repeat a block of code
multiple times, without you having to copy and paste the same instructions over and over again.

Loops are one of the most powerful tools in programming. They save time, make your code shorter, and make it much easier to change. Instead of updating 10 identical blocks, you update one loop.

Why Use Loops in Robotics?

Think about a robot that has to pick up 5 objects from a conveyor belt and place them one by one. Without a loop, you would need 5 identical "pick and place" sequences written out separately. What if the number changes to 8? You'd have to rewrite everything.

With a loop, you write the pick-and-place sequence once, and tell the robot to repeat it as many times as needed. This is exactly what we will explore in this workshop!

🗂️ Section 2 — The Loop Blocks in NiryoStudio

In NiryoStudio, all loop blocks are found in the Loops section of the block palette on the left side. There are five loop-related blocks available:

Block	Description
repeat N times	Runs the loop body exactly N times. The simplest loop — no condition needed.
repeat while / until	Repeats as long as a condition is true (while), or until a condition becomes true (until).
count with i from ... to ... by ...	Loops with a counter variable i. You control the start, end, and step size.
for each item in list	Loops over every item in a list, one by one. The variable takes each value in turn.
break out of loop	Immediately exits the current loop, no matter how many repetitions are left.

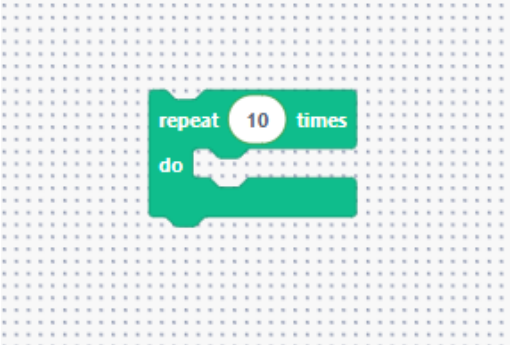
🔁 Section 3 — Repeat N Times

What It Does

The simplest loop block. You set a number N, and everything inside the loop runs exactly N times. No conditions, no variables — just a fixed count.

> 💡 **Definition**
> repeat [N] times
> do [instructions]
>
> The instructions inside the 'do' slot are executed N times in sequence, then the program moves on.

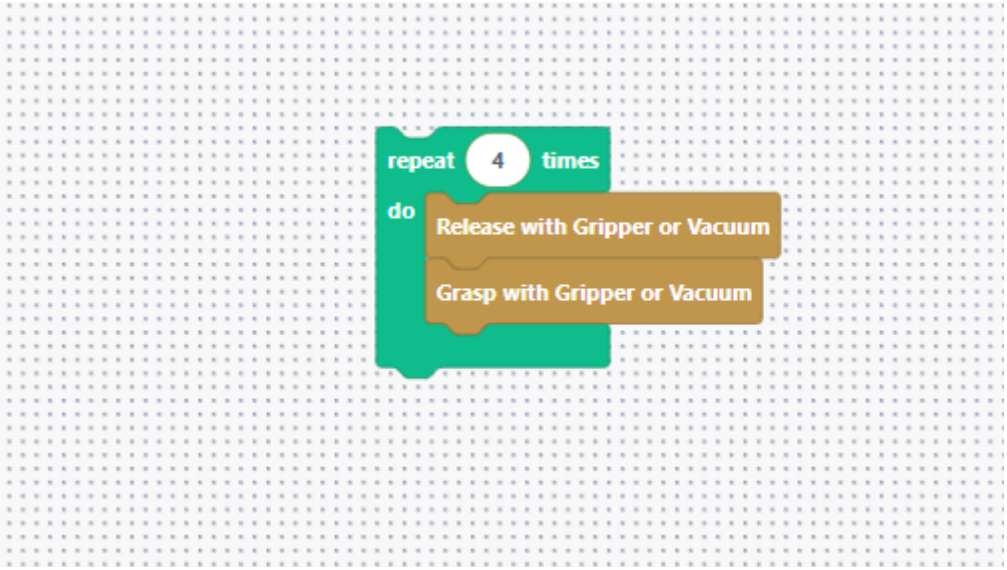
The Block



The 'repeat N times' block — here set to repeat 10 times

Real Example: Gripper Testing

In the example below, the robot alternates between releasing and grasping with its gripper or vacuum tool, repeating this sequence 4 times. This is useful to test whether the gripper is responding correctly, without writing 8 separate blocks.



repeat 4 times — alternating Release and Grasp with Gripper or Vacuum

🌍 **Real Example Walkthrough**

The loop repeats 4 times. Each iteration runs two actions:

- Release with Gripper or Vacuum — opens the gripper/releases the vacuum
- Grasp with Gripper or Vacuum — closes the gripper/activates the vacuum

This gives you a quick way to test or cycle the tool 4 times with a single compact block.

⚙️ **How to Use**

1. Drag the 'repeat N times' block onto the workspace.
2. Click the number and type your desired repeat count.
3. Drag the instruction blocks you want to repeat into the 'do' slot.

🔄 Section 4 — Repeat While / Until

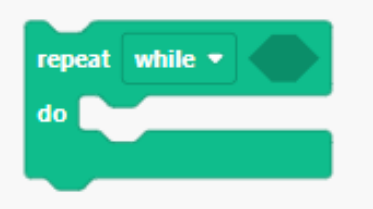
What It Does

This loop doesn't use a fixed count — instead, it keeps running as long as a condition is true (while mode) or until a condition becomes true (until mode). It is ideal when you don't know in advance how many times you need to loop.

💡 **Definition**
repeat [while ▼] [condition]
do [instructions]

WHILE mode: Runs the loop body as long as the condition is true. Stops when the condition becomes false.
UNTIL mode: Runs the loop body as long as the condition is false. Stops when the condition becomes true.

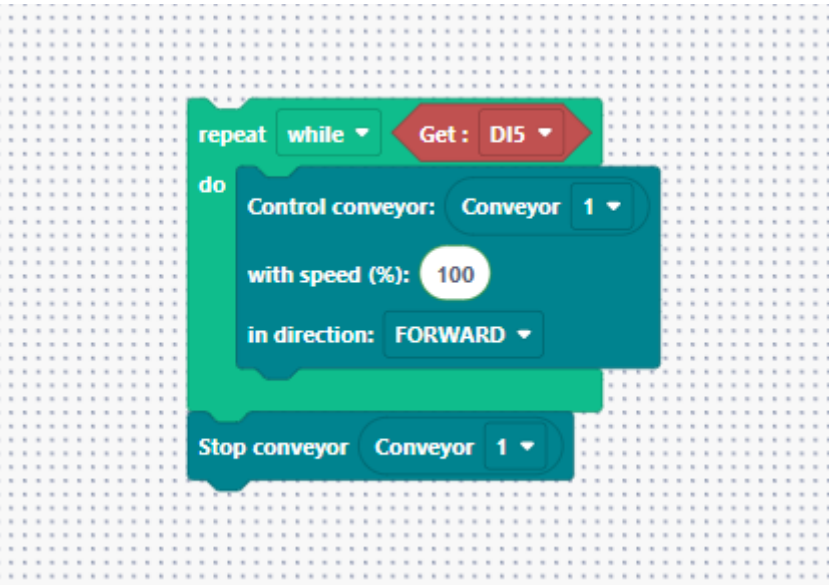
The Block



The 'repeat while / until' block — the dropdown switches between while and until

Real Example 1: Conveyor Controlled by a Sensor

In the example below, the conveyor belt runs continuously while a digital input sensor (DI5) is active. The moment the sensor changes state, the loop exits and the conveyor stops.



repeat while Get: DI5 — conveyor runs at 100% FORWARD as long as sensor DI5 is active

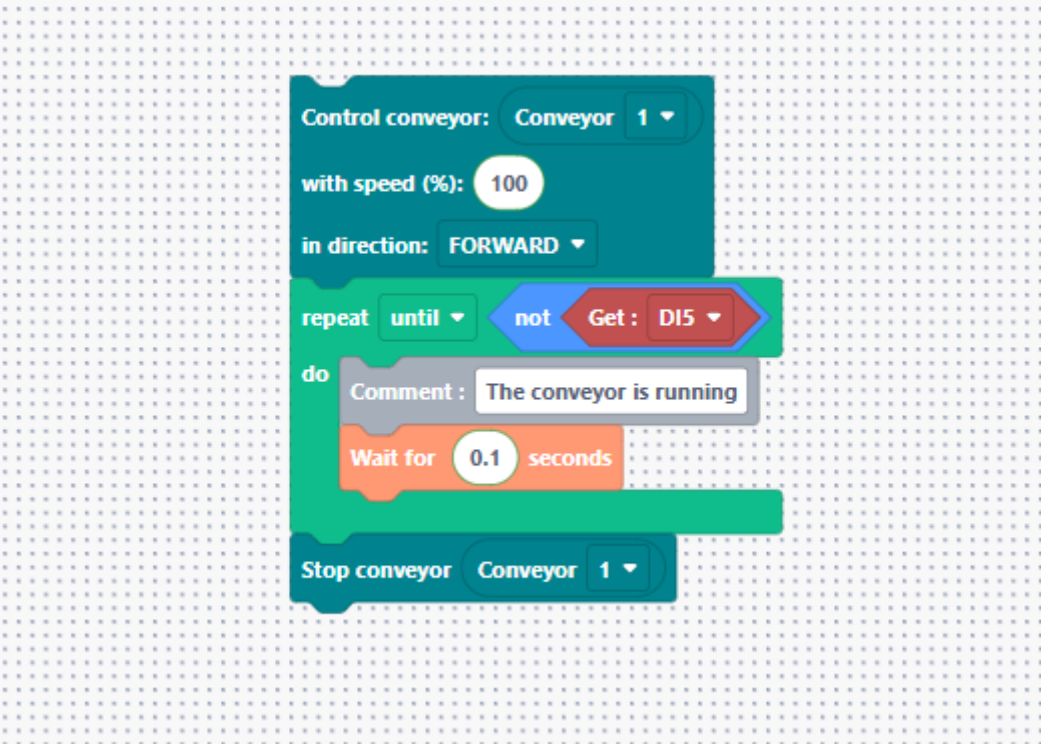
🌍 **Real Example 1 Walkthrough**
• 'Get: DI5' reads the current value of digital input 5 (a sensor).
• While DI5 is active (true), the loop body runs:
→ Control conveyor: Conveyor 1 at 100% FORWARD

• When DI5 becomes inactive (false), the loop exits.
• 'Stop conveyor Conveyor 1' runs immediately after — the belt stops.

This is much better than using a timer! The conveyor stops exactly when the sensor says so.

Real Example 2: Repeat Until a Condition is Met

In this variant, the 'until' mode is combined with 'not Get: DI5', so the loop keeps going until the sensor DI5 is no longer active. Inside, a comment reminds us that the conveyor is running, and a small wait of 0.1 seconds is added.



repeat until (not Get: DI5) — conveys with a comment and a 0.1s wait each iteration

🌍 Real Example 2 Walkthrough

• The condition is: 'not Get: DI5' → this means 'until DI5 is OFF'.
• The loop runs as long as DI5 is still ON (the sensor is still detecting something).
• Inside the loop:
  → A comment 'The conveyor is running' is displayed (useful for debugging)
  → Wait for 0.1 seconds — avoids overloading the processor with rapid checks
• Once DI5 turns OFF, the loop exits and the conveyor stops.

⚠️ Caution — Infinite Loops

If the condition never becomes false (while mode) or never becomes true (until mode), the loop will run forever — this is called an infinite loop! Always make sure your

condition can eventually be triggered, or use a 'break out of loop' block as a safety exit.

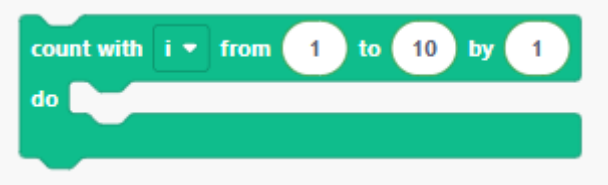
🔢 Section 5 — Count With (For Loop)

What It Does

This loop block is a more powerful version of 'repeat N times'. It introduces a counter variable (usually called i) that automatically increases or decreases with each iteration. You can use the counter inside the loop — for example, to change a robot's speed progressively.

> 💡 **Definition**
> count with [i ▼] from [start] to [end] by [step]
> do [instructions]
>
> The variable i starts at 'start', increases by 'step' each iteration, and the loop stops when i goes
> beyond 'end'. You can use the i variable block inside the loop body.

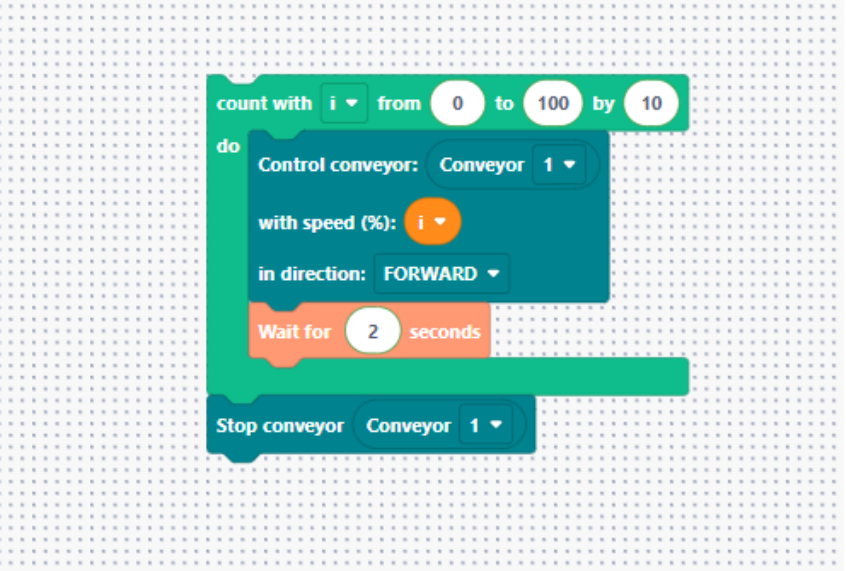
The Block



The 'count with i from 1 to 10 by 1' block — default values shown

Real Example: Ramping Up Conveyor Speed

In the example below, the conveyor speed increases progressively. The counter i starts at 0, goes up to 100, in steps of 10. Each iteration, the conveyor runs at speed i%, then waits 2 seconds before increasing again.



count with i from 0 to 100 by 10 — conveyor speed set to i% each iteration, 2s pause

🌍 Real Example Walkthrough

Iteration 1: i = 0 → conveyor speed = 0% → wait 2s
Iteration 2: i = 10 → conveyor speed = 10% → wait 2s
Iteration 3: i = 20 → conveyor speed = 20% → wait 2s
... (and so on until i = 100)
Iteration 11: i = 100 → conveyor speed = 100% → wait 2s
Loop ends. Stop conveyor Conveyor 1.

This creates a smooth acceleration effect — very useful in industrial robotics!

🧠 Key Insight

The variable i is automatically updated each iteration — you never need a 'change i by 10' block.
The loop block handles the counting for you. You just focus on what to do with each value of i.

⚙️ How to Use

1. Drag the 'count with i' block onto the workspace.
2. Set the 'from' value (starting value of i).
3. Set the 'to' value (the loop stops before exceeding this).
4. Set the 'by' value (how much i increases each time; use a negative value to count DOWN).
5. Drag the i variable block (orange oval) into any value slot inside the loop body.

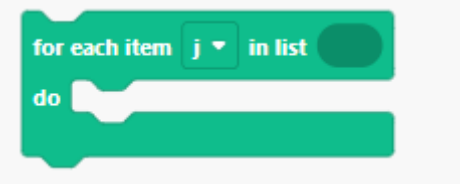
📋 Section 6 — For Each Item in List

What It Does

This loop block iterates over every item in a list. Instead of specifying a range of numbers, you provide a list of values — the loop variable automatically takes each value in turn, one per iteration. This is ideal when you have a specific set of values to apply (e.g., precise speed steps, a list of positions).

> 💡 **Definition**
> for each item [i ▼] in list [list]
> do [instructions]
>
> The variable i takes the value of each item in the list, one by one. When the last item is reached, the loop ends. You can use the i variable block inside the loop body.

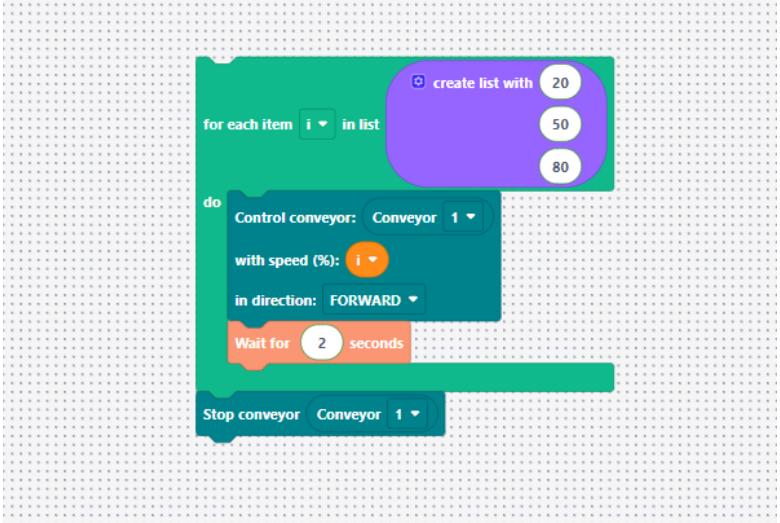
The Block



The 'for each item j in list' block — the list is created separately and plugged in

Real Example: Specific Conveyor Speed Sequence

In the example below, the conveyor runs at three specific speed values: 20%, 50%, then 80%. These values are stored in a list. The loop picks each one automatically — no manual repetition needed.



for each item i in list [20, 50, 80] — conveyor runs at each speed for 2 seconds

🌍 **Real Example Walkthrough**
Iteration 1: i = 20 → Control conveyor: speed = 20% FORWARD → Wait 2s
Iteration 2: i = 50 → Control conveyor: speed = 50% FORWARD → Wait 2s
Iteration 3: i = 80 → Control conveyor: speed = 80% FORWARD → Wait 2s
Loop ends. Stop conveyor Conveyor 1.

This is different from the 'count with' loop: the speed values are not evenly spaced.
You have full control over exactly which values are used, in exactly which order.

🧠 **For Each vs. Count With**
Use 'count with' when your values follow a regular pattern (e.g., 0, 10, 20, 30...).
Use 'for each' when your values are irregular or hand-picked (e.g., 20, 50, 80).
Both loop types use a variable that you can reference inside the loop body.

🚪 Section 7 — Break Out of Loop

What It Does

Sometimes you need to exit a loop before it would naturally end — for example, if an error is detected, a safety condition is triggered, or you've found what you were looking for. The 'break out of loop' block immediately stops the loop and jumps to the next block after it.

💡 **Definition**
break out ▼ of loop

Immediately exits the innermost loop that contains this block. The 'break out' dropdown can also be set to 'continue with next iteration', which skips the rest of the current iteration and goes directly to the next one.

The Block

The 'break out of loop' block — can also be set to 'continue with next iteration'

Typical Use Case

The break block is typically placed inside an 'if' condition inside the loop body. When the condition is met, the loop exits immediately.

break out of loop	continue with next iteration
Exits the loop entirely. No more iterations will run. Use when: an error is detected, a safety limit is hit, or the task is done early.	Skips the rest of the current iteration and jumps to the next one. Use when: a specific item should be skipped but the loop should continue.

> ⚠️ **Important**
> The 'break out of loop' block only exits the innermost loop. If you have nested loops (a loop inside a loop), break only exits the one it is directly inside.

📋 Quick Reference Cheat Sheet

Block	What it does	Typical Use Case
repeat [N] times	Runs the loop body exactly N times. No condition, no counter.	Test gripper N times; pick N objects
repeat while [cond]	Loops as long as cond is TRUE. Stops when cond becomes FALSE.	Run conveyor while sensor is active
repeat until [cond]	Loops as long as cond is FALSE. Stops when cond becomes TRUE.	Wait until object is detected
count with i from A to B by S	Counter i goes from A to B in steps of S. Use i inside the loop.	Gradually increase conveyor speed
for each item i in list	i takes each value in the list, one per iteration.	Apply a set of specific speed values
break out of loop	Immediately exits the current loop.	Emergency stop / early exit on error
continue next iteration	Skips the rest of this iteration; goes to next.	Skip one item in a sequence