TOKY

TOKYMAKER TUTORIAL

BLOCKS INDEX

INDEX OVERVIEW

create.tokylabs.com has a large number of blocks. They are divided into the following categories:

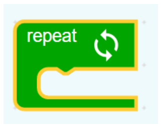
- **Control**
- **Input**
- Output
- Display
- Number
- **Logic**
- **Variables**
- Text

Control

CONTROL

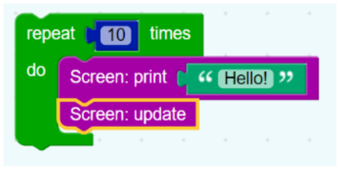
The **Control** category holds blocks that control whether other blocks placed in their **body** are run. (For example, in the below "repeat" block, the body contains the "print" block and its input.) There are two types of control blocks: IfElse and these, which control how many times the body is run and, in some cases, the value of a variable used within the body. These structures are called **loops** since the body is repeated (possibly) multiple times, reminiscent of a rope containing loops. Each pass through the loop is called an **iteration**. For more information, see https://en.wikipedia.org/wiki/Control_flow#Loops.

REPEAT FOREVER



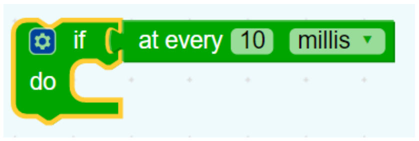
The simplest "repeat" block runs the code in its body in an endless loop. Will execute the inner instructions continuously, once and again.

REPEAT X TIMES



This version of the “repeat” block runs the code in its body the specified number of times. For example, the following block will print "Hello!" ten times.

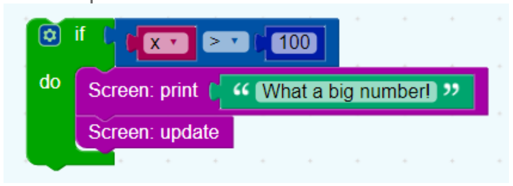
AT EVERY



”at every” block, together with the conditioner if/do, runs the code in each period of time specified in milliseconds or minutes

IF BLOCKS

The simplest conditional statement is an **if** block, as shown:

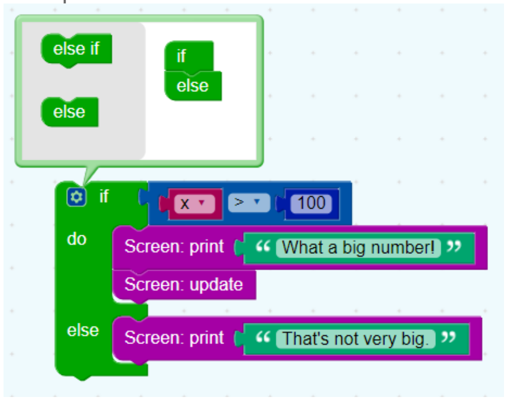


```
if x > 100
do  Screen: print " What a big number! "
    Screen: update
```

When run, this will compare the value of the variable **x** to 100. If it is larger, "What a big number!" will be printed. Otherwise, nothing happens.

IF-ELSE BLOCKS

It is also possible to specify that something should happen if the condition is *not* true, as shown in this example:



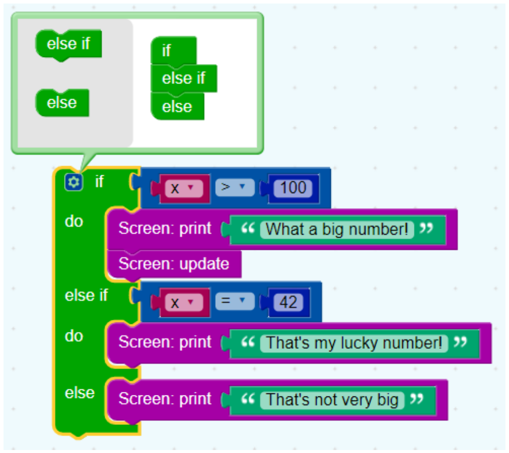
```
else if        if
else           else

if x > 100
do    Screen: print " What a big number! "
      Screen: update
else  Screen: print " That's not very big. "
```

As with the previous block, "What a big number!" will be printed if **x** > 100; otherwise, "That's not very big." will be printed.

An **if** block may have zero or one **else** sections but not more than one.

IF-ELSE BLOCKS

It is also possible to test multiple conditions with a single **if** block by adding **else if** clauses:



```
else if        if
               else if
else           else

if x > 100
do       Screen: print " What a big number! "
         Screen: update
else if x = 42
do       Screen: print " That's my lucky number! "
else     Screen: print " That's not very big "
```

The block first checks if **x** > 100, printing "What a big number!" if it is. If it is not, it goes on to check if **x** = 42. If so, it prints "That's my lucky number." Otherwise, nothing happens.

An **if** block may have any number of **else if** sections. Conditions are evaluated top to bottom until one is satisfied, or until no more conditions are left.

BLOCK MODIFICATION

To add **else if** and **else** clauses, the user needs to click on the gear icon, which opens a new window.

The user can then drag **else if** and **else** clauses into the **if** block, as well as reordering and removing them. When finished, the user should click on the minus sign, which closes the window, as shown in the previous blocks.

Logic

LOGICe

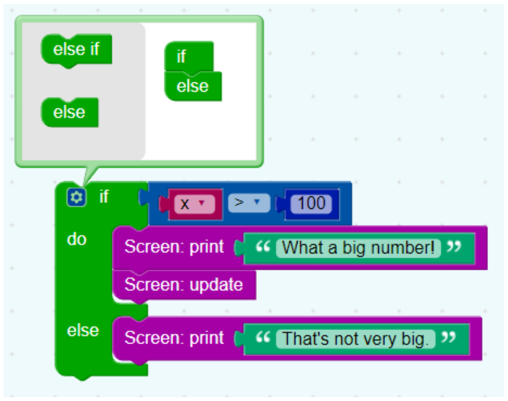
Boolean algebra is a mathematical system that has two values:

- **true**
- **false**

Boolean values (also called *conditions*) are used in these control block, which contain examples:

- conditional blocks
- repeat blocks

One of the many examples from those pages is:



If the value of the variable **x** is greater than 100, the condition is **true**, and the text "What a big number!" is printed. If the value of **x** is not greater than 100, the condition is **false**, and "That's not very big." is printed.

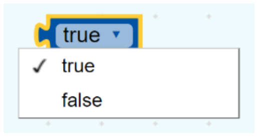
Boolean values can also be stored in variables and passed to procedures, the same as number, text, and list values.

BLOCKS

If a block expects a Boolean value as an input, it usually interprets an absent input as **false**. An example is provided below. Non-Boolean values cannot be directly plugged in where Boolean values are expected, although it is possible (but inadvisable) to store a non-Boolean value in a variable, then plug that into the input. Neither of these practices are recommended, and their behaviour could change in future versions.

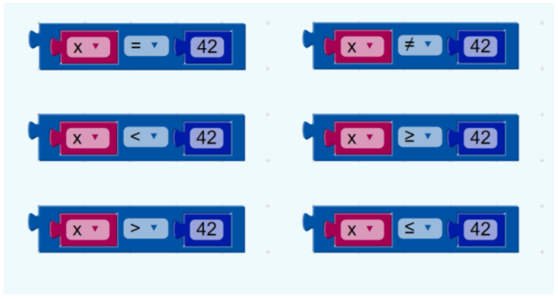
VALUES

A single block, with a dropdown specifying either **true** or **false**, can be used to get a boolean value:



COMPARISONS

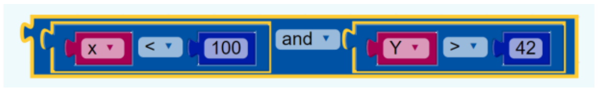
There are six comparison operators. Each takes two inputs (normally numbers) and returns true or false depending on how the inputs compare with each other.



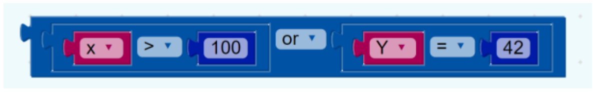
The six operators are: equals, not equals, less than, less than or equal, greater than, greater than or equal.

LOGICAL OPERATIONS

The **and** block will return **true** only if both of its two inputs are also true.

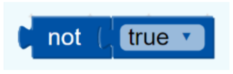


The **or** block will return **true** if either of its two inputs are true.

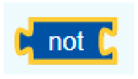


NOT

The **not** block converts its Boolean input into its opposite. For example, the result of:



is false.
As mentioned above, if no input is provided, a value of **true** is assumed, so the following block produces the value **false**:



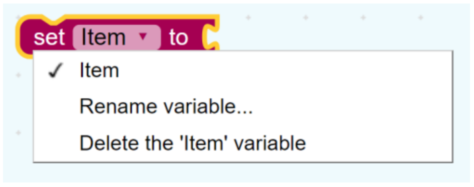
Leaving an input empty is not recommended, however.

We use the term *variable* the same as it is used in mathematics and in other programming languages: a named value that can be changed (varies). Variables can be created in several different ways.

- Every count with and for each block uses a variable and defines its values. These values can only be used within the block. A traditional computer science term for these are loop variables.
- User-defined functions (also known as "procedures") can define inputs, which creates variables that can be used only within the function. These are traditionally called "parameters" or "arguments".
- Users may create variables at any time through the "set" block. These are traditionally called "global variables".

DROPDOWN MENU

Clicking on a variable's dropdown symbol (triangle) gives the following menu:

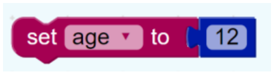


The menu provides the following options.

- the names of all variables defined in the program.
- "Rename variable...", which changes the name of this variable wherever it appears in the program. Selecting this opens a small window that prompts the user for the new name with the text: "Rename all %1 variables to:", where %1 is replaced by the old name (here "item").
- "New variable...", which enables the user to enter a new name for the variable, without replacing or changing variables with the old name (here "item"). Selecting this opens a small window that prompts the user for the new name with the text "New variable name:".

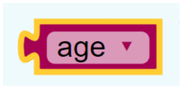
SET

The **set** block assigns a value to a variable, creating the variable if it doesn't already exist. For example, this sets the value of the variable named "age" to 12.



GET

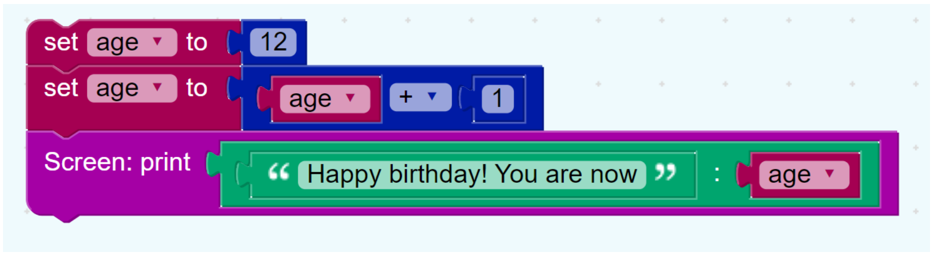
The **get** block provides the value stored in a variable, without changing it.



It is possible, but a bad idea, to write a program in which a **get** appears without a corresponding **set**

EXAMPLE

Consider the following example code:



```
set age to 12
set age to age + 1
Screen: print "Happy birthday! You are now" : age
```

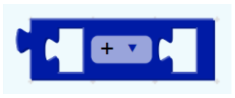
The first row of blocks creates a variable named "age" and sets its initial value to the number 12. The second row of blocks gets the value 12, adds 1 to it, and stores the sum (13) into the variable. The final row displays the message: "Happy birthday! You are now 13"

Numbers are mathematical values that play a huge role in programming. In Blockly, they are used for optimizing and making algorithms function properly. Without numbers, addition, multiplication, etc. would not be possible.

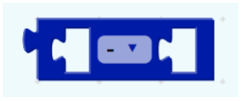
MATHEMATICAL FUNCTIONS

The following table lists various mathematical functions and how to perform them in Scratch; it is not inclusive to all functions:

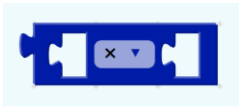
Addition:



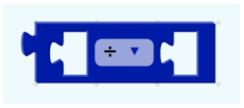
Subtraction



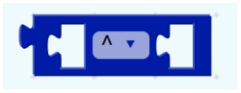
Multiplication



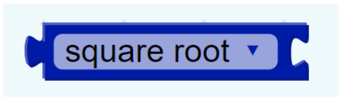
Division



Exponential

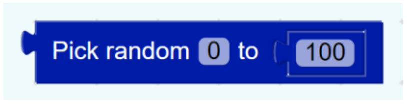


Square root



RANDOM NUMBER

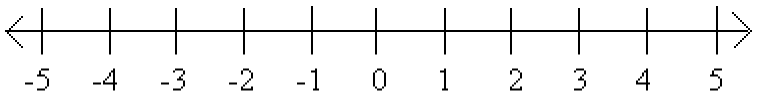
This block is conceived to give you the possibility of having a random number.



You can choose the range of variability by changing the second number. You could include a variable too.

NUMBER LINE

A number line can be used to represent integers and the values in between them. The following image depicts a number line:

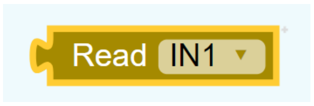


INPUT

Inputs are blocks that use the information from the environment to be applied in our code. To capture this information we need **sensors.** Tokymaker has a large variety of sensors to be use in your projects. If your want to know more about what a sensor is, check this useful video: *https://www.youtube.com/watch?v=v25PCV_IJCw&t=3s*

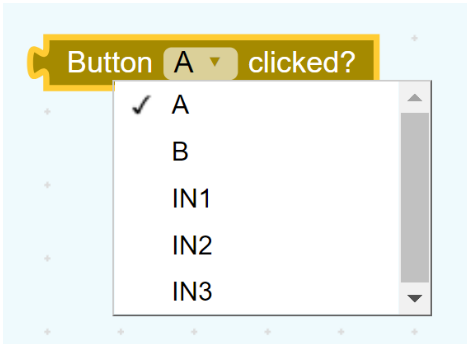
READ IN

This is probably one of the most important blocks of create.tokylabs.com. This can act as a box to store the information that the sensor is detecting. You can select the port where this sensor is connected: IN1, IN2, or IN3. The value goes from 0 to 100. For example, if you connect a light sensor in the Input 1, this block will store a 0 in total darkness and a 100 when facing a powerful sun.

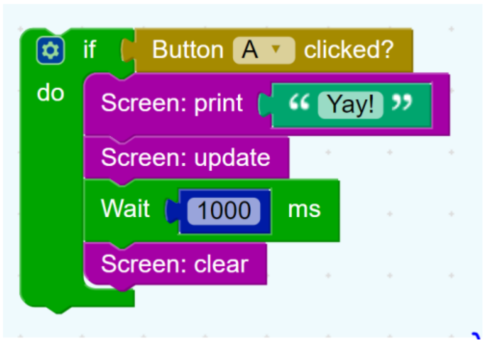


BUTTON CLICKED

You could connect a button to any of the Inputs, butTokymaker has already two physical buttons embedded. This block useful when you want to do anything in response to a click of the button. The Clicked and pressed block is a Sensing block and a Boolean block. If the user is clicking the selected object, the block returns *true*; if it is not, it returns *false*.



From the drop down list, you can choose what buttont to listen to. Let's see a simple example:

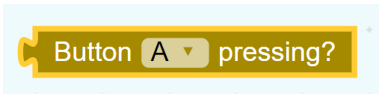


```
if Button A clicked?
do  Screen: print "Yay!"
    Screen: update
    Wait 1000 ms
    Screen: clear
```

In this code fraqment, the screen will show a funny message once the button is pressed. The message will last for one second and then dissapear.

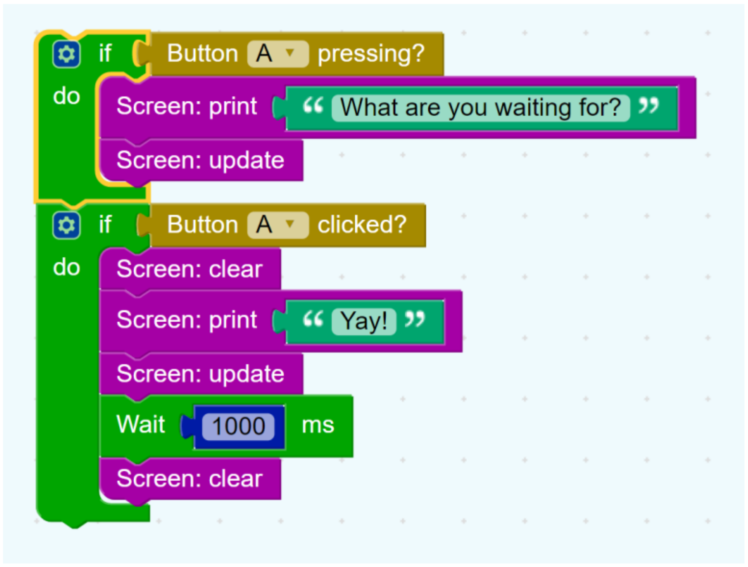
BUTTON PRESSING

Alternatively, you might want to do something when the user is pressing but not yet clicking. on. The pressing block is a Sensing block and a Boolean block. If the user is clicking the selected object, the block returns *true*; if it is not, it returns *false*.



```
Button A ▾ pressing?
```

If we want to use it, the example could be something like:

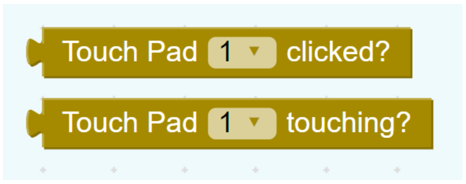


```
if Button A ▾ pressing?
do
    Screen: print "What are you waiting for?"
    Screen: update
if Button A ▾ clicked?
do
    Screen: clear
    Screen: print "Yay!"
    Screen: update
    Wait 1000 ms
    Screen: clear
```

Now, a waiting message Will be shown in the screen until the user finally clicks.

TOUCH PAD

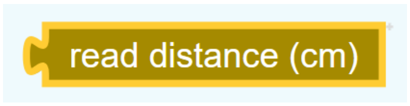
Tokymaker also counts with three touch pads that act as a button in the same way as Buttons A and B.



```
Touch Pad 1 ▾ clicked?
Touch Pad 1 ▾ touching?
```

READ DISTANCE

Read distance is a specific block for an Ultrasonic distance detector. This block stores the value of the distance in centimeters, ranging from 0 to 200 cm (2 meters). As this block can give distances between objects, it is very useful in projects that require a great deal of careful sensing and movement.



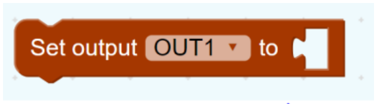
```
read distance (cm)
```

Since this sensor is getting an information from the physical environment, it’s block has to be an input.

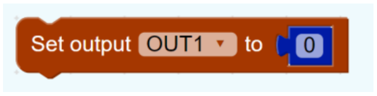
Outputs are the actuators. They create an action based on your code. It is the way to interactuate with the world!

SET OUTPUT

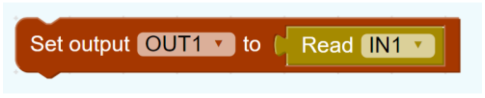
This block creates an action on OUT1. If you connect one of our actuators (motors, lights, vibrators, relays, etc) it will react based on the numeric block you include in the empty space. It can be any number (constant or variable) that goes from 0 to 100.



For example, if we connect a LED module in the OUT1 and we want it off, It should be like that:



But if we want to vary the light intensity (from 0% to 100%) we can connect a potentiometer module (rotation sensor) in IN1 and set put the next code:

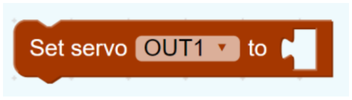


This way we can adjust intensity manually by modifying the Input 1.

Behind the block, there is a Pulse With Modulation (PWM) code that creates an average voltage value that goes from 0 Volts (Duty cycle 0%) to 3.7 Volts (Duty cycle 100%). Follow the next link to know more about PWM. https://en.wikipedia.org/wiki/Pulse-width_modulation

SET SERVO

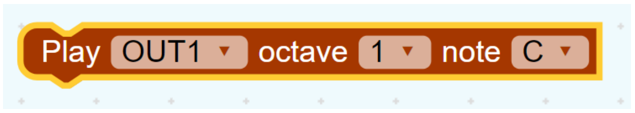
A servo motor is different from a continuous rotation motor. It stays in a fixed angle based on the value we include in the empty space. You could connect a button to any of the Inputs, butTokymaker has already two physical buttons embedded. This block useful when you want to do anything in response to a click of the button.



The servo itself can rotate around 180º, but we reescaled the angles to adapt it to the input percentage values (from 0% to 100%).
Then, we have to put a number 0 if we want the servo to stay at 0º and we have to put a number100 if we want it to stay at 180º. Alternatively, we can also put variables, or Inputs, etc.
Find out more at: https://en.wikipedia.org/wiki/Servo_(radio_control)

PLAY TONE

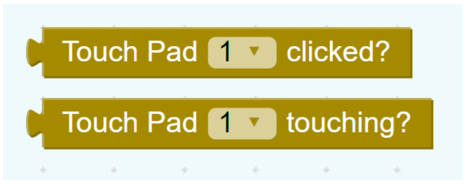
Tokymaker can include a large variety of actuators. One of them is a Speaker to produce notes and tones. The operation is fairly simple, we just need to select the Output where we connected the module and select from the two dropdown menus the octave and the note.



You need top ut a wait module.f wait.

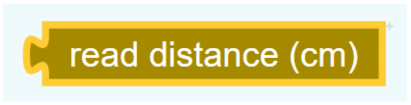
TOUCH PAD

Tokymaker also counts with three touch pads that act as a button in the same way as Buttons A and B.



READ DISTANCE

Read distance is a specific block for an Ultrasonic distance detector. This block stores the value of the distance in centimeters, ranging from 0 to 200 cm (2 meters).



Since this sensor is getting an information from the physical environment, it’s block has to be an input.