Block description Bionics4Education

Contents

- Home
  - Action
  - Sensors
  - Control
  - Logic
  - Math
  - Text
  - Lists [Expert-blocks]
  - Colours
  - Variables
  - Functions [Expert-blocks]

»Program start «

Each program starts with the »Program start block«, which cannot be deleted. The first block to be used is connected directly to the »Sequence connection« on the program start block.

Settings:
- Create a new global variable.
- Remove the global variable.

If global variables have been created:
- Text, name of the variable.
- Type of the variable.
- Value, initial value of the variable.

Action

The category »Action« contains blocks that you can use to control the actions of your robot.

You can choose from the following blocks:
- show on Serial Monitor ...
- set servo motor ... to ° ...
- turn LED on ...

»show on Serial Monitor ... «

With this block you can transfer a text via the serial interface to your computer and read it out with a serial monitor. For this you can use the built-in serial monitor of the Open Roberta Connector.

Input values:
- String, text you want to transfer to your computer.

»set servo motor ... to ° ... «

With this block you can control the connected servo motor and adjust its position. The position is described as an angle, where 360° is a full rotation.

Options and input values:
- Option, choose the connected servo motor that you want to control.
- Number, the new angle for the servo motor.
»turn LED on ... «

With this block you can either switch a connected LED on or off.

Options:

- Option, choose which LED to control.
- Option, choose whether to turn the LED on or off.

Sensors

In the category »Sensors« there are blocks that you can use to control the sensors of your robot. With sensors your robot can perceive its environment. The following blocks are available:

- get value ms timer ...
- reset timer ...

»get value ms timer ...«

With this block you can query the value of one of the build-in timers of your robot. The first timer is automatically started at program start and all other timers can be started by the »reset timer ... « block.

Options:

- Option, choose the timer which value want to query.

Return value:

- Number, time since program start or last reset in milliseconds.

»reset timer ... «

With this block you can reset the value of one of the build-in timers of your robot. The first timer is automatically started at program start and all other timers can be started by this block.

Options:

- Option, choose the timer which you want to reset.

Control

The category "control" includes blocks with which the program sequence can be controlled. The category includes the following blocks:

- if do
- if do else
- repeat indefinitely
- repeat n times
- wait
- wait until
- repeat while/until [Expert-Block]
- for each item in list [Expert-Block]
- count with from to [Expert-Block]
- break out/continue with next iteration of loop [Expert-Block]
»if do«

With the block »if do« you can selectively trigger actions to be executed by your robot. The »if do« block therefore requires a logical value as an input parameter, the condition. Only if the condition of the »if« statement is true, the inserted block will be executed. In a nested »if do« block, if a further distinction was added, the first »if« condition is queried. If it is not fulfilled (condition = false), the second »else if« condition will be checked. Also this second condition requires a logical value as an input parameter.

The conditions may arbitrarily be expanded by clicking the "+" plus symbol. The "−" minus symbol reduces the block.

Settings and input values:
- Insert an additional condition.
- Delete the last condition.
- Boolean value, »true« or »false«.
- Blocks that will be executed

»if do else«

With the block »if do else« you can selectively trigger actions which are executed by your robot. The »if do then« block therefore requires a logical value as an input parameter. If the condition in »if« is true the inserted block will be executed, otherwise (condition is not fulfilled = false) the block connected to the »else« statement will be executed. In a nested »if do else« block with further distinctions added, the first »if« condition is queried. If it is not true, the second condition »else if« is checked. Also the second condition requires a logical value as an input parameter. Only when both conditions are not true, the block which is inserted at the »else« statement will be executed.

The conditions may arbitrarily be expanded by clicking the "+" plus symbol. The "−" minus symbol reduces the block.

Settings:
- Insert an additional if-do-else condition.
- Delete the last if-do-else condition.
- Boolean value, »true« or »false«.
- Do blocks that will be executed if the according condition evaluates to »true«.
- Else blocks that will be executed if the according condition evaluates to »false«.

»repeat indefinitely«

With the block »repeat indefinitely« you can endlessly run the blocks on your robot. All blocks which are within the »repeat indefinitely« block will be executed endlessly. The blocks are applied sequentially from top to bottom. Once the last block has been executed, the program repeats with the first block again. Therefore, this block is also called »loop«.

Input:
- Blocks to be repeated indefinitely.

»repeat n times«

With the block »repeat« you can run other block as many times as you like. All blocks in the »repeat« block will be executed as often as defined in the entry field. The blocks are applied sequentially from top to bottom. Once the last block has been executed, the program repeats with the first block again. Therefore, this block is also called »loop«.

Settings and input:
- Number that indicates how often the contained blocks will be repeated.
Blocks to be repeated as often as defined.

»wait«
With the block »wait« you can "pause" your program at the point where you inserted the »wait« block. Your program will then remain for the specified duration at this point. After the specified time the next block will be executed. For example you can display text in the screen of your robot for exactly the time you specified in the »wait« block.

Settings and input values:
- Number, waiting time in milliseconds.

»wait until ...«
With the block »wait until« you can "stop" your program at the point you insert the »wait until« block. Your program then waits until the condition is true. The »wait until« block you can extended by click on "+" symbol. Your program then waits until (at least) one of the condition of your »wait until« block is true.

Settings:
- Add a new condition.
- Delete the last condition.
- Boolean value, »true« or »false«.

»break out/continue with next iteration of loop« [Expert-block]
With the block »break out/continue with next iteration of loop« a loop can be terminated ahead of schedule. As soon as the block is entered within a sequence of blocks, all further blocks up to the end of the loop will be ignored.

Settings:
- Type of breaking behavior, »break out« or »continue with next iteration«.

»count with from to« [Expert-block]
With the block »count with from to« you can run other block as many times as you like. All blocks in the »count with from to« block will be executed as long as the counting is in progress. The blocks are applied sequentially from top to bottom. Once the last block has been executed, the program repeats with the first block again if the counting is in progress. The last parameter declares the step width for counting.

Settings:
- Variable, name of free choice; numbers of the counter will be delivered one after the other to the variable.
- Number, initial value of the counter.
- Number, final value of the counter. As the counter exceeds this value the loop ends.
- Number, defining the increment. The variable value will increase by this amount after every loop cycle.
- Blocks that will be executed in every loop cycle.
**for each item in list**  [**Expert-block**]

With the block «for each item in list» all list items will successively be bound to a variable. The variable can be used within the loop. With each loop cycle the next item of the list will be bound until all list elements have been processed.

Settings and input value:

- Type of list elements, «Number», «String», «Boolean», «Colour», «Connection»
- Variable, name of free choice; list elements will be delivered one after the other to the variable.
- List that contains elements of the desired type. If the list elements are not of the correct type then the list will not fit to the input slot.
- Blocks which will be executed as often as there are elements in the list.

**repeat while/until**  [**Expert-block**]

With the block «repeat while/until» you can run other block as many times as you like. All blocks in the «repeat while/until» block will be executed as long as the condition in the entry field is true. The blocks are applied sequentially from top to bottom. Once the last block has been executed, the program repeats with the first block again if the condition is still true. Therefore, this block is also called «conditional loop».

Settings and input values:

- Option, «while» or «until», defining the type of the conditional repetition.
- Boolean value, «true» or «false».
- Blocks that will be repeated while/until the condition evaluates to «true».

**Logic**

With the «logic» blocks you can "create" conditions. With this condition you can interrelate states, values, and events with each other.

The following «logic» blocks are available at NEPO:

- comparison
- and/or
- not [**Expert-block**]
- true/false
- null [**Expert-block**]
- test [**Expert-block**]

**comparison**

With the block «comparison» you can compare different parameters of the same type (number, color, logical value, text). This block can only be used in conjunction with another block which requires a logical value as an input parameter.

Settings and input values:

- Value for the left hand side.
- Comparison, select one of =, , <, , >.
- Value for the right hand side.

Return value:

- Boolean value, «true» or «false».

**and/or**

With the block «and/or» you can interrelate logical values with each other. The «and/or» block with the setting "and" will be true only if both logical parameters are "true". If the block has the setting «or» it is sufficient if one of the two parameters is "true", so that the «and/or» block will return true.
Settings:
- Boolean value on the left hand side.
- Boolean function, »and« or »or«.
- Boolean value on the right hand side.

Return value:
- Boolean value »true« or »false«.

**true/false**
With the block »true/false« you can return either the logical value »true« or »false« to another block.

*Settings:*
- Boolean value, make your choice.

*Return value:*
- Boolean value, »true« or »false«.

**not** [Expert-block]
Using the »not« lets you invert a logical value and pass this value to another block.

*Input value:*
- Boolean value, to be inverted.

*Return value:*
- Boolean value, »true« or »false«; result of inverting.

**test** [Expert-block]
Using the »test« block will perform a test and returns a value which depends on the test result.

*Input values:*
- Boolean value for the test; if no input value is given "true" will be assumed.
- Arbitrary value, evaluated by a block.
- Arbitrary value, evaluated by a block.

*Return value:*
- Value of arbitrary type.

**null** [Expert-block]
The block »null« is a place holder for an input value that is not yet specified. If for instance a new connection variable has been created and is not yet bound, the initial value of this connection will be set to »null«.

*Return value:*
- Boolean value, »null«.

---

**Math**
Using "Math" blocks offers calculations and parameter settings.

The following "Math" blocks are available at NEPO:

- parameter
- calculating
- mathematical function [Expert-Block]
- trigonometric function [Expert-Block]
- constant [Expert-Block]
- number property [Expert-Block]
- change by ... [Expert-Block]
- round [Expert-Block]
- list evaluation [Expert-Block]
- remainder of [Expert-Block]
- list of list [Expert-Block]
- constrain [Expert-Block]
- random integer [Expert-Block]
- random fraction [Expert-Block]

### »parameter«

With the block "parameter" you can send numbers to another block.

Settings and input values:

- Number

Return value:

- Number

### »calculating«

With the block "calculating" you can sum up, subtract, multiply, and divide numbers. This block can only be used in conjunction with another block which requires a number as an input parameter.

Settings and input values:

- Number, first number on the left you want to calculate with.
- Option, mathematical operator, choose one of +, -, ÷, ×.
- Number, second number on the right you want to calculate with.

Return value:

- Number, result of the calculation.
>mathematical function< [Expert-block]

With the block `mathematical function` some elementary mathematical functions may be calculated. Available functions are `square root`, `absolute`, `invert` (multiply by -1), `ln` (natural logarithm), `log10` (decadic logarithm), `e^` (exponential function), `10^` (base 10 exponent)

Settings and input value:

- Option, mathematical function, choose one.
- Number to apply the function to.

Return value:

- Number, result of the function application.

>trigonometric functions< [Expert-block]

With the block `trigonometric function` sine, cosine, tangent and their respective reverse functions can be calculated. Input values are expected in radian measure (see hint above).

Settings and input value:

- Option, trigonometric function to choose from.
- Number, in radian measure.

Return value:

- Number, result of the trigonometric calculation.

>constant< [Expert-block]

With the `constant` block some mathematical constant values are available: `π` (3,1415...), `e` (2,718...), `φ` (1,618...), `sqrt(2)` (1,414...), `sqrt(½)` (0,7071...), ... .

Settings:

- Option, mathematical constants, choose one.

Return value:

- Number, value of the selected mathematical constant. Infinity will return «infinity».

>number property< [Expert-block]

With the block `number property` you check whether a given number has a specific property: `even`, `odd`, `prime`, `whole`, `positive`, `negative`, `divisible by`.

Settings and input values:
- Number, the property of this input value will be checked.
- Option, number property, choose one.
- Number, evaluated from a block. The second input value is only required for the property » divisible by «.

Return value:
- Boolean value, »true« or »false«, depending on the selected property.

»change by ... « [Expert-block]
The block »change by ... « increments a numerical variable by a defined value.

Settings and input values:
- Variable name, which value is to be changed. Choose the variable to be changed.
- Number, given by a block.

»round« [Expert-block]
With the block »round« values may be rounded. Rounding will set the decimal places to 0. It depends on the value of the decimal places whether the block rounds up or down. You may also decide by settings to always round up or down.

Settings and input value:
- Type of rounding, choose a round mode.
- Number you want to round.

Return value:
- Number, rounded whole number, according to the round mode.

»list evaluation« [Expert-block]
With the block »list evaluation« you may analyse a list.

Modes for list evaluation:
- sum - addition of all list values
- min - smallest value in the list
- max - largest value in the list
- average - average of all list values
- median - median of all list values
- standard deviation - standard deviation of all list values
- random item - one list value randomly selected

Settings and input value:
- Mode of list evaluation, choose one mode.
- List of numbers

Return value:
- Number, calculated according to the selected list evaluation mode.

»remainder of« [Expert-block]
The block »remainder of« calculates a division and returns the remainder of the division.

Input values:
- Number to be divided (dividend).
- Number, divisor.
Return value:

- Number, rest of the division.

**»constrain« [Expert-block]**

The block »constrain« ensures that given boundaries will not be exceeded.

Input values:

- Number, that will be constrained.
- Number, lower bound.
- Number, upper bound.

Return value:

- Number, constrained by the lower and the upper bound.

**»random integer« [Expert-block]**

With the block »random integer« you may generate random integer numbers within defined limits

Input values:

- Number, lower bound
- Number, upper bound

Return value:

- Number, a whole random number from within the upper and lower bounds.

**»random fraction« [Expert-block]**

With the block »random fraction« a random value between 0.0 and 1.0 is calculated.

Return value:

- Number, positive random value between 0.0 and 1.0.

---

**Text**

»Text« blocks perform simple operations on text strings.

Available »Text« blocks in NEPO are:

- Text
- create text [Expert-block]
- append text [Expert-block]

---

**»Text«**

The simple »Text« block creates a little text.

Input value:

- String, containing arbitrary characters.
Return value:

- String, containing arbitrary characters.

»comment«

Document your program with this block, so that you and others will find it easier to understand your program later. This comment will also be visible in the generated source code.

Input value:

- String, containing arbitrary characters.

»append Text« [Expert-block]

The »append text« block will append some string to a given string, for instance to extend a message with a signature.

Input values:

- String, to which some other text will be appended.
- String, that shall be appended.

Lists [Expert-blocks]

A list is a sequence of data (of identical parameter types) with elements in a fixed succession. Create a »list« type variable to make use of a »Lists« block. NEPO provides five different parameter types for lists: number, string, boolean, colour and connection.

NEPO provides the following »Lists« blocks:

- create list
- repeat element in list
- length of
- is empty?
- find in list
- get list element
- set list element
- get sublist

»create list«

The block »empty list« creates a list with no content. The block »list« generates a list with some predefined values.

This block may only be used in the context of a »set variable« block.

Using »++« or »--« enables you to extend or reduce the list at its end.

Settings and input values:

- List type, »Number«, »String«, »Boolean«, »Colour«, »Connection«.
- Create further list element, append to the end of the list.
- Delete list element at the end of the list.
- Values, according to the list type. Here the input of initial values is possible.

Return value:

- Empty list or list, containing elements of the specified list type.

»repeat element in list«
The »repeat element in list« block generates a list of equal elements.

Settings and input values:
- List type, »Number«, »Boolean«, »String«, »Colour« or »Connection«.
- List element, according to the selected list type. This value will be repeated in the list.
- Number, defining how often the list element will be repeated.

Return value:
- List of the specified list type, containing the specified number of equal elements.

»length of«

The »length of« block returns a value which is the length of the list given as parameter. An empty list has a length of 0.

Input value:
- List, evaluated by an appropriate block.

Return value:
- Number, number of list elements.

»is empty?«

A list given as parameter will be checked whether it is empty.

Input value:
- List, evaluated by an appropriate block.

Return value:
- Boolean, either »true« or »false«.

»find in list«

A list is searched for an item. If the item is in the list, the list position will be returned. If the item is not in the list the result is -1.

Settings and input values:
- List, that will be examined.
- Position of the occurrence, either »first« or »last«.
- Value, list item to be found.

Return value:
- Number, indicating the position where the element was found in the list.

Note: Counting list positions will start with 0.

»get list element«

This block accesses an item of a list. Depending on the settings this item may be altered.

A list is given as parameter. Then a drop-down-list specifies what will happen to the list item under consideration.

Settings:
- List, that is under consideration.
- Option, action for the element found: »get« reads the element and leaves it unchanged, »get and remove« reads the element and removes it from the list, »remove« just removes the element from the list.
- Option, position of the element found: »#«, »# from end«, »first«, »last« or »random«.
- Number, indicating the list position. This input value is not required if »first«, »last« or »random« was selected as position.

Note: Counting of list positions starts with 0.

Return value:
- List element that has been found at the specified list position; »undefined«, if the list position does not exist.

With selection of »remove« there is no return value; instead the list will be shortened by this list element.

»set list element«

In a list given as input parameter one specified element will be replaced by a new value.

Settings and input values:
- List, that will be changed.
- Option, action for the element found; »set« changes the element, »insert at« inserts a new element into the list.
- Option, position for the element to be changed: »#«, »# from end«, »first«, »last« or »random«.
- Number, indicating the list position. This input value is not required if »first«, »last« or »random« was selected.

Note: Counting of list positions begins with 0.
- List element, that will be set or inserted at the selected list position.

»get sublist«

From a list given as parameter a sublist will be created. The sublist contains all those elements that match the further specifications of the block.

Settings and input values:
- List to be examined.
- Position, start of the sublist: »#«, »# from end« or »first«.
- Number, indicating the list position. This input value is not required if »first« has been selected.

Note: Counting of list positions starts with 0.
- Position, end of the sublist: »#«, »# from end« or »last«.
- Number, indicating the list position. This input value is not required if »last« has been selected.

Note: Counting of list positions starts with 0.

Return value:
- List, a sublist of the same type as the given list.
Variables

The »Variables« blocks are used to create global variables of six different parameter types. The global variables store values which may be used at arbitrary positions in a program.

Each variable has to be created prior to its use. Create a variable by clicking the »+« sign in the »start « block. Each variable has to be assigned a name and a parameter type. Further clicks on the »+« sign will create additional variables.

The name of a variable has to be unique in a program.

Settings and input values:

- Insert another variable.
- Delete this variable.
- Variable name, special characters and space sign are not allowed.
- Variable type, »number«, »boolean«, »string«, »colour«, »connection«, »list number«, »list boolean«, »list string« or »list connection«
- Initial values, according to the variable type.

These »Variables« blocks are available in NEPO:

- set variable
- get variable

»set variable«

Using the block »set variable« will assign a value to a variable. Depending on the variable type the value may be assigned by an input connector.

Settings and input value:

- Variable, which value is to be changed.
- Value, new value for the variable, evaluated from a suitable block.

»get variable«

Using the block »get variable« returns the value of a variable to another block. The type of the output parameter is equal to the type that has been assigned to the variable in the »start« block.

Settings:

- Variable, that will be read. The value will not be changed by reading.

Return value:

- Value, stored in the variable.

Functions [Expert-blocks]

With »Functions« blocks (in Java also called »methods« or »subprograms«) you may create more readable programs. NEPO offers some function blocks:

- Function blocks with/without input parameters and no return parameter
- Function blocks with/without input parameters and one return parameter
- If block to be used within a function

With elementary function blocks you may program simple and complex functions which may be used by function calls. Each programmed function is available immediately as an additional block in the »Functions« category. Those are

- Function calls with/without input parameters and without return parameter
- Function calls with/without input parameters and also one return parameter
Depending from the number of input and return parameters the selection of function blocks varies.

A unique name is assigned by NEPO to each new function definition (doSomething, doSomething2, ...). This name may be altered. Overwriting of a defined function is not possible.

**»Function blocks with/without input parameters and no return statement«**

In a function block with/without input parameter and without return parameter a sequence of program statements will be condensed. The input parameters will be determined by the local variables of the function block. Local variables may be generated by using the `++` sign of the function block. Local variables will not be initialized.

If a function block contains an if-block the function may be terminated before reaching its end.

The function block is available in the »Functions« category immediately after its definition. Using function blocks improves the readability of complex programs.

Settings:

- Function name, no special characters and white spaces allowed.
- Generate new local variables, that will be assigned with the function call.
- Delete the associated local variable.

- The name of a function block has to start with a lower case letter. Special characters are not allowed in a function block name.
- If the function block defines input parameters (local variables), all the parameter slots have to be occupied when calling the function.

**»Function blocks with/without input parameters with return statements«**

In a function block with/without input parameter and with return parameter a sequence of program statements will be condensed. The input parameters will be determined by the local variables of the function block. Local variables may be generated by using the `++` sign of the function block. Local variables will not be initialized. After running through all the blocks of the function a value will be returned.

If a function block contains an if-block the function may be terminated before reaching its end. An alternative value may be returned by the if-block.

The function block is available in the »Functions« category immediately after its definition. Using function blocks improves the readability of complex programs.

Settings:

- Function name, no special characters and white spaces allowed.
- Generate new local variables, that will be assigned with the function call.
- Delete the associated local variable.
- Data type for the return value: choose one data type.
- Block, that returns a value of the defined data type.

Return value:

- Value of the defined data type, evaluated in an appropriate block.

- The name of a function block has to start with a lower case letter. Special characters are not allowed in a function block name.
- If the function block defines input parameters (local variables), all the parameter slots have to be occupied when calling the function.
»if block to be used within a function«

The if statement within a function is of special importance. As the function sequence meets an if statement the validity of the condition will be checked.

If statement within a function with return value:

- If the condition evaluates to »true« the rest of the function will be ignored and the function terminates. The second input value of the if statement will be returned. A possibly defined return value of the function will be ignored. The return value data type is determined by the return data type of the function definition.
- If the condition evaluates to »false« the sequence of the function blocks will be continued. The return value of the function will be returned after reaching the end of the function.

If statement within a function with no return value:

- An if statement within a function without return value has just the if statement as input parameter.
- If the condition evaluates to »true« the rest of the function will be ignored and the function terminates.
- If the condition evaluates to »false« the sequence of the function blocks will be continued.

Settings and input values:

- Boolean value, possibly evaluated by a condition.
- Value of the specified data type.

Return value:

- Value of the specified data type.

»function call without return value «

With this block you can call a previous defined function in your program.

input values:

- element, depending of the required input value of the function.

»function call with return value «

With this block you can call a previous defined function in your program.

input values:

- element, depending on the required input value of the function.

return value:

- element, depending on the return value of the function.