Loops

The “Controller” area contains blocks that control whether other blocks placed inside them are executed. There are two kinds of control blocks: **if do** blocks (which are described on a separate page) and blocks that control how often the action inside them is executed. The latter are called loops, since the action inside them, called the loop body or body may be repeated multiple times. Each run of a loop is called an iteration.

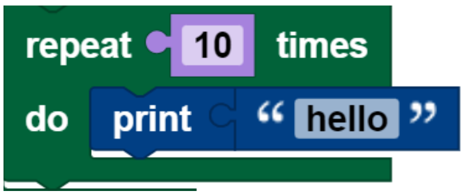
Blocks for creating loops

repeat continuously

The **repeat continuously** block executes the code in the body until the program ends.

repeat

The **repeat** block executes the code in the body as many times ad indicated. The following block, for example, will output “Hello!” ten times:

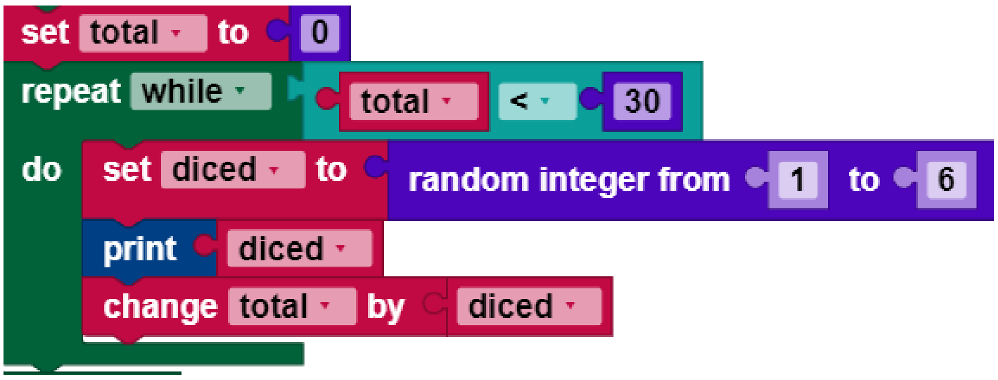


```
repeat 10 times
do  print “hello”
```

repeat as long as

Imagine a game in which a player throws a dice and adds up all of the values shown, as long as the total is less than 30. The following blocks implement this game:

1. A variable named **total** contains an initial value of 0.
2. The loop starts with a check whether **total** is less than 30. If so, the blocks in the body are run.
3. A random integer between 1 and 6 is generated (to simulate a dice value) and a variable named **diced** is saved.
4. The thrown (“diced”) number is output.
5. The variable **total** is increased by the number thrown, or **diced**.
6. Once the end of the loop is reached, the controller goes back to step 2.

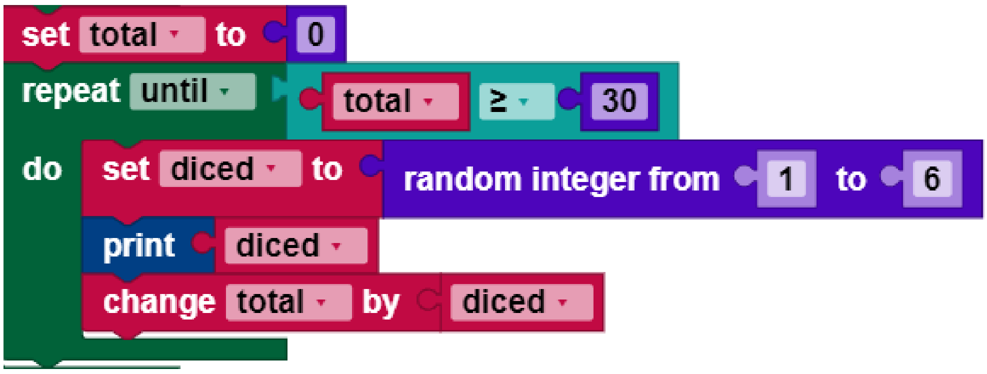


```
set total to 0
repeat while  total < 30
do  set diced to  random integer from 1 to 6
    print diced
    change total by diced
```

After the loop is ended, the controller runs through all of the following blocks (not shown). In the example, the loop ends after a certain number of random integers between 1 and 6 have been output, and the variable **total** then has the value of the total of these numbers, which is at least 30.

repeat until

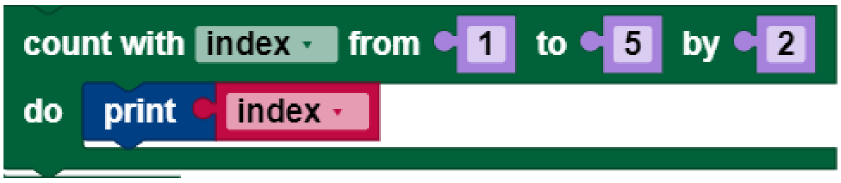
repeat as long as loops repeat their body **as long as** a condition is fulfilled. **Repeat until** loops are similar, with the difference that they repeat the body **until** a certain condition is fulfilled. The following blocks are equivalent to the previous example, because the loop runs until **total** is greater than or equal to 30.



```
set total to 0
repeat until total ≥ 30
do  set diced to random integer from 1 to 6
    print diced
    change total by diced
```

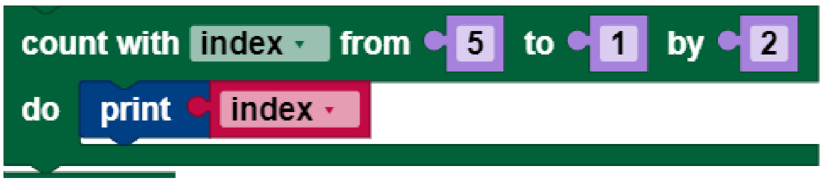
count from to

The **count from to** loop increases the value of a variable, starting with an initial value and ending with a second value, and in steps from a third value, whereby the body is executed once for each value of the variable. The following program, for example, outputs the numbers 1, 3, and 5.

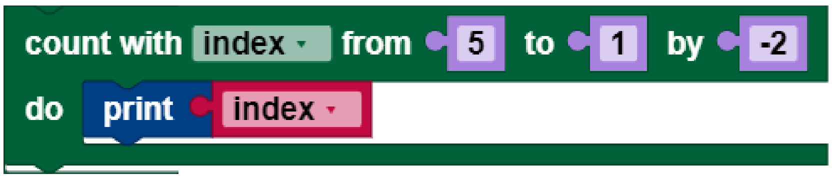


```
count with index from 1 to 5 by 2
do  print index
```

As the following two loops show, which each output the numbers 5, 3 and 1, this first value can be greater than the second. The behavior is the same, regardless of whether the incremental amount (third value) is positive or negative.



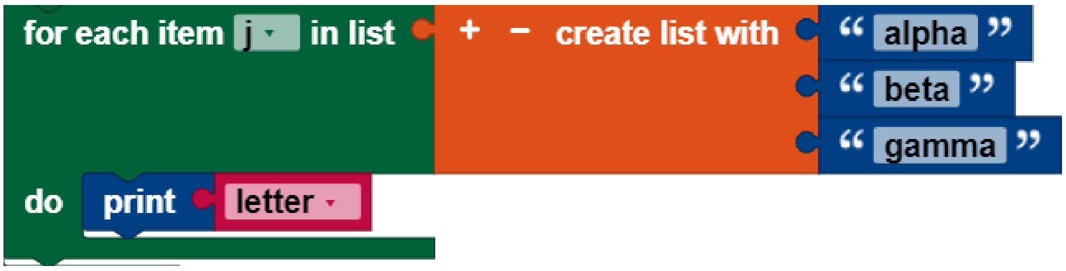
```
count with index from 5 to 1 by 2
do  print index
```



```
count with index from 5 to 1 by -2
do  print index
```

for each

The **for each** block is similar to the **count from to** loop, but instead of the loop variables in a numerical sequence, it uses the values from a list in sequence. The following program outputs each element in the list “alpha,” “beta,” “gamma”:



```
for each item j in list  + − create list with  “ alpha ”
                                               “ beta ”
                                               “ gamma ”
do  print  letter
```

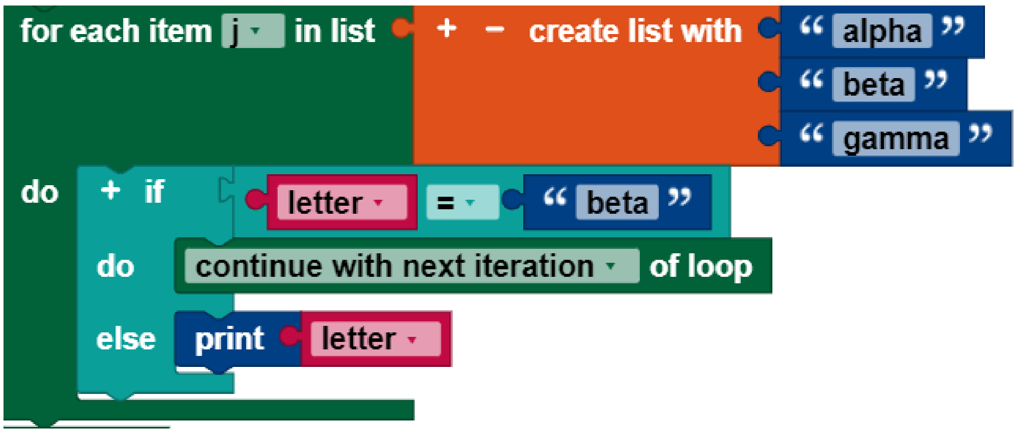
Break out blocks

Most loops are run until the abort condition (for **repeat** blocks) is fulfilled, or until all values for the loop variable have been taken (for **count with** and **for each** loops). Two rarely needed, yet occasionally used blocks offer additional options for controlling loop behavior. They can be used with any kind of loop, even though the following example shows their use in the **for each** loop.

continue with next iteration

continue with next iteration causes the remaining blocks in the loop body to be skipped, and the next iteration of the loop to begin.

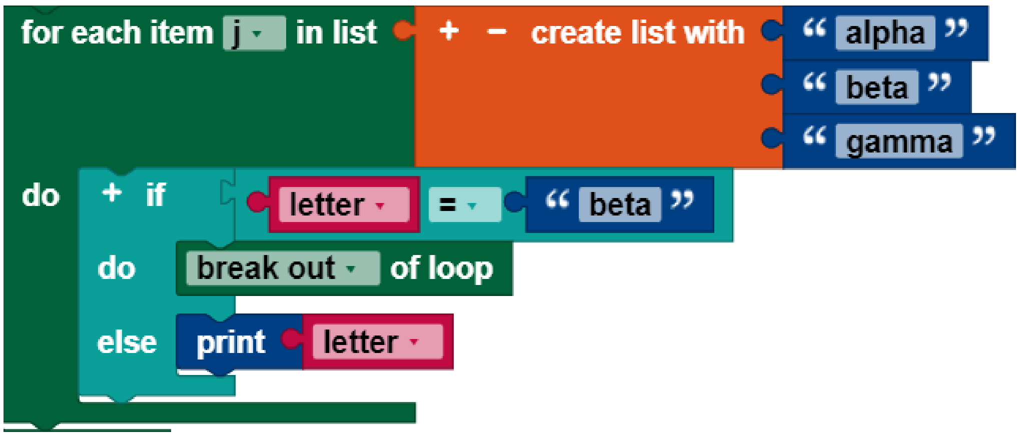
The following program outputs “alpha” during the first iteration of the loop. During the second iteration, the block **continue with next iteration** is executed, causing the output of “beta” to be skipped. In the last iteration, “gamma” is printed.



```
for each item j in list  + − create list with  “ alpha ”
                                               “ beta ”
                                               “ gamma ”
do  + if    letter = “ beta ”
      do    continue with next iteration of loop
      else  print  letter
```

Break out

The **break out** block makes it possible to prematurely exit a loop. The following program outputs “alpha” for the first iteration, then breaks out of the loop during the second iteration when the loop variable equals “beta.” The third point in the list is never reached.



```
for each item j in list  + − create list with  “ alpha ”
                                               “ beta ”
                                               “ gamma ”
do  + if    letter = “ beta ”
      do    break out of loop
      else  print  letter
```

Revision #1
Created 21 February 2022 15:50:51 by Admin
Updated 9 January 2023 10:29:05 by Admin