Loops

For Loops

For Loops

The **for** statement has two variants: the *numerical* **for** and the *generic* **for**.

Numerical For

Numerical For

A numerical **for** has the following syntax:

```
for var = exp1, exp2, exp3 do
  something
end
```

Numerical For

A numerical **for** has the following syntax:

```
for var = exp1, exp2, exp3 do
  something
end
```

This loop will execute `something` for each value of `var` from `exp1` to `exp2` using `exp3` as the *step* to increment `var`.

Numerical For

A numerical **for** has the following syntax:

```
for var = exp1, exp2, exp3 do
  something
end
```

This loop will execute `something` for each value of `var` from `exp1` to `exp2` using `exp3` as the *step* to increment `var`.

This third expression is optional. When absent, Lua assumes one as the step value.

Numerical For

Consider a program to print all even numbers until 20.

```
for i = 0,20,2 do
    print(i)
end
```

Numerical For

A very important aspect of the numerical is that the control variable is a local variable declared by the for statement, and it is visible **only inside** the loop. A common mistake is to assume that the variable still exists after the loop ends:

```
for i = 1, 20 do print(i) end
solution = i -- wrong!
                -- i does not exist outside
                -- the for loop
```

Numerical For

We can access the value of the control variable anytime we want and save it outside the loop in a very simple way.

```
-- find index of negative value in a list
 local position = nil
 for i = 1, #a do
   if a[i] < 0 then
     position = i -- save value of 'i'
     break
   end
 end
 print(position)
```