Control Structure Tutorial

Lua control structures will be familiar to programmers. Sections 2.4.4 and 2.4.5 of the Reference Manual are a little sparse but provide the necessary information. The conditional expressions mentioned ("exp") on this page can be read about in the ExpressionsTutorial. Please be sure to read the "*Note on test expressions and nil*" at the end of the ExpressionsTutorial.

while

The conditional looping statement `while` has the form:

`while` *exp* **`do`** *block* **`end`**

For example, a simple loop:

```
> i = 3
> while i>0 do
>>  print(i)
>>  i = i-1
>> end
3
2
1
```

We can exit the control of a `while` statement using the `break` keyword. Note, in Lua the `break` keyword must be the last statement in a block, i.e. the keyword `end` must follow. You will get compilation errors if you don't have `break end`.

```
> a,b = 0,1
> while true do               -- infinite loop
>>  io.write(b, ", ")
>>  a,b = b,a+b
>>  if a>500 then break end   -- exit the loop if the condition is true
>> end
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, >
```

repeat

The conditional looping statement `repeat` has the form:

`repeat` *block* **`until`** *exp*

For example, a simple loop:

```
> i = 3
> repeat
>>  print(i)
>>  i = i-1
>> until i==0
3
2
1
```

Like the `while` statement we can exit a `repeat` loop using a `break` statement:

```
> i = 1
> repeat
>>  print(i)
>>  i = i+1
>>  if i>3 then break end
>> until cows_come_home
1
2
3
```

`cows_come_home` is a variable which is not defined. When we access it we get the value `nil`, which means "until false", or forever.

for

The iterating statement `for` has two forms. The first is for numerical iteration, e.g.,

```
> for count = 1,3 do print(count) end  -- numerical iteration
1
2
3
```

The second is for sequential iteration, e.g. to print the contents of a table:

```
> for key,value in {10, math.pi, "banana"} do print(key, value) end
1       10
2       3.1415926535898
3       banana
```

The above example is syntactic sugar for following example. `for` is passed an *iterator* function, which here is `pairs()`, whose purpose it is to supply the values of each iteration:

```
> for key,value in pairs({10, math.pi, "banana"}) do print(key, value) end
1       10
2       3.1415926535898
3       banana
```

There is more detail on all the forms of `for` in the ForTutorial.

if ... then ... else ... end

The statement `if` has the form:

`if` *exp* **`then`** *block* { **`elseif`** *exp* **`then`** *block* } [**`else`** *block*] **`end`**

For example, `if` ... `then` ... `end`

```
> if 10>2 then print("bigger") end
bigger
```

`if` ... `then` ... `else` ... `end`

```
> if 1>10 then print("bigger") else print("smaller") end
smaller
```

`if` ... `then` ... `elseif` ... `else` ... `end`

```
> number = 3
> if number < 1 then
>>   value = "smaller than one"
>> elseif number==1 then
>>   value = "one"
>> elseif number==2 then
>>   value = "two"
```

```
>> elseif number==3 then
>>   value = "three"
>> else
>>   value = "bigger than three"
>> end
> print(value)
three
```

FindPage · RecentChanges · preferences
edit · history
Last edited March 8, 2004 10:55 am PDT (diff)