Lesson 2, Week 3: Structures I
(`if ... elseif ... else ... end`)

AIM

— To learn to use `if` blocks to write code branches

After this lesson, you will be able to

* Use a simple `if` block to branch between code that executes or not depending on a logical test
* Use `elseif` and `else` blocks inside an `if` block to choose among several code branches

The reserved keyword `if`

Like `function`, the keyword `if` opens a code block which must be closed with the keyword `end`.

However, they differ as to scope: the scope of an `if ... end` block is global; the keyword `if` does not create a local scope.

`if` must be followed by a truth value—normally, the result of a logical test. The code body of the `if` block consists of the lines after the logical test and before the `end`.

An `if` block may contain only one code path. In that case, `if true ... end` will ensure that the code in the code path is executed, and `if false ... end` will simply do nothing. This is the simplest kind of branch you can introduce in your code.

Simple examples

Let's start with `if (a > 1) println("a is greater than 1") end` The code body is a single line, so it is convenient to write the entire `if` block on on line. We need to assign a value to `a` before the code will run, of course. We can also play with the actual code body[1], but we'll resist doing too much.

[1] It is quite tempting to add lots of lines ...

If we wrap the :

```
function ifeg1(a)
    if (a > 1) println("a is greater than 1") end
end
```

DEMO: try a few values for `a` and in a comprehension

The comprehension `[ifeg1(x) for x in [0, 1, 2, 3]]` shows on screen the two lines that correspond to calling `println` twice—and the value `nothing` four times! We can clarify things with the lines

```
x=[(if x > 1 println("yes") end) for x in [2,3,51] ];
x
```

which separates out the creation of `x` and its display. [DEMO]

The value `nothing` goes into `x` comes from `println`—it formats its input and prints it on the screen, but it does not create any other value that can be used to make an element of `x`. If we add something that does, for example some value like a number or a string or a logical constant, that makes things even clearer. Compare

`y=[(if x > 1 println("yes"); true end) for x in [2,3,51] ];` to

`z=[(if x > 1 true; println("yes") end) for x in [2,3,51] ];`

Choosing between two code blocks

In the previous example, the `if` block chose between doing something or doing nothing. Often you want to execute one of two code blocks. The keyword `else` allows you to do that:

```
function ifeg2(a, b)
    if a > b
        println("a is bigger than b")
    else
        println("a is not bigger than b")
    end
end
```

DEMO:

Note that code block starting with `else` is the if-all-else-fails option: it runs whenever the logical test is results in `false`. This will become clearer in the next section.

Choosing among three or more options

Let's write a function that specifies Goldilocks' dialogue:

```
function goldisays(porrdegreesC)
   if porrdegreesC > 55
      println("The porridge is too hot!")
   elseif porrdegreesC < 45
      println("The porridge is too cold!")
   else
      println("Hmm!  The porridge is just right.")
   end
end
```

DEMO:

You should be aware that there are usually lots of logical expressions that will give the required results. For example, we could replace the code body of the function with

```
if ((porrdegreesC > 45) && (porrdegreesC < 55)) println("Hmmm!  Just right!")
elseif (porrdegreesC < 45) println("Ugh!!  Too cold!")
else println("Ouch!  Too hot!")
end
```

This also shows that the whole `if` block is just one statement, as in

`if true print("true") else print("false") end` and

```
if false print("first true") elseif false print("second true") else print("none true") end
```

A word of advice

If you read about coding style, you are likely to get warnings about "elseif ladders" and how they're bad style. Such ladders can happen when your logical tests must pick one of many options (that is, quite a few more than three). You test for one, then you test for the next, and so on, until all of the options are tested. Every test is an `elseif` rung on the ladder.

That kind of coding problem is undoubtedly hard. The logic for choosing where and what to eat when you go out may mean choosing among several eateries, and several menu options, and how you travel depends on where you go and what (or indeed whether) you have dessert depends on what you had before, and so on. It can be a long job to code, a very hard job to test properly, and a nightmare for anybody else to read[2].

My advice is, don't worry about style. Get your `if` blocks to work properly with 4 or 6 options, no matter how ugly they are. Always use only one `if` block for choosing one among many options. Avoid especially trying to use a sequence of separate `if` blocks, you can never be sure that only one of them will be executed. Once you have mastered getting it right you can start working on making it look good and making it readable.

[2]Including you yourself a months or sometimes a few days later.

Review and summary of Lesson

* An `if ...end` code block has global scope
* Syntax is `if (<logical test> ... elseif (<logical test>) ... else ...end)`
* Each logical test is an expression that evaluates either to `true` or `false`
* You may repeat `elseif` code blocks inside the `if` block
* Beginners shouldn't worry about the style of their `if`, but should instead focus on making sure the (formal) logic is correct