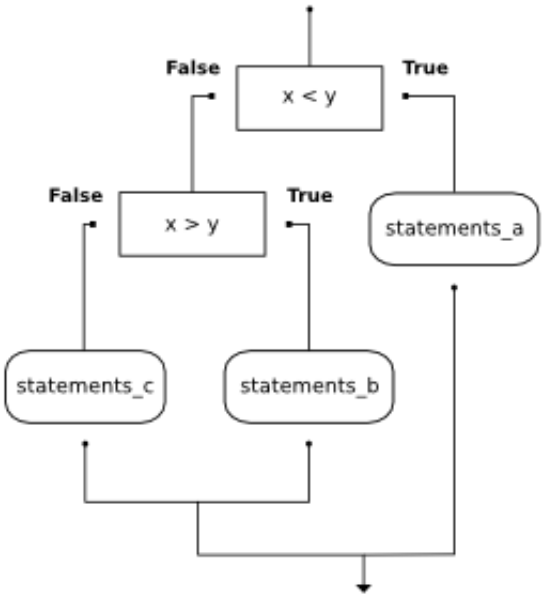
# Nested conditionals

One conditional can also be **nested** within another. For example, assume we have two integer variables, `x` and `y`. The following pattern of selection shows how we might decide how they are related to each other.

```
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```

The outer conditional contains two branches. The second branch (the else from the outer) contains another `if` statement, which has two branches of its own. Those two branches could contain conditional statements as well.

The flow of control for this example can be seen in this flowchart illustration.




Here is a complete program that defines values for `x` and `y`. Run the program and see the result. Then change the values of the variables to change the flow of control.

```
x = 10
y = 10

if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")

```

**ActiveCode: 1** (sel2)

Run

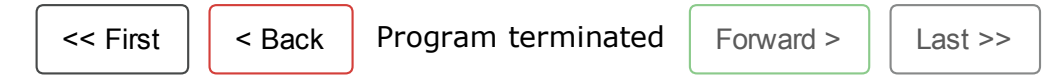
```
x and y must be equal
```

**Note**

In some programming languages, matching the if and the else is a problem. However, in Python this is not the case. The indentation pattern tells us exactly which else belongs to which if.

If you are still a bit unsure, here is the same selection as part of a codelens example. Step through it to see how the correct `print` is chosen.

```
 1  x = 10
 2  y = 10
 3
 4  if x < y:
 5      print("x is less than y")
 6  else:
 7      if x > y:
 8          print("x is greater than y")
 9      else:
10          print("x and y must be equal")
```




line that has just executed

next line to execute

Program output:

```
x and y must be equal
```

Frames Objects

Global variables

| | |
|---|---|
| x | 10 |
| y | 10 |

**CodeLens: 1 (sel1)**

**Check your understanding**

sel-9: Will the following code cause an error?

```
x = -10
if x < 0:
    print("The negative number ",  x, " is not valid here.")
else:
    if x > 0:
        print(x, " is a positive number")
    else:
        print(x," is 0")
```

a) No

b) Yes

Check Me | Compare Me

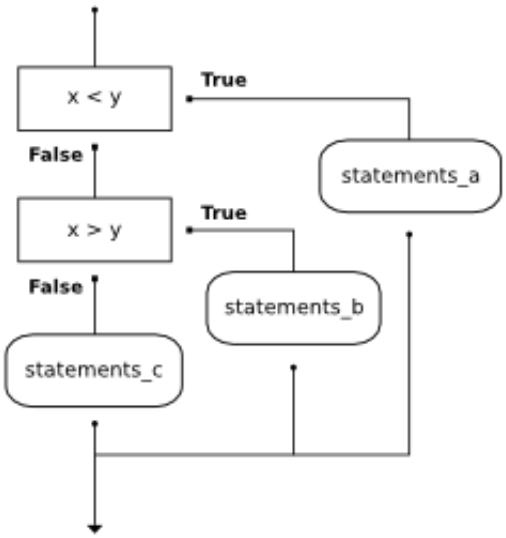
Correct!! This is a legal nested if-else statement. The inner if-else statement is contained completely within the body of the outer else-block.

# Chained conditionals

Python provides an alternative way to write nested selection such as the one shown in the previous section. This is sometimes referred to as a **chained conditional**

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```

The flow of control can be drawn in a different orientation but the resulting pattern is identical to the one shown above.




`elif` is an abbreviation of `else if`. Again, exactly one branch will be executed. There is no limit of the number of `elif` statements but only a single (and optional) final `else` statement is allowed and it must be the last branch in the statement.

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Here is the same program using `elif`.

```
x = 10
y = 10

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")

```

**ActiveCode: 2** (sel4)

Run

```
x and y must be equal
```

**Note**

This workspace is provided for your convenience. You can use this activecode window to try out anything you like.

```

```

**ActiveCode: 3** (scratch_06_02)

Run

**Check your understanding**

sel-10: Which of I, II, and III below gives the same result as the following nested if?

```
# nested if-else statement
x = -10
if x < 0:
    print("The negative number ",  x, " is not valid here.")
else:
    if x > 0:
        print(x, " is a positive number")
    else:
        print(x, " is 0")
```

```
I.

if x < 0:
    print("The negative number ",  x, " is not valid here.")
else (x > 0):
    print(x, " is a positive number")
else:
    print(x, " is 0")
```

```
II.

if x < 0:
    print("The negative number ",  x, " is not valid here.")
elif (x > 0):
    print(x, " is a positive number")
else:
    print(x, " is 0")
```

```
III.

if x < 0:
    print("The negative number ",  x, " is not valid here.")
if (x > 0):
    print(x, " is a positive number")
else:
    print(x, " is 0")
```

- a) I only
- b) II only
- c) III only
- d) II and III

e) I, II, and III

Check Me | Compare Me

Correct!! Yes, II will give the same result.

sel-11: What will the following code print if x = 3, y = 5, and z = 2?

```
if x < y and x < z:
    print ("a")
elif y < x and y < z:
    print ("b")
else:
    print ("c")
```

a) a

b) b

c) c

Check Me | Compare Me

Correct!! Since the first two Boolean expressions are false the else will be executed.

# Boolean Functions

We have already seen that boolean values result from the evaluation of boolean expressions. Since the result of any expression evaluation can be returned by a function (using the `return` statement), functions can return boolean values. This turns out to be a very convenient way to hide the details of complicated tests. For example:

```
1 def isDivisible(x, y):
2     if x % y == 0:
3         result = True
4     else:
5         result = False
6 
7     return result
8 
9 print(isDivisible(10,5))
10 
```

**ActiveCode: 4** (ch06_boolfun1)

Run

```
True
```

The name of this function is `isDivisible`. It is common to give **boolean functions** names that sound like yes/no questions. `isDivisible` returns either `True` or `False` to indicate whether the `x` is or is not divisible by `y`.

We can make the function more concise by taking advantage of the fact that the condition of the `if` statement is itself a boolean expression. We can return it directly, avoiding the `if` statement altogether:

```
def isDivisible(x, y):
    return x % y == 0
```

Boolean functions are often used in conditional statements:

```
if isDivisible(x, y):
    ... # do something ...
else:
    ... # do something else ...
```

It might be tempting to write something like `if isDivisible(x, y) == True:` but the extra comparison is not necessary. The following example shows the `isDivisible` function at work. Notice how descriptive the code is when we move the testing details into a boolean function. Try it with a few other actual parameters to see what is printed.

```
def isDivisible(x, y):
    if x % y == 0:
        result = True
    else:
        result = False

    return result

if isDivisible(10,5):
    print("That works")
else:
    print("Those values are no good")

```

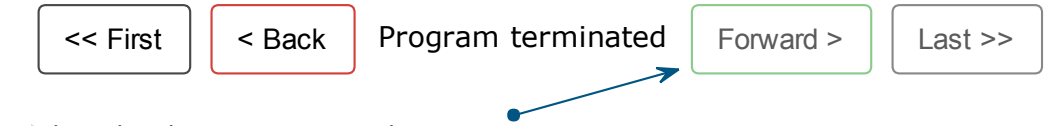
**ActiveCode: 5** (ch06_boolfun2)

Run

```
That works
```

Here is the same program in codelens. When we evaluate the `if` statement in the main part of the program, the evaluation of the boolean expression causes a call to the `isDivisible` function. This is very easy to see in codelens.

```
 1 def isDivisible(x, y):
 2     if x % y == 0:
 3         result = True
 4     else:
 5         result = False
 6
 7     return result
 8
 9 if isDivisible(10,5):
10     print("That works")
11 else:
12     print("Those values are no good")
```




line that has just executed

next line to execute

Program output:

```
That works
```

Frames Objects

Global variables

isDivisible

function
isDivisible(x, y)

**CodeLens: 2 (ch06_boolcodelens)**

**Check your understanding**

sel-12: What is a Boolean function?

- a) A function that returns True or False
- b) A function that takes True or False as an argument
- c) The same as a Boolean expression

Check Me Compare Me

Correct!! A Boolean function is just like any other function, but it always returns True or False.

sel-13: Is the following statement legal in Python (assuming x, y and z are defined to be numbers)?

```
return x + y < z
```

- (•) a) Yes
- ( ) b) No

Check Me | Compare Me

Correct!! It is perfectly valid to return the result of evaluating a Boolean expression.

**Note**

This workspace is provided for your convenience. You can use this activecode window to try out anything you like.

```
1
2
```

**ActiveCode: 6** (scratch_06_03)

Run