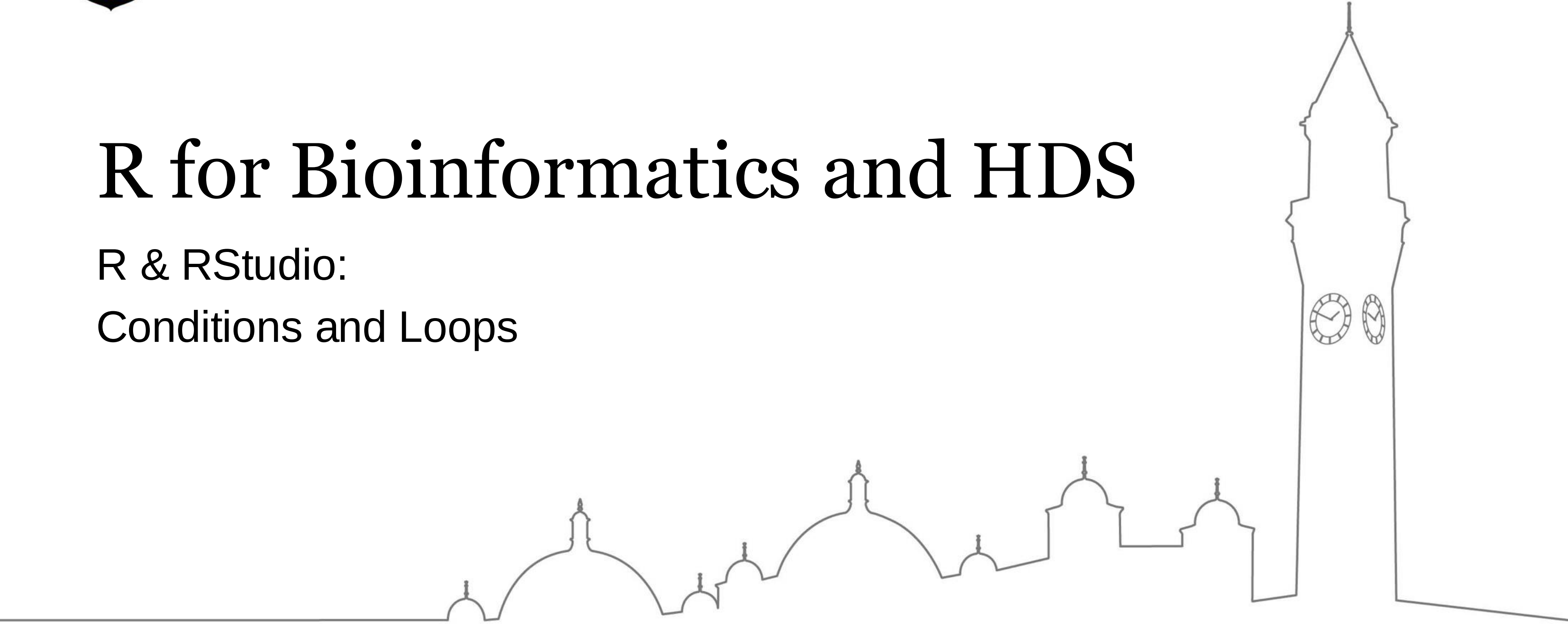
R for Bioinformatics and HDS

R & RStudio:

Conditions and Loops



Vasileios Panagiotis Lenis

Laura Bravo Merodio

Course Structure

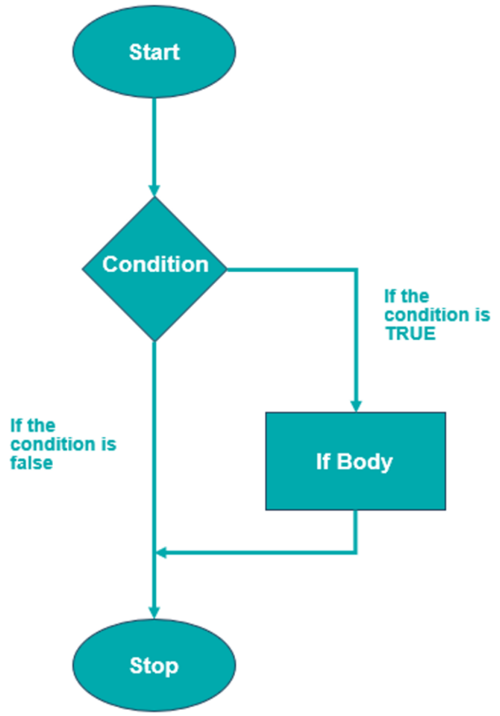
- Introduction to R & RStudio
- Syntax, Comments, Variables, Data Types and Operators
- Conditions, Loops, Functions and Data Structures
- Working with Data frames

UNIVERSITY OF BIRMINGHAM

Conditions: How to control your program

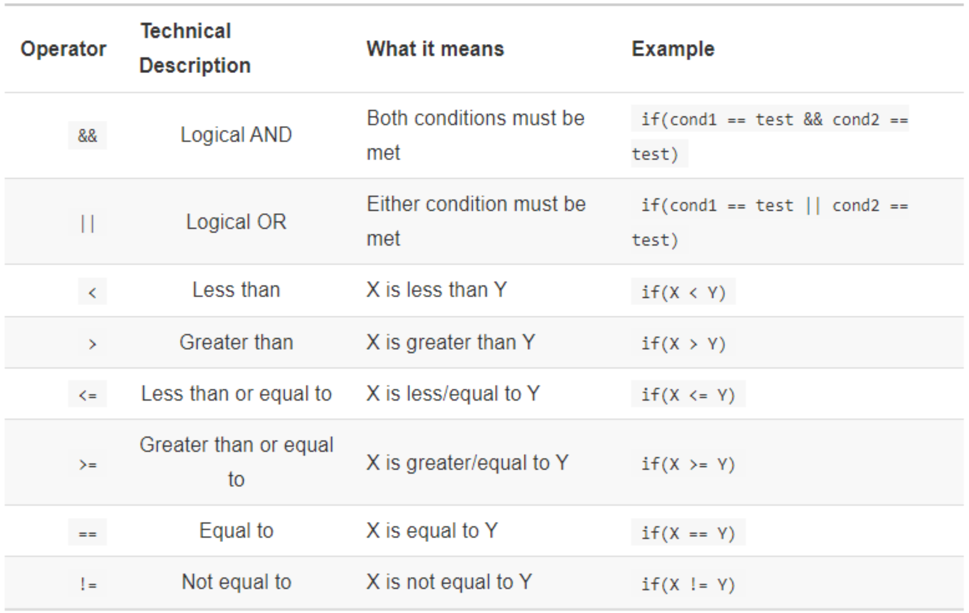
- Conditions provide us with the ability to control what the program does and perform different actions based on logic statements.

- Using "*if, then*" logic statements we control the flow of our program allowing it to make choices based on the values they are manipulating.



UNIVERSITY OF BIRMINGHAM

Types of Operator in R



Operator	Technical Description	What it means	Example
`&&`	Logical AND	Both conditions must be met	`if(cond1 == test && cond2 == test)`
`||`	Logical OR	Either condition must be met	`if(cond1 == test || cond2 == test)`
`<`	Less than	X is less than Y	`if(X < Y)`
`>`	Greater than	X is greater than Y	`if(X > Y)`
`<=`	Less than or equal to	X is less/equal to Y	`if(X <= Y)`
`>=`	Greater than or equal to	X is greater/equal to Y	`if(X >= Y)`
`==`	Equal to	X is equal to Y	`if(X == Y)`
`!=`	Not equal to	X is not equal to Y	`if(X != Y)`

- There are two distinct types of operators, Relational and Logical
- Relational tells us how R objects relate
- Logical allows the combining of logical values

UNIVERSITY OF BIRMINGHAM

The *if()* statement

- The most used conditional statement is ***`if`***
- The syntax is defined as '`if(condition){your code}`'
- Broadly means 'if X is TRUE, do a thing'

```
if (condition) {

  Code executed when the condition is TRUE



  }
```

UNIVERSITY OF BIRMINGHAM

The *if()* statement in use

```
x <- -10 # Define a variable
if (!x > 0)
{ print("x is a negative number")
} # State your code
"x is a negative number"
```

- Here we define our variable x
- Set our if statement as a relational statement meaning ‘if x is NOT greater than 0
- If the condition is fulfilled, the code is executed, and the statement is printed

UNIVERSITY OF BIRMINGHAM

The *else()* statement

- The logic of the `‘if’` statement can be expanded by adding `‘else’`
- The syntax is defined as `‘if(condition){codeA} else’{codeB}’`
- Meaning *‘if X is TRUE, do a thing, or else do something else’*
- It is used in combination with `‘if’`, not requiring a condition

```
if (condition) {
  Code executed when the condition is
TRUE
  } else {
  Code executed when the condition is
FALSE
  }
```

UNIVERSITY OF BIRMINGHAM

The *else()* statement in use

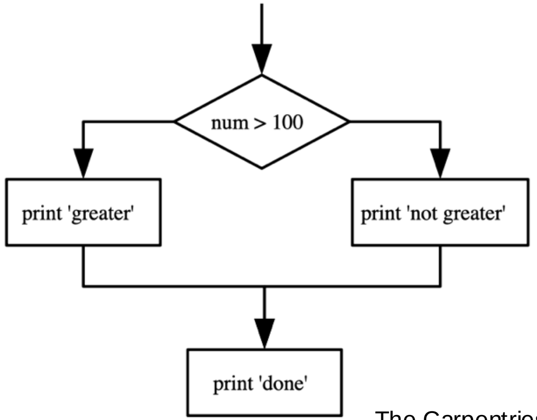
```
x <- -10 # Define a variable

if (x > 0){
    print("x is a positive number")

}else{
    print("x is a negative number")
}

"x is a negative number"
```

- We defined our variable as x <- -10
- Here, the ‘`if()`’statement (if x is greater than 0) was not TRUE
- Therefore, the ‘`else()`’ statement was executed

UNIVERSITY OF BIRMINGHAM

Example

```
num <- 37
if (num > 100) {
  print("greater")
} else {
  print("not greater")
}
print("done")
```



The Carpentries

- In the example above, the test **num > 100** returns the value **FALSE**, which is why the code inside the if block will be skipped and the code inside the else statement will run instead.

(Conditional statements don't have to include an else. If there isn't one, R simply does nothing if the test is false)

UNIVERSITY OF BIRMINGHAM

The "*for()*" loop

- Quite often we need to carry out the same procedure multiple times. This is called iteration, and the simplest form of iteration is the *for()* loop.
- The basic structure of a *for()* loop is:

```
for (iterator in set of values) {
  do a thing
}
```

The “*for()*” loop in use

```r
for (i in 1:10) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

- The 1:10 bit creates a vector on the fly; you can iterate over any other vector as well.
- We can use a *for()* loop nested within another *for()* loop to iterate over two things at once.

UNIVERSITY OF BIRMINGHAM

The “*for()*” loop in use 2

- We can use a *for()* loop nested within another *for()* loop to iterate over two things at once.

```r
for (i in 1:5) {

  for (j in c('a', 'b', 'c', 'd', 'e')) {
    print(paste(i,j))
  }

}
```

```
[1] "1 a"
[1] "1 b"
[1] "1 c"
[1] "1 d"
[1] "1 e"
[1] "2 a"
[1] "2 b"
[1] "2 c"
[1] "2 d"
[1] "2 e"
[1] "3 a"
[1] "3 b"
[1] "3 c"
[1] "3 d"
[1] "3 e"
[1] "4 a"
[1] "4 b"
[1] "4 c"
[1] "4 d"
[1] "4 e"
[1] "5 a"
[1] "5 b"
[1] "5 c"
[1] "5 d"
[1] "5 e"
```

The ***paste()*** function adds a space between the concatenated elements. The subsequent sections illustrate this.

UNIVERSITY OF BIRMINGHAM

v.p.lenis@bham.ac.uk (Vasilis)
l.bravo@bham.ac.uk (Laura)