Contents

1 Conditional statements (`if`, `else`)

The `if` and `else` keywords allow us to control what lines of code to run. Its usage is similar to other languages:

```
if (condition) { do stuff } else
if (condition) { do other stuff } else
{ do fallback stuff }
```

Several notes on this syntax:

- the `condition` must be surrounded with parentheses.
- the curly braces (`{}`) can be omitted if only a single expression is to be run
- the `else` keyword **must be read on the same line as the end of the block**, or it will be considered a separate (invalid) conditional statement.

```
x <- 123

if (x < 100) print(x)
```

The two sets of code below are equivalent. You may find one style neater than another.

```
x <- 123
if (x < 100) print("small") else
if (x < 1000) print("medium") else
print("large")
```

```
## [1] "medium"
```

```
x <- 123
if (x < 100) {
  print("small")
} else if (x < 1000) {
  print("medium")
} else {
  print("large")
}
```

```
## [1] "medium"
```

2 String concatenation

In Python we could use the + operator to concatenate strings together. In R, the `paste()` function is used. The arguments passed to `paste()` are glued together with a space character. The separator can be changed by changing the `sep` parameter.

There is also a shorthand to have no 'gluing character' with `paste0()` (i.e. it has `sep=""` by default)

```r
x <- 5
y <- 6
paste(x, "+", y, "=", x+y)
```

```
## [1] "5 + 6 = 11"
```

```r
paste(x, "+", y, "=", x+y, sep="")
```

```
## [1] "5+6=11"
```

```r
paste0(x, "+", y, "=", x+y)
```

```
## [1] "5+6=11"
```

3 Iteration

The `for` and `while` keywords are used to repeat lines of code to run. Its use is similar as in other languages:

```
for (var in iterable) { code to run }
while (condition) { code to run }
```

The `break` keyword is available to terminate the `for` or `while` loop, and the `next` keyword is available to skip until the next iteration.

Similar to conditional statements, the iterating sequence/condition must be in parentheses, and the curly braces can be omitted.

```r
for (x in 1:5) if (x %% 2 == 1) print(paste(x, "is odd"))
```

```
## [1] "1 is odd"
## [1] "3 is odd"
## [1] "5 is odd"
```

```r
for (x in c('a', 'b', 'c', 'd', 'e')) {
  print(x)
  if (x == 'c') break
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
```

```r
x = 2
while (x < 10) {x = x + 3; print(x)}
```

```
## [1] 5
## [1] 8
## [1] 11
```

4 Defining functions

To define our own functions, we use the `function()` function, with the code to run enclosed within curly braces (`{}`). If only a single expression is to be evaluated, the curly braces can be omitted.

The default behaviour is that the return value of the *last expression* in the function definition is returned. Alternatively, the return value can also be defined with the `return()` function, which will end the execution of the function.

To name the functions, we can assign them just like variables.

```r
addTwo <- function(x) x + 2
addTwo(5)
```

```
## [1] 7
```

```r
addTwoNoReturn <- function(x) y <- x + 2
addTwoNoReturn(6)
```

```r
addTwoIfEven <- function(x) {
  if (x %% 2 == 1) return (x+2)
  return(x)
}
addTwoIfEven(5); addTwoIfEven(6)
```

```
## [1] 7

## [1] 6
```