Data Handling

Lecture 2:

Programming with Data

Dr. Aurélien Sallin

Recap

DEVELOPING EMPLOYEES

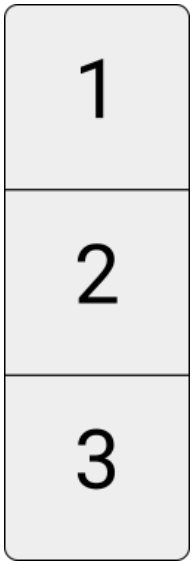
Prioritize Which Data Skills Your Company Needs with This 2×2 Matrix

by Chris Littlewood

OCTOBER 18, 2018 **UPDATED** OCTOBER 23, 2018

Basic Programming Concepts

Vectors



Vectors

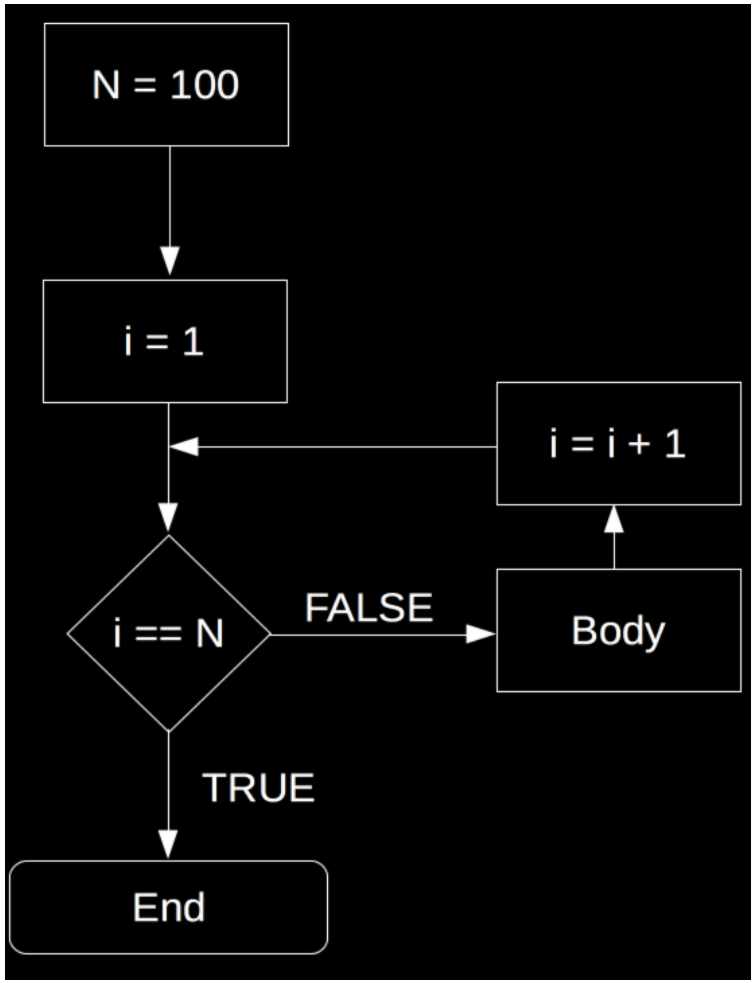
Math operators: basic arithmetic

```
# basic arithmetic
2+2
sum_result <- 2+2
sum_result
sum_result -2
4*5
20/5

# Modulo (remainder)
5 %% 3

# Integral division
5 %/% 3
```

Math operators: other operators

```
# other common math operators and functions
4^2
sqrt(4^2)
log(2)
exp(10)
log(exp(10))
```

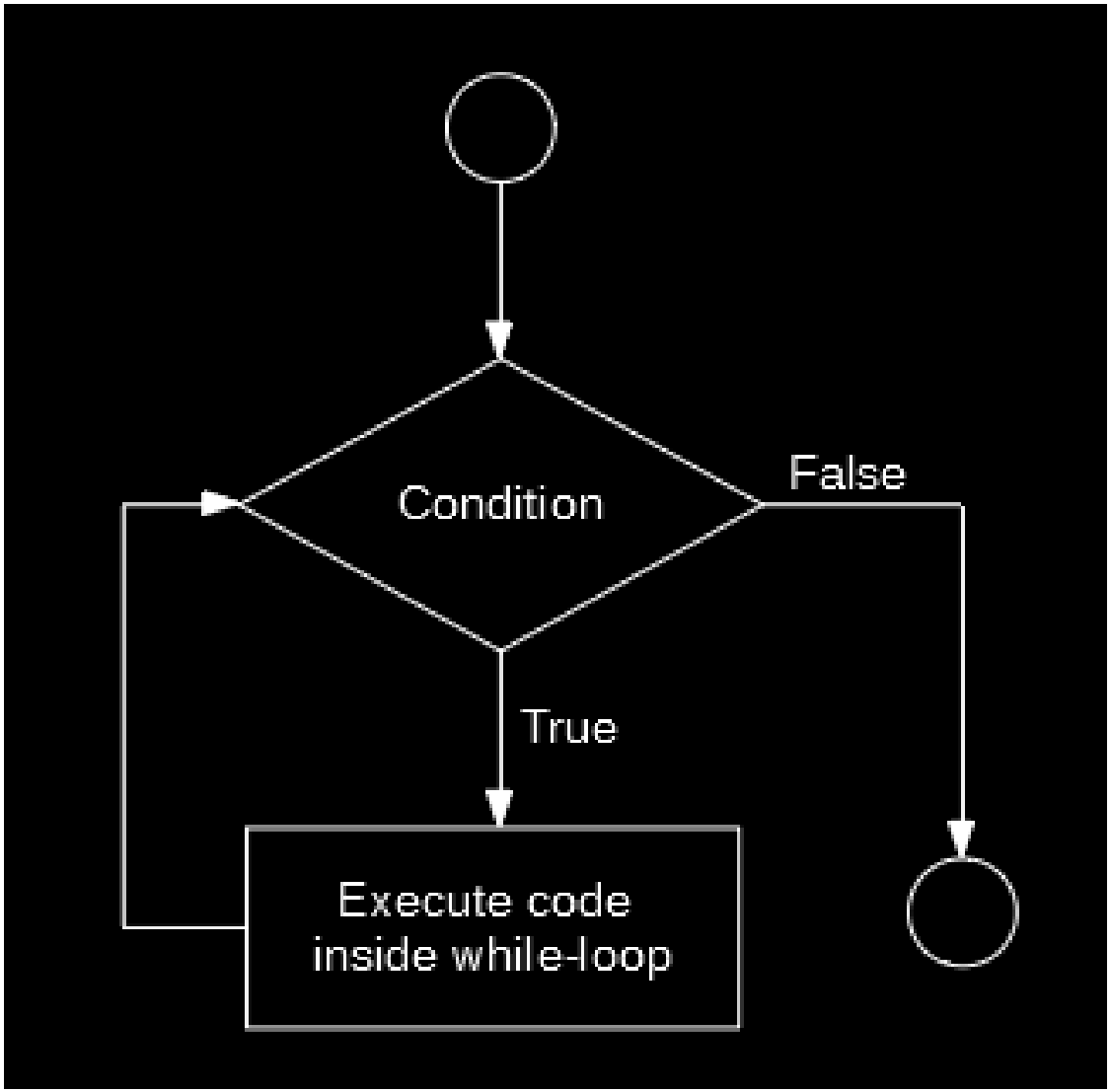
for-loop



for-loop

```r
# number of iterations

n <- 100

# start loop
for (i in 1:n) {

     # BODY

}
```

while-loop



while-loop

```r
# initiate variable for logical statement
x <- 1

# start loop
while (x == 1) {

     # BODY

}
```

Logical statements

```
2+2 == 4 # is equal to
3+3 == 7
4!=7 # is not equal to
6>3
6<7
6<=6
```

Control statements

```
condition <- TRUE

if (condition) {
     print("This is true!")
} else {
     print("This is false!")
}
```

```
## [1] "This is true!"
```

Functions

$$f : X \to Y$$

Functions

$$2 \times X = Y$$

Functions in R

Load existing functions:

```
# install a package
install.packages("<PACKAGE NAME>")
# load a package
library(<PACKAGE NAME>)
```

Functions in R

Functions have three elements:

1. **formals()**, the list of arguments that control how you call the function
2. **body()**, the code inside the function
3. **environment()**, the data structure that determines how the function finds the values associated with the names (not the focus of this course)

Functions in R

```r
myfun <- function(x, y){

  # BODY
  z <- x + y

  # What the function returns
  return(z)
}
```

```r
formals(myfun)
```

```
## $x
##
##
## $y
```

```r
body(myfun)
```

```
## {
##     z <- x + y
##     return(z)
## }
```

Functions in R

Example of a function: a simple power function

```
powerFunction <- function(base, exponent){

  results <- base ^ exponent

  return(results)
}

powerFunction(exponent = 2, base = 3)
powerFunction(base = 2, exponent = 3)
powerFunction(2, 3)
powerFunction(c(2,4,3), 3)
```

Advanced: writing better code with the “apply” family

Writing better code: the "apply" family

`apply` applies a function to margins of an array or matrix. It loops over rows (`MARGIN = 1`) or columns (`MARGIN = 2`) of a matrix.

```
# Create an empty matrix with 2 rows and 4 columns
mymatrix <- matrix(c(1,2,3, 11,12,13, 1,10),
                   nrow = 2,
                   ncol = 4)
print(mymatrix)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3   12    1
## [2,]    2   11   13   10
```

Writing better code: the “apply” family

```r
# Compute the power function of each column of the matrix

# With for-loops
for (i in 1:ncol(mymatrix)){
  print(powerFunction(mymatrix[, i], 2))
}

# With apply
apply(mymatrix, MARGIN = 2, powerFunction, exponent = 2)
```

Writing better code: the "apply" family

`lapply` applies a function over a list ("list-" apply). It loops over each element of a list to execute a function.

```
mylist <- list(1,2,5,6,90)

# With a for loop
for (i in mylist){
  powerFunction(i, 2)
}

# With lapply
lapply(mylist, powerFunction, exponent = 2)
```

Recap

Loops

for-loops	apply	map
Gets messy fast	Elegant and readable	Allows for better syntax and elegant code
Long code	Short code, based on functions	Elegant integration into tidyverse
Exam relevant	Exam relevant	Not exam relevant

Tutorial 1: A Function to Compute the Mean

Tutorial 1: A Function to Compute the Mean

Starting point: we should be aware of how the mean is defined:

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) = \frac{x_1 + x_2 + \cdots + x_n}{n}.$$

Tutorial 1: A Function to Compute the Mean

```
#####################################
# Mean Function:
# Computes the mean, given a
# numeric vector.

meaN <- function(x){

}
```

Tutorial 2: on slow and fast sloths

Tutorial 2: on slow and fast sloths

We can use loops to simulate natural processes over time. Write a program that calculates the populations of two kinds of sloths over time. At the beginning of year 1, there are 1000 slow sloths and 1 fast sloth. This one fast sloth is a new mutation that is genetically able to use roller blades. Not surprisingly, being fast gives it an advantage, as it can better escape from predators.

A slow sloth

A fast sloth in its natural element

Tutorial 2: on slow and fast sloths

Each year, each sloth has one offspring. There are no further mutations, so slow sloths beget slow sloths, and fast sloths beget fast sloths. Also, each year 40% of all slow sloths die each year, while only 30% of the fast sloths do.

So, at the beginning of year one there are 1000 slow sloths. Another 1000 slow sloths are born. But, 40% of these 2000 slow sloths die, leaving a total of 1200 at the end of year one. Meanwhile, in the same year, we begin with 1 fast sloth, 1 more is born, and 30% of these die, leaving 1.4.

Beginning of Year	Slow Sloths	Fast Sloths
1	1000	1
2	1200	1.4
3	1440	1.96

Enter the first year in which the fast sloths outnumber the slow sloths.

Tutorial 3: A loop function

Tutorial 3: A loop function

Be the function

```
appendsums <- function(lst){

    #Repeatedly append the sum of the current last three elements
    #of lst to lst.
}
```

Create a function that repeatedly appends the sum of the current last three elements of the vector lst to lst. Hint: use the `append` and `tail` functions. Your function should loop 25 times. To check if your function is correct, run:

```
sum_three = c(0, 1, 2)
appendsums(sum_three)

# Solution for testing:
sum_three[10] == 125
```

Q&A