**Control Structures – Introduction**

* Allow you to control the flow of execution of R
* Example
  + - If, else
      * Testing a condition
    - For
      * Execute a loop for a fixed number of times
    - While
      * Execute a loop **while** a condition is true
    - Repeat
      * Execute an infinite loop
    - Break
      * Break the execution of a loop
    - Next
      * Skip an interaction of a loop
    - Return
      * Exit a function

**Control structures – If-else**

If(<condition) {

## do something

}

Else {

## do something else

}

**Or:**

If(condition) {

## do something

}

Else if(condition) {

## something else

}

Else {

## something even else

}

* Example
  + - If(x>3) {
      * Y <- 10

}

Else {

Y <- 0

}

**Control structures – for loops**

* Example

For(i in 1:10) {

Print(i)

}

* For can be nested. However, be careful, as nesting beyond 2-3 levels is already hard to understand

**Control structures – while loops**

* Example

Count <- 0

While(count < 10) {

Print(count)

Count <- count +1

}

**Control structures – repeat, next and break**

* **Repeat: an infinite loop**

X0 <-1

Tol <- 1e – 8

Repeat {

X1 <- func()

If(abs(x1 – x0) < tol {

Break

} else {

X0 <- x1

}

}

* Repeat is dangerous as we don’t know whether it’ll stop
* **Next: skip the interation of a loop**

For(I in 1:100) {

If(I <= 20) {

Next

}

Else{

## do something here

}

}

* **Return: signals that a function should exit and return a given value**

**Your first R function**

**Functions**

* Are created by **function()**
* Class: “function”

F <- function(x) {

##something something

}

* Functions have named arguments, which have default values. To see their arguments, use **formals()**
* Can be matched positionally or by name. Example:

Mydata <- rnorm(100)

Sd(mydata)

Sd(x = mydata)

Sd(x = mydata, na.rm = FALSE)

Sd(na.rm = FALSE, x = mydata)

Sd(na.rm = FALSE, mydata)

Are all THE SAME.

* **Argument matching**

Args(lm)

**Defining a function**

F <- function(a, b = 1, c = 2, d = NULL) {

}

* “Lazy” evaluation

**The “…” argument**

* This indicates a variable number of argument that are usually passed on to other functions
* “…” is often used when extending another function and you don’t wanna copy the entire argument list of the original function

Myplot <- function(x, y, type = “1”, …) {

Plot(x, y, type = type, …)

}

Mean

Function(x, …)

UseMethod(“mean”)

Args(paste)

Function(…, sep = “ “, collapse = NULL)

Args(Cat)

Function(…, file = “”, sep = “ “, fill = FALSE, labels = NULL, append = FALSE)

* Any arguments appear after “…” on the list must be names **exactly**

**Scoping Rules – Symbol Binding**

* Search()
* R uses *lexical scoping* or *static scoping*

**Scoping Rules – R Scoping Rules**

* Other languages that support lexical scoping
  + - Scheme
    - Python
    - Perl
    - Common Lisp

**Coding Standards in R**

1. Always use a text files/text editor
2. Indent your code (at least 4 spaces)
3. Limit the width of your code (80 columns?)
4. Limit the length of individual functions

**Dates and Times in R**

* R has a special way of representing dates and times
  + - Dates are represented by the Date class
    - Times are represented by the POSIXct or the POSIXlt class
    - Dates are stores internally as the number of days since 1970-01-01
    - Times stores internally as the number of seconds since 1970-01-01
* **Example**

x <- as.Date("1970-01-01")

x

unclass(x)

* **Times:**

weekdays(x)

months(x)

quarters(x)

x <- Sys.time()

x

p <- as.POSIXlt(x)

p

names(unclass(p))

p$sec

datestring <- c("September 6, 2018 00:40", "May 14, 2019 12:00")

x <- strptime(datestring, "%B %d, %Y %H:%M")

x

class(x)