R Basic Programming

Branching with if

```
if ( logical_expression ) {
  expression_1
}
```

```
if (logical_expression) {
  expression_1
} else {
  expression_2
}
```

If logical_expression is TRUE then the first group of expression is executed and the second group of expression is not executed.

Conversely if logical_expression is FALSE then only the second group expression is executed.

If you type:

```
if (logical_expression) {
  expression_1
else {
  expression_2
}
```

then you get an error.

This is because R is believes the if statement is finished before it sees the else part.
That is, R treats the else as the start of a new command, but there is no command that starts with an else.

The followings are equivalent.

```
if (logical_expression_1){
  expression_1
} else {
    if (logical_expression_2){
      expression_2
    } else {
      expression_3
    }
}
```

```
if (logical_expression_1){
  expression_1
} elseif (logical_expression_2){
  expression_2
} else {
  expression_3
}
```

Example : root of quadratic

```
rm(list=ls())

a0 <- 1; a1 <- 6; a2 <- 7
```

```r
discrim <- a1^2 - 4*a2*a0

if (discrim > 0) {
  roots <- c( (-a1 + sqrt(a1^2 - 4*a2*a0))/(2*a2), (-a1 - sqrt(a1^2 - 4*a2*a0))/(2*a2) )
} else if (discrim == 0) {
  roots <- -a1/(2*a2)
} else {
  roots <- c()
}

show(roots)
```

```
## [1] -0.2265409 -0.6306019
```

Looping with for

```r
for (x in vector) {
  expression_1
  ...
}
```

where x is a simple variable.

Example : summing a vector

We have a built-in function sum() but for an illustrative purpose:

```r
(x_list <- seq(1, 9, by = 2))
```

```
## [1] 1 3 5 7 9
```

```r
sum_x <- 0
for (x in x_list) {
  sum_x <- sum_x + x
  cat("The current loop element is", x, "\n")
  cat("The cumulative total is", sum_x, "\n")
}
```

```
## The current loop element is 1
## The cumulative total is 1
## The current loop element is 3
## The cumulative total is 4
## The current loop element is 5
## The cumulative total is 9
## The current loop element is 7
## The cumulative total is 16
## The current loop element is 9
## The cumulative total is 25
```

- `cat()` : for concatenate
- `\n` : new line

Example : n factorial

```r
n <- 6
n_factorial <- 1
for (i in 1:n) {
  n_factorial <- n_factorial * i
}
show(n_factorial)
```

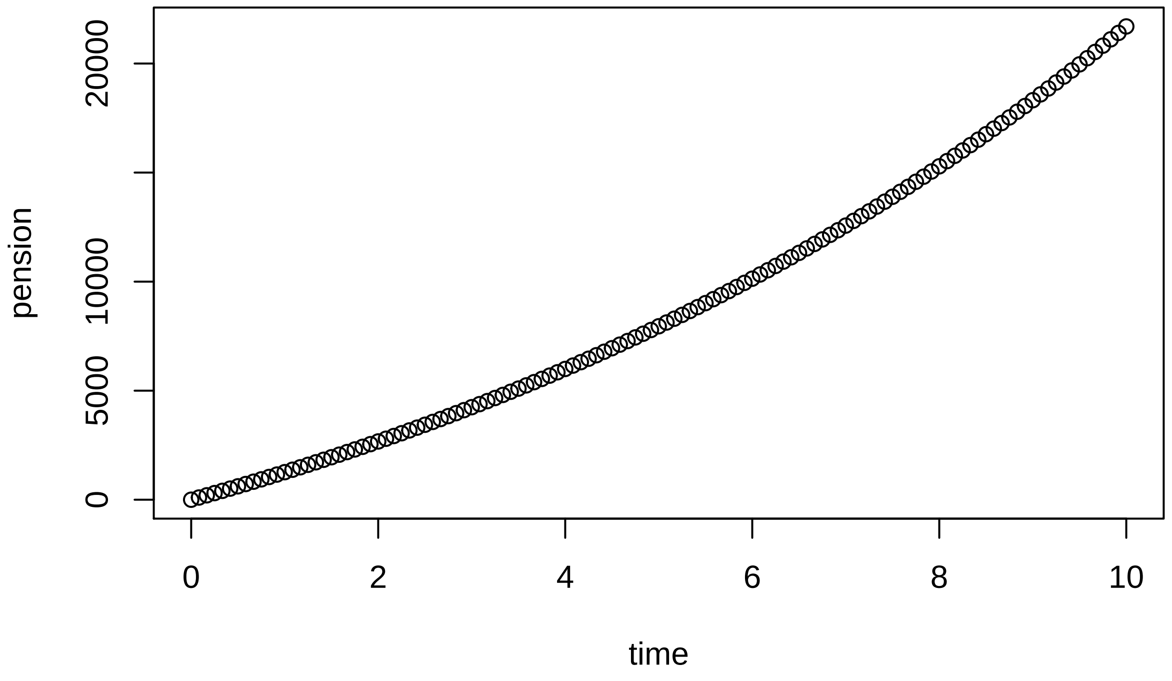
```
## [1] 720
```

Example : pension

```r
r <- 0.11              # Annual interest rate
term <- 10             # Forecast duration (in years)
period <- 1/12         # Time between payments (in years)
payments <- 100        # Amount deposited each period
# Calculations

n <- floor(term/period)  # Number of payments
pension <- 0
for (i in 1:n) {
  pension[i+1] <- pension[i]*(1 + r*period) + payments
}
time <- (0:n)*period

# Output
plot(time, pension)
```



Example : redimensioning

Program 1 and 2 produce the same result but 1 is faster.

```r
#Program1
n <- 100000
x <- rep(0, n)
for (i in 1:n) x[i] <- i
```

```r
#Program2
n <- 100000
x <- 1
for (i in 2:n) x[i] <- i
```

Program1 of preallocation is faster than Program2 of redimensioning.

Looping with while

```r
while (logical_expression) {
  expression_1
  ...
}
```

- logical_expression is evaluated first.
- If it is TRUE, then the expression in { } is executed.
- back to the start of the command.
- If logical_expression is FALSE, the loop stop.
- We can always rewrite a for loop as a while loop.

Example : Fibonacci numbers

```r
rm(list=ls())
F <- c(1, 1)
n <- 2
while (F[n] <= 100) {
  n <- n +1
  F[n] <- F[n-1] + F[n-2]
}
cat("The first Fibonacci number > 100 is F(", n, ") =", F[n], "\n")
```

```
## The first Fibonacci number > 100 is F( 12 ) = 144
```

Example : compound interest rate

```r
rm(list=ls())
r <- 0.11                  # Annual rate
period <- 1/12             # Time between repayments (in years)
debt_initial <- 1000       # Amount borrowed
repayments <- 12           # Amount repaid each period

# Calculations
time <- 0
debt <- debt_initial
while (debt > 0) {
  time <- time + period
  debt <- debt*(1 + r*period) - repayments
}

# Output
cat('Loan will be repaid in', time, 'years\n')
```

```
## Loan will be repaid in 13.25 years
```

Vector-based programming

Using vector operations is more efficient computationally.
Sum of the first n squares using loop:

```
n <- 100; S <- 0
for (i in 1:n)  S <- S + i^2
print(S)
```

```
## [1] 338350
```

Alternatively, using vector operations:

```
sum((1:n)^2)
```

```
## [1] 338350
```

The ifelse function performs elementwise conditional evaluation upon a vector.

- ifelse(test, A, B)
- test : logical expression
- returns a vector consist of
  - A : when element of test are true
  - B : when element of test are true

```
x <- c(-2, -1, 1, 2)
ifelse( x>0, "Positive", "Negative")
```

```
## [1] "Negative" "Negative" "Positive" "Positive"
```

pmin and pmax:
vectorised versions of the minimum and maximum

```
pmin( c(1, 2, 3), c(3, 2, 1), c(2, 2, 2))
```

```
## [1] 1 2 1
```

Program flow

```
x <- 3
for (i in 1:3) {
  show(x)
  if (x %% 2 == 0) {
    x <- x/2
  } else {
    x <- 3*x + 1
  }
}
```

```
## [1] 3
## [1] 10
## [1] 5
```

```r
show(x)
```

```
## [1] 16
```

Basic debugging : correcting errors

- To find an error or bug, you need to see how your variables change.
- include statements like `cat("var=", var, "\n")`
- dry run : using simple starting conditions for which you know what the answer should be.
- use short and simple versions of the final program
- Use graph and summary statistics.
- Careful use of indentation.

Example

```r
x <- 3
for (i in 1:3) {
  show(x)
  cat("i = ", i, "\n")
  if (x %% 2 == 0) {
    x <- x/2
  } else {
    x <- 3*x + 1
  }
}
```

```
## [1] 3
## i =  1
## [1] 10
## i =  2
## [1] 5
## i =  3
```

```r
show(x)
```

```
## [1] 16
```

Good programming habits

- Good programming is clear rather than clever.
- in practice, much more time is spent correcting and modifying codes than is ever spent writing them.
- Write comments.
- Variable name should be descriptive.
- Avoid using reserved names of functions
  - for example, t, c and q are all function names in R
  - check with exists() function
- Use black line to separate sections.