Chapter 4 Markdowns

By @TNTprizz80315.

Disclaimer: I assume that you've run through ENGG1100(C programming) before taking STAT2005, and you ought to have basic programming solving skills after that. I will only include the syntaxes involved without examples.

Programming in R

Declaring a function

Declaring a function called `factorial()`:

```r
factorial <- function(x) {
    if (any(x %% 1 != 0) | any(x < 0)) stop("Please use positive integers.")
    if (x == 0) return(1)
    else return(x*factorial(x - 1))
}
factorial(5)
```

Output:

```
[1] 120
```

Built-in functions

`return(x)` Self-explanatory.
Do not use `return x` or an error will occur.

`warning(msg)` Printout a warning message `msg`.

`stop(msg)` Throw out an error and terminate the program with message `msg`.

Scope of functions

In R, variables declared in a function are local variables, which are unique in every functions, despite having the same names.

Super assignment

We can declare a global variable inside a function by `<<-`.

For instance, you want to count the number of times the function is called:

```
factorial <- function(x) {
    if (any(x %% 1 != 0) | any(x < 0)) stop("Please use positive
integers.")
    if (!exists("ct")) ct <<- 1
    else ct <<- ct + 1
    if (x == 0) {
        print(ct)
        return(1)
    }
    else return(x*factorial(x - 1))
}
factorial(20)
```

Output:

```
[1] 21 # The program recursed for 21 times.
[1] 2.432902e+18
```

Logical Expression

For comparison between 2 objects:
`> < <= >= == !=`

For boolean comparison:
`||`(or) `&&`(and) `!`(not)

Comparing boolean vectors:
`&`(and) `|`(or) `!`(not)

```
a <- c(TRUE, FALSE, TRUE)
b <- c(TRUE, FALSE, FALSE)
a & b
a | b
!a
```

Output:

```
[1]  TRUE FALSE FALSE
[1]  TRUE FALSE  TRUE
[1] FALSE  TRUE FALSE
```

Control

`if` statement

```
if (statement) {
    # do if statement is TRUE
} else if (statement2) {
    # do if statement2 is TRUE
} else {
    # do otherwise
}
```

`switch` statement

```
switch(var,
    var1 = {
        do sth if var == var1
    },
    var2 = {
        do sth if var == var2
    },
    ...
)
```

Looping

`for` loop

```
j = c()
for (i in 1:10) {
    j <- c(j, i)
}
j
```

Output:

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

`while` loop

```
i <- 1
j = c()
while (i <= 10) {
    j <- c(j, i)
    i <- i + 1
}
j
```

Output:

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

`repeat` loop

```
i <- 1
j = c()
repeat {
    j <- c(j, i)
    i <- i + 1
    if (i > 10) break; # If you don't do this, infinite loop will
occur.
}
j
```

Output:

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

`next` statement inside a loop

You can directly skip to the next iteration using `next`.

```
j = c()
for (i in 1:10) {
    next # Do nothing for 10 times.
    j <- c(j, i) # This line is skipped for every iterations.
}
j
```

Output:

```
NULL
```

`break` statement inside a loop

You can terminate an iteration using `break`.

```r
j = c()
for (i in 1:10) {
    break # Quit the loop immediately.
    j <- c(j, i) # This line is skipped.
}
j
```

Output:

```
NULL
```