Programming structures

> Created by Woo Hyun (Ray) Hwang

Programming structures

for loop

To repeat certain procedures, we use the for loop.

`for(name in vector) {commands}` : variable name will execute all commands until the end of the vector

```
#The Fibonacci seq.
Fibonacci <- numeric(12)
Fibonacci[1] <- 1
Fibonacci[2] <- 1
for(i in 3:12) Fibonacci[i] <- Fibonacci[i-2]+Fibonacci[i-1]
Fibonacci
```

Making a string loop:

```
for(i in 1:3) {
  cat("The value of i is", i, "\n") #cat function connects the elements
}
```

In a character vector:

```
for(word in c("hello", "new", "world")) {
  cat("The current word is", word, ".", "\n")
}
```

Example using lists:

```
loop_list <- list(a=c(1, 2, 3), b=c("a", "b", "c", "c"))
for(item in loop_list) {
  cat("item: \n length:", length(item),
      "\n class:", class(item))
}
```

Change the global var. to make changes to a for loop:

```
s <- 0
for(i in 1:100) {
```

```
  s <- s+i
}
s
```

The random walk:

```
set.seed(123)
x <- numeric(1000)
for(t in 1:(length(x)-1)) {
  x[t+1] <- x[t] + rnorm(1, 0, 0.1)
}
plot(x, type="s", main="Random walk", xlab=t)
```

Calulating sum of i^4 from 1 to 10:

```
temp <- 0
for(i in 1:10) {
  temp <- temp + i^4
}
temp
#To see steps, add a print function
temp <- 0
for(i in 1:10) {
  temp <- temp + i^4
  print(temp)
}
temp
```

`break` : when the `if` statement is fulfilled, stop the for loop completely:

```
for(i in 1:5){
  if(i == 3) break
  cat("message", i, "\n")
}
```

`next` : when the `if` statement is fulfilled, skip and move on:

```
for(i in 1:5){
  if(i == 3) next
  cat("message", i, "\n")
}
```

`while()` statement

`while(condition) {statement}` : unknown or vague pattern, so do until the condition and stop when condition becomes false.

```
#The Fibonacci seq up to max. 300
Fib1 <- 1; Fib2 <- 1
```

```
Fibonacci <- c(Fib1, Fib2)
while (Fib2 < 300) {
  Fibonacci <- c(Fibonacci, Fib2)
  oldFib2 <- Fib2
  Fib2 <- Fib1 + Fib2
  Fib1 <- oldFib2
}
Fibonacci
```

Sum of i^4 until i^4 < 100000:

```
temp <- 0
i <- 1
while(i^4 < 100000) {
  temp <- temp + i^4
  i <- i+1
}
c(i-1, temp)
```

`while` is the same as `repeat-break`:

```
temp <- 0
i <- 1
repeat{
  if(i^4 >= 100000) break
  temp <- temp + i^4
  i <- i+1
}
c(i-1, temp)
```

`if` Conditionals

`if(condition) {commands when TRUE}`: execute command when condition is TRUE

`if(condition) {commands when TRUE} else{commands when FALSE}`

```
#simple example
x <- 5
z <- (if(x > 3) y <- 2*x else y <- 3*x)
z
```

Creating a `corplot`:

```
corplot <- function(x, y, plotit) {
  if(plotit==TRUE) {plot(x, y)}
  cor(x, y)
}
x <- c(3,6,9,11,14,16,18)
y <- c(2,4,8,14,18,24,28)
corplot(x, y, TRUE) #will produce the plot
corplot(x, y, FALSE) #only shows the correlation
```

Getting all the prime numbers (Eratosthenes' sieve):

```
eratosthenes <- function(n) {
  if(n>=2) {
    sieve <- seq(2, n)
    primes <- c()
    for(i in seq(2, n)){
      if(any(sieve==i)) {
        primes <- c(primes, i)
        sieve <- c(sieve[!(sieve %% i ==0)], 1)
      }
    }
    return(primes)
  } else{
    stop("Input values of n should be at least 2.")
  }
}
eratosthenes(50)
```

User defined functions

`f <- function(arguments) {statements}`

```
#Calculation of amount of money with interest i after n years
amt <- function(n, r, i) {
  r*((1+i)^n-1)/i
}
amt(10, 400, 0.02) #400, 10 years, 0.02 interest rate
```

All variables within the function are local variables, used only inside the function itself. To make it a global variable, use the operator `<<-`.

Applications on eratosthenes' sieve, the `sort()` function, Blackjack, Slot machines are found on the PPT, and needs reviewing.