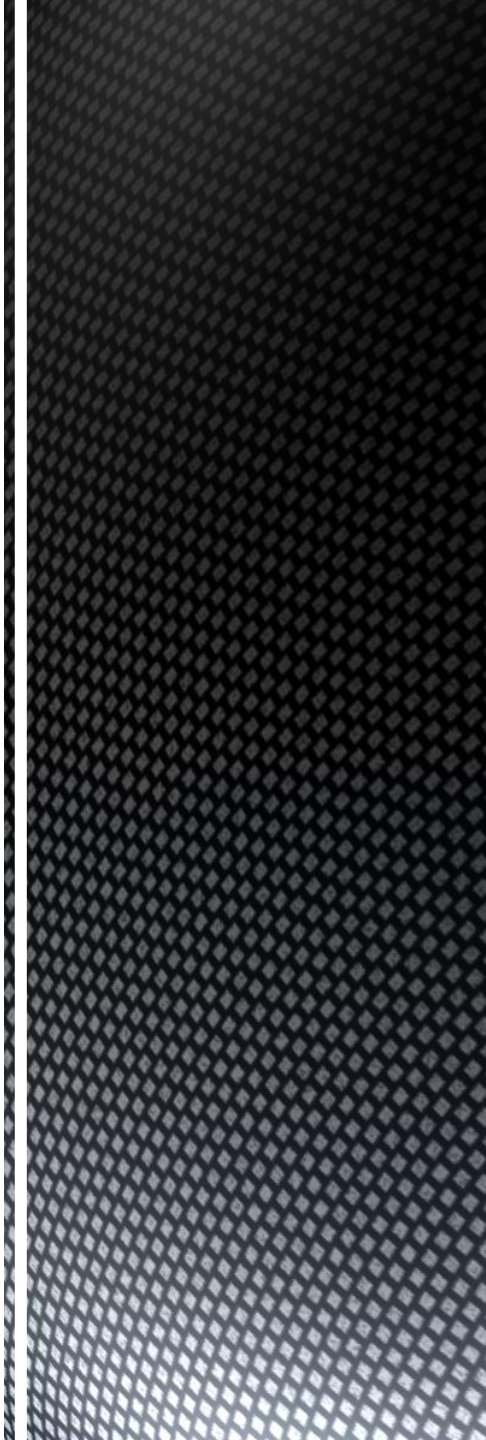
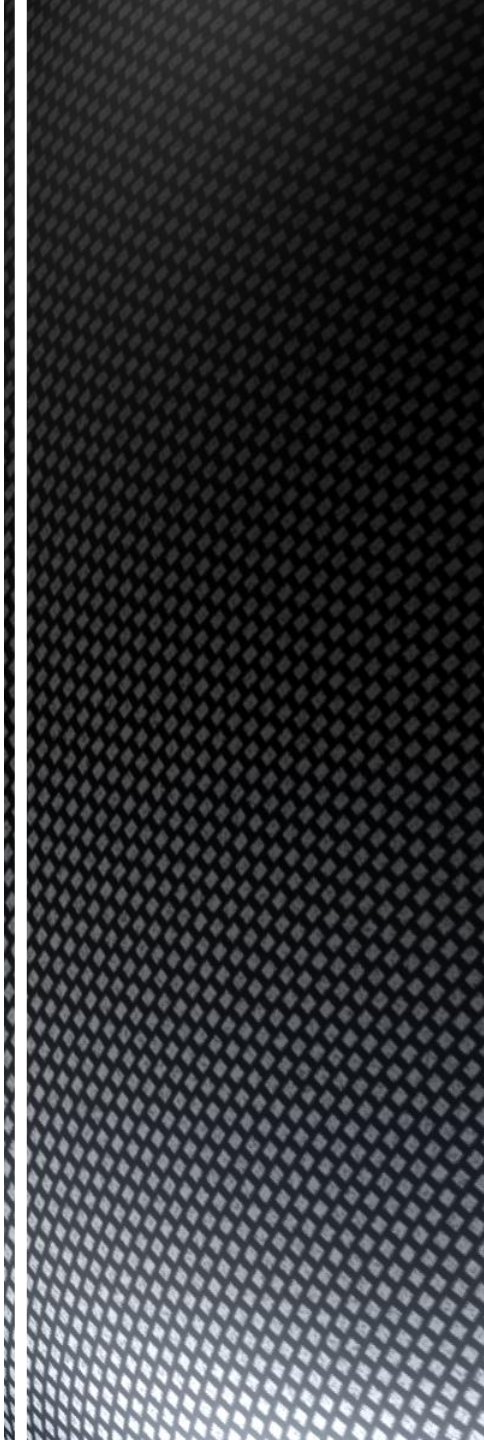
R Programming



Basic R Programming



Pendahuluan

- Keunggulan R dibandingkan softwares dengan *pull-down menu* adalah kemampuannya melakukan pemrograman atas suatu rangkaian analisis akan dieksekusi secara berurutan
- R memiliki beberapa features yang membuat programming menjadi lebih mudah untuk semua orang.
- Setelah struktur kontrol pemrograman R dasar dipahami, pengguna dapat menggunakan bahasa R sebagai *environment* yang sangat *powerful* untuk melakukan analisis kompleks dari hampir semua jenis data.

R programming

- Control structure:
  - Struktur percabangan
  - Struktur perulangan
- Apply, sapply, dan lapply
- Fungsi

Control Structures

- R memiliki beberapa control structures yang tidak berbeda dengan bahasa C.
- Control structures di R mencakup:
  - if, else
  - switch
  - for
  - while
  - repeat
  - break
  - next

STRUKTUR PEMILIHAN / PERCABANGAN

- Secara analogi dalam kehidupan sehari-hari percabangan dapat kita lihat pada saat kita berjalan dimana ketika tiba di persimpangan kita hendak memilih jalan manakah yang akan kita tuju, apakah belok ke kanan, ke kiri, atau lurus ?

- Pertimbangan :

✓ Jarak yang dilalui

✓ Kondisi jalan

✓ Waktu tempuh

Struktur Pemilihan

Pada struktur ini, jika kondisi terpenuhi maka salah satu aksi akan dilaksanakan dan aksi yang ke dua diabaikan.

Kondisi adalah persyaratan yang dapat dinilai benar atau salah sehingga akan memunculkan 'aksi' yang berbeda dengan 'kondisi' yang berbeda.

if, else statement

Berikut adalah sintaks untuk if,else statement:

```
if (condition) {
    # do something
} else {
    # do something else
}
```

Contoh:

```
x <- 1:15
if (sample(x, 1) <= 10) {
    print("x is less than 10")
} else {
    print("x is greater than 10")
}
```

if, else statement hanya untuk vector berukuran 1. Berikut adalah contoh dari if-else:

Contoh

```
> if(1==0) {
+   print(1)
+   } else {
+     print(2)
+   }
[1] 2

> w = 3
> if( w < 5 ) {
+   d=2
+   } else {
+     d=10
+     }
> d
[1] 2
```

```
> x <- 1:15
> if (sample(x,1)<10){
+   print("x kurang dari 10")
+   }else{
+     print("x lebih dari atau sama dengan 10")
+   }
[1] "x kurang dari 10“


> price = 110000
> if (price >= 100000) net.price <- price*0.9
> net.price
[1] 99000
```

Ifelse statement

- Sebagai alternatif, kita bisa menggunakan *statement* **ifelse** → lebih rapi dan ringkas
- Ifelse dapat digunakan untuk object tipe vector dengan panjang n.
- **Penggunaan**

```
ifelse(test, yes, no)
```

- **Arguments**

test	**an object which can be coerced to logical mode.**
yes	return values for true elements of test.
no	return values for false elements of test.

Contoh

```
> x <- 1:10 # Creates sample data
> ifelse(x<5 | x>8, x, 0)
 [1]  1  2  3  4  0  0  0  0  9 10
>
> ifelse(x > 5, "high", "low")
 [1] "low"  "low"  "low"  "low"  "low"  "high" "high"
"high" "high" "high"

x <- readline(prompt = "Masukkan nilai x : ")
Masukkan nilai x : 4
x <- as.numeric(x)
> ifelse(x>=0,sqrt(x),NA)
[1] 2
```

switch statement

- Untuk memilih lebih dari 2 pilihan, kita dapat menggunakan fungsi switch() dimana input nya adalah berupa expresi dari input yang diinginkan serta pilihan yang tersedia.
- Sintaks:

```
switch(EXPR, ...)
```

- Jika EXPR terisi integer, maka itu akan digunakan sebagai urutan dari pilihan yang diberikan.

- Sedangkan jika input (EXPR) adalah berupa string, maka akan dicocokkan dengan nama pilihan yang tersedia

contoh

```
> x <- switch(3,
+             "first",
+             "second",
+             "third",
+             "fourth"
+ )
> x
[1] "third“
```

EXPR berupa integer

```
> inpt = "mean"
> x <- 1:10
> switch(inpt,
+           mean = mean(x),
+           median = median(x))
[1] 5.5
```

EXPR berupa string

```
> x <- rnorm(70,10)
> print("Pilih: 1. Hitung mean, 2. Hitung median")
[1] "Pilih: 1. Hitung mean, 2. Hitung median"
> inpt <- as.numeric(readline("Pilihan: "))
Pilihan: 2
> switch(inpt,
+        mean = mean(x),
+        median = median(x))
[1] 9.893732


> ##atau
> print("Pilih: 'mean' atau 'median'")
[1] "Pilih: 'mean' atau 'median'"
> inpt <- readline("Pilihan: ")
Pilihan: median
> switch(inpt,
+         mean = mean(x),
+         median = median(x))
[1] 9.893732
```

Struktur Pengulangan

Digunakan untuk program yang pernyataannya akan dieksekusi berulang-ulang. Instruksi dikerjakan selama memenuhi suatu kondisi tertentu. Jika syarat (kondisi) masih terpenuhi maka pernyataan (aksi) akan terus dilakukan secara berulang.

- Struktur loop yang paling umum digunakan dalam R adalah for, while, dan apply.
- Selain itu ada juga perintah repeat, namun jarang digunakan.
- Fungsi break digunakan untuk keluar dari *loop*, dan selanjutnya menghentikan pengolahan iterasi saat ini dan kemajuan indeks perulangan.

For loop

- Loop ini dikendalikan oleh sebuah vektor perulangan. Sedangkan jumlah iterasi *loop* didefinisikan dengan jumlah nilai yang disimpan dalam vektor perulangan dan akan diproses dalam urutan yang sama seperti yang disimpan dalam vektor perulangan.
- Penggunaan syntax:

for (*var* in *seq*) *expr*

```
for (vector counter)
          {Statements}
```

For loop bekerja pada variabel yang iterable dan memberikan nilai berturut-turut sampai akhir urutan.

contoh

```
for (i in 1:10) {
    print(i)
}
x <- c("apples", "oranges", "bananas", "strawberries")

for (i in x) {
    print(x[i])
}

for (i in 1:4) {
    print(x[i])
}

for (i in seq(x)) {
    print(x[i])
}

for (i in 1:4) print(x[i])
```

Contoh:

```
> h <- seq(from=1, to=10)
>s <- c() ## definisikan objek “s”

> ## lakukan iterasi sebanyak 10x
> for(i in 1:10)
+   {
+     ## memasukkan nilai hasil ke objek s
+       s[i] <- h[i] * 10
+       }
> s
 [1]  10  20  30  40  50  60  70  80  90 100
```

Contoh: Kita dapat juga memasukkan nilai iterasi per baris dari sebuah matrix seperti contoh dibawah ini:

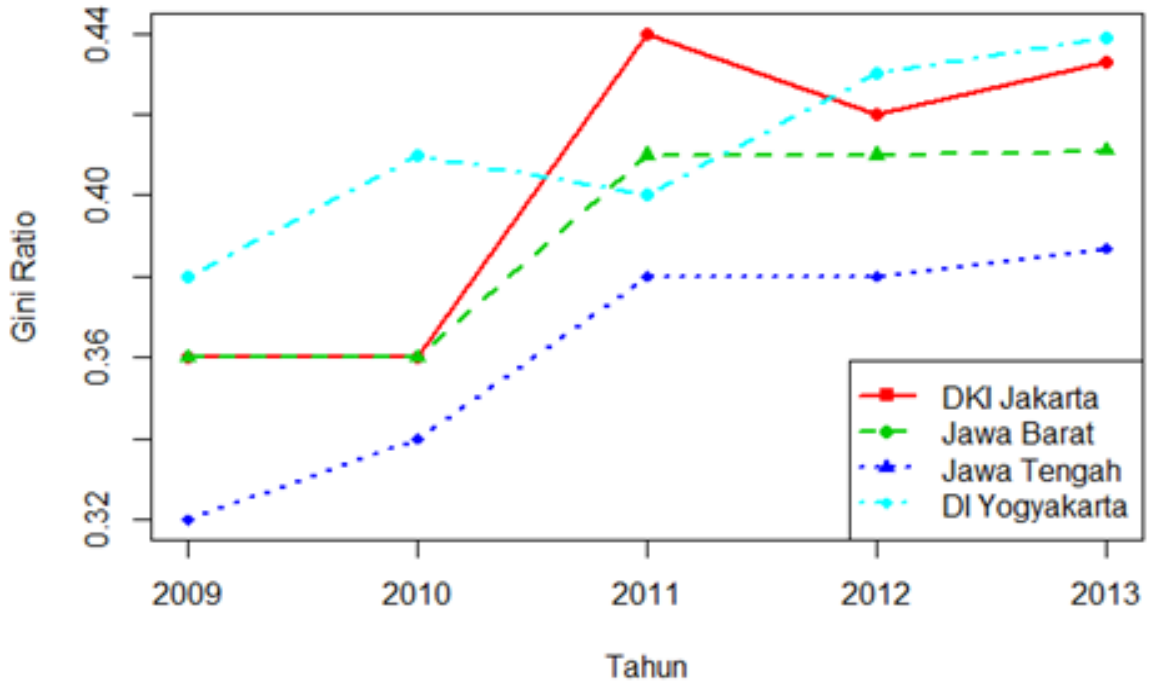
```
> sqr <- seq(1, 10, by=2)
> sqr
[1] 1 3 5 7 9
> ## definisikan vector hasil
> res <- NULL
> ## definisikan matrik untuk hasil
> resMat <- matrix(NA,5,2)
> for (i in 1:5) {
+   res [i] <- sqr[i]^2
+   resMat[i,] <- c(i, sqr[i]^2)
+   }
> resMat
[,1] [,2]
[1,] 1 1
[2,] 2 9
[3,] 3 25
[4,] 4 49
[5,] 5 81
> res
[1] 1 9 25 49 81
```

Contoh: aplikasi struktur perulangan dalam membuat grafik

```
> ## Data Gini Ratio ##
> Tahun <- rep(c(2009, 2010, 2011, 2012, 2013), times=4)
> Gini <- c(0.36,0.36,0.44,0.42,0.433,0.36,0.36,0.41,0.41,
+               0.411, 0.32,0.34,0.38,0.38,0.387,0.38,0.41,
+               0.40,0.43,0.439)
> Prov <- rep(c("DKI Jkt","Jawa Barat","Jawa Tengah","DIY"),
+               each=5)
> DataGini <- data.frame(Prov, Tahun, Gini)
> head(DataGini)
  Prov Tahun Gini
  1 DKI Jakarta 2009 0.360
  2 DKI Jakarta 2010 0.360
  3 DKI Jakarta 2011 0.440
  4 DKI Jakarta 2012 0.420
  5 DKI Jakarta 2013 0.433
  6 Jawa Barat 2009 0.360

> # mendapatkan rentang nilai untuk sumbu x dan y
> xrange <- range (DataGini$Tahun)
> yrange <- range(DataGini$Gini)
> # Membuat plot kosong ##
> plot(xrange, yrange, type="n", xlab="Tahun",
+       ylab="Gini Ratio" )
```

```
> Prov <- unique(DataGini$Prov)
> Prov
[1] DKI Jakarta Jawa Barat Jawa Tengah DI Yogyakarta
Levels: DI Yogyakarta DKI Jakarta Jawa Barat Jawa Tengah
> ## mengisi plot yang telah dibuat dengan garis dan titik
> for(i in 1:length(Prov)){
+               datai <- DataGini[DataGini$Prov == Prov[i],]
+               lines(datai$Tahun,datai$Gini,col= i+1,lty=i,lwd=2,
+               pch=15+i)
+               points(datai$Tahun, datai$Gini, col= i+1, pch=15+i)
+               }
> legend("bottomright", legend= Prov, col=2:6, pch=15:21, lty=1:5,
lwd=2)
```



Nested loop

- Nested loop → struktur perulangan di dalam struktur perulangan lain.
- Syntax:

```
m <- matrix(1:10, 2)
for (i in seq(nrow(m))) {
    for (j in seq(ncol(m))) {
        print(m[i, j])
    }
}
```

While

- Statemen while sama dengan for loop, hanya iterasinya di control oleh sebuah kondisi.
- Syntax:

```
while(condition) statements
```

while (*cond*) *expr*

- Be sure there is a way to exit out of a while loop.
- Avoid indefinite loop

e.g.

```
i <- 1
while (i < 10) {
    print(i)
    i <- i + 1
}
```

contoh

```
> i <- 1 ##inisialisasi
> while (i<6){
+   print(i)
+   i<-i+1
+ }
   [1] 1
   [1] 2
   [1] 3
   [1] 4
   [1] 5

> z <- 0
> while(z < 10) { ## print selama z < 10
+   z <- z + 2 # kenaikan 2
+   print(z)
+ }
   [1] 2
   [1] 4
   [1] 6
   [1] 8
   [1] 10
```

repeat statement

- Untuk repeat, kondisi diberikan di akhir statemen. Sintaks:

```
repeat
{statements...if(condition){break}}
```

```
repeat {
    # simulations; generate some value have an expectation if within some range,
    # then exit the loop
    if ((value - expectation) <= threshold) {
        break
    }
}
```

- Untuk menghentikan iterasi, digunakan perintah *break*
- Perbedaan dengan while adalah pada repeat minimal 1 kali iterasi dijalankan

```
> sum <- 1 ##inisialisasi
> repeat
+   {
+     sum <- sum + 2;
+     print(sum);
+     if (sum > 11)
+       break;
+     }
[1] 3
[1] 5
[1] 7
[1] 9
[1] 11
[1] 13
```

Next

- Jika *break* adalah untuk menghentikan iterasi, maka *next* adalah perintah untuk loncat (skip) ke iterasi berikutnya.

```
for (i in 1:20) {
    if (i%%2 == 1) {
        next
    } else {
        print(i)
    }
}
```

- This loop will only print even numbers and skip over odd numbers.
- We can use other functions that will help us avoid these types of slow control flows (mostly the while and for loops).

Vectorization

- Dalam R, faktanya adalah kita jarang menggunakan control structure baik percabangan atau pengulangan → Thank's to Vectorization
- Vektorisasi menjadikan proses perulangan implisit dalam ekpresi.
- Misal kita menjumlahkan vektor x dan y menjadi z:

  ```
  > z <- x + y
  ```

  Dalam R penjumlahan ini tidak perlu dilakukan looping per elemen seperti berikut:

  ```
  > z <- numeric(length(x))
  > for (i in 1:length(z)) z[i] <- x[i] + y[i]
  ```

- Sedangkan untuk percabangan (if ... else) bisa dihindari dengan penggunaan logical indexing;

  ```
  > y[x == b] <- 0
  > y[x != b] <- 1
  ```

apply

- Selain dari perintah *looping* diatas, R memiliki fungsi/command yang dapat melakukan *looping* secara implisit dan cepat yaitu keluarga fungsi apply.

- Fungsi ini digunakan untuk input data dengan 2 dimensi.

- Fungsi ini merupakan salah satu keunggulan R, sehingga jika memerlukan perulangan, usahakan gunakan apply function, dibandingkan loop seperti for-loop atau while.

- Syntax:

```
apply(X, MARGIN, FUN, ARGs)
```

dimana

X: 2 dimensi array, matriks atau data.frame,
MARGIN : 1 untuk baris dan 2 untuk kolom dan
, c(1,2) untuk keduanya;
FUN : fungsi yang ingin dijalankan;
ARGs : argument yang diperlukan untuk fungsi yang telah didefinisikan.

Contoh

```
> #data nilai Kalkulus dan Alpro mahasiswa
> nilai<-data.frame(Nama=c("Andi","Bagus","Charlie",
+                          "Dani","Erik"),
+                    Kalk=c(60,75,72,68,80),
+                  Alpro=c(70,78,82,70,78),
+                    row.names = 1)
> #tampilkan rata-rata nilai per mahasiswa
> apply(nilai,1,mean)
   Andi   Bagus Charlie    Dani    Erik
   65.0    76.5    77.0    69.0    79.0
> #tampilkan rata-rata nilai per mata kuliah
> apply(nilai,2,mean)
 Kalk Alpro
 71.0  75.6
```

tapply

- Jika dataset kita bisa dibagi menjadi beberapa kategori, dan kita ingin menerapkan suatu fungsi pada masing-masing kelompok data tersebut, maka kita bisa menggunakan *tapply***.**
- Penggunaan syntax:

```
tapply(X, index, FUN)
```

Dimana

- X adalah objek R, biasanya vektor
- Index adalah variabel kategorik yang membagi data
- FUN adalah fungsi yang ingin diterapkan

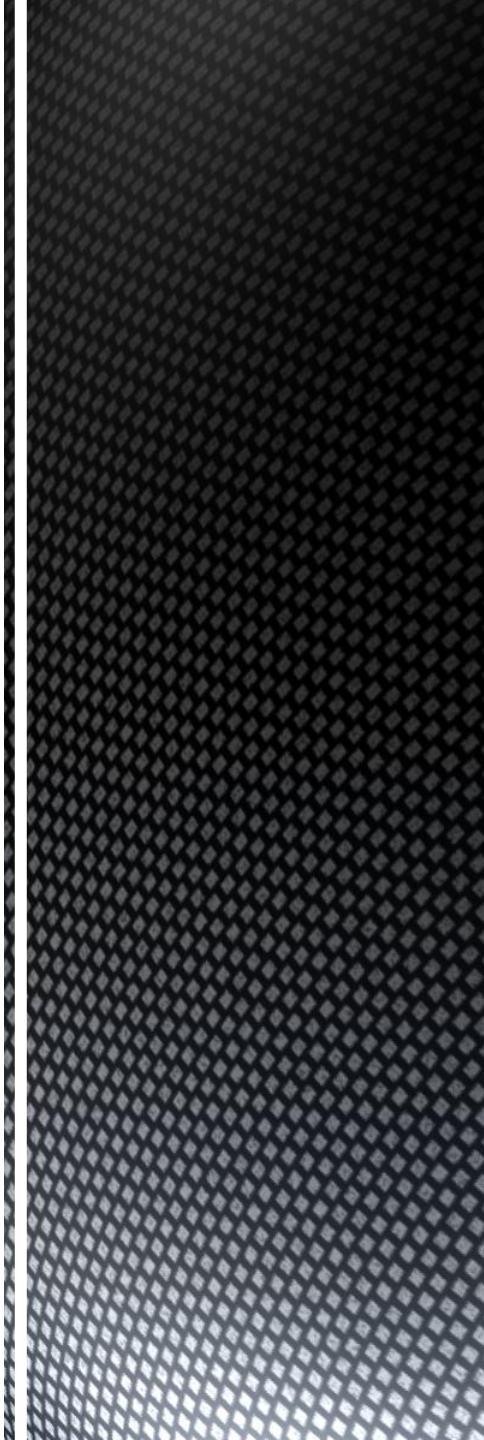
contoh

```
> ## generate data for medical example
> medical.ex <- data.frame(patient = 1:100,
+                          age = rnorm(100,
+                         mean = 60, sd = 12),
+                          treatment = gl(2, 50,
+                          labels=c("Treatment","Control")))
> summary(medical.ex)
    patient            age            treatment
 Min.   :  1.00   Min.   :33.84   Treatment:50
 1st Qu.: 25.75   1st Qu.:52.30   Control  :50
 Median : 50.50   Median :59.53
 Mean   : 50.50   Mean   :60.45
 3rd Qu.: 75.25   3rd Qu.:67.30
 Max.   :100.00   Max.   :92.13
> tapply(medical.ex$age, medical.ex$treatment, mean)
Treatment   Control
 60.00739  60.88739
```

- *tapply* hanya dapat digunakan pada vector, jika ingin melakukan hal yang sama ke matris maka dapat menggunakan fungsi aggregate:

```
> aggregate(medical.ex[,-3], list(medical.ex$treatment),
+           mean)
    Group.1 patient      age
1 Treatment    25.5 58.42530
2   Control    75.5 59.53114
```

Developing Function and Packages



- Meskipun R sudah menyediakan hampir semua package namun terkadang kita membutuhkan melakukan programming dengan R.

- Untuk kegiatan yang dilakukan secara berulang-ulang atau sangat kompleks, maka lebih baik kita membuat sebuah fungsi (*function*) yang dapat kita panggil/gunakan sehingga pekerjaan kita akan lebih efektif.

- Struktur fungsi R:

```
name <- function(arg1, arg2,... )
{
expression
}
```

- Arg1, Arg2... adalah argumen yang akan diberikan kepada ekspresi berikutnya.
- Untuk mengembalikan hasil perhitungan dari ekspresi yang diberikan gunakan perintah return().

Komponen dasar dari fungsi

- body() → kode yang ada di dalam fungsi.
- formals() → "formal" argument list, yang mengatur bagaimana fungsi tersebut dipanggil.
- `environment()`` → menentukan cakupan variabel yang ada dalam fungsi.
- args() → list arguments.

- Nama Argumen dapat kita definisikan saat input ataupun tidak.
- Perbedaannya adalah ketika didefinikan (misalkan, arg=1, arg2= 2) maka urutan peletakan tidak bermasalah, sedangkan jika kita tidak mendefinikan pada input maka urutan input disesuaikan dengan urutan arg yang telah didefinisikan.

Contoh:

```
myfun <- function(S, F)
{
    data <- read.table(F)
    plot(data$V1, data$V2, type="l")
    title(S)
}
```

- Agar bisa dijalankan, suatu fungsi harus di-*load* ke dalam memori terlebih dahulu. Terdapat beberapa cara:
  - The lines of the function can be typed directly on the keyboard, like any other command, or copied and pasted from an editor. → diketik langsung, atau copy-paste
  - If the function has been saved in a text file, it can be loaded with source() → gunakan perintah source()
  - If the user wants some function to be loaded each time where R starts, they can be saved in a workspace → simpan di workspace
  - Another possibility is to configure the file ".Rprofile" → advanced
  - Create a package → buat package R

Menghasilkan apa fungsi dibawah ini?

```
first <- function(x, y) {
     z <- x + y
     return(z)
 }
 add <- function(a, b) {
     return(a + b)
 }
 vector <- c(3, 4, 5, 6)

 sapply(vector, add, 1)
```

```
x <- 5
f <- function() {
    y <- 10
    c(x = x, y = y)
}
f()
```

Menghasilkan apa fungsi dibawah ini?

```
x <- 5
g <- function() {
    x <- 20
    y <- 10
    c(x = x, y = y)
}
g()
```

```
x <- 5
h <- function() {
    y <- 10
    i <- function() {
        z <- 20
        c(x = x, y = y, z = z)
    }
    i()
}
h()
```

Contoh:
Fungsi menampilkan deret Fibonacci

```
> fibo <- function(n) {
+ x <- c(0,1)
+ while (length(x) < n) {
+ position <- length(x)
+ new <- x[position] + x[position-1]
+ x <- c(x,new)
+ }
+ return(x)
+ }
> fibo(10)
[1] 0 1 1 2 3 5 8 13 21 34
```

Contoh

- Fungsi untuk memilih tipe statistik (dengan fungsi switch())

```
> exp1 <- function(x, type) {
+ switch(type,
+ kuadrat =x^2,
+ akar = sqrt(x),
+ exp =exp(x)
+ )
+ }
> x <- 1:5
> exp1 (x, "kuadrat" )
[1] 1 4 9 16 25
> exp1 (x, "akar" )
[1] 1.000000 1.414214 1.732051
2.000000 2.236068
```

LATIHAN

- Dari file **data_karyawan**, hitunglah mean dan standar deviasi dari variabel gaji per bulan menggunakan looping.

- Buatlah fungsi untuk menghitung mean dan standar deviasi.

- Bangkitkan data berdistribusi normal dengan mean 10 standar deviasi 3 sebanyak n=20. Kemudian aplikasikan fungsi yang dibuat untuk menghitung mean dan standar deviasi data bangkitan tersebut.