**Nama** : **Alfianri Manihuruk**

**NIM** : **120450088**

**Matkul** : **Komputasi Statistik**

# Pengembangan Fungsi dalam R

## Membuat Fungsi

```
func_name <- function (argument) {

statement

}
```

1. function: Fungsi dasar dalam membuat suatu fungsi
2. statement yang berada di dalam kurung kurawal, bisa diabaikan jika statement berbentuk satu baris (kode satu baris)
3. func_name: Nama dari fungsi
4. Gunakan fungsi return () agar fungsi memberikan suatu hasil.

Contoh:

```
# Membuat Fungsi
contoh= function()" hello world "
contoh

op<- function(val) return(val^3+2)
op(2)

op() # Akan menghasilkan eror karena nilai tidak dimasukkan
```

Akan menghasilkan out sebagai berikut.

```
> contoh= function()" hello world "
> contoh
function()" hello world "
> op<- function(val) return(val^3+2)
> op(2)
[1] 10
> op() # Akan menghasilkan eror karena nilai tidak dimasukkan
Error in op() : argument "val" is missing, with no default
```

## Default Argument

```
func_name <- function (argument=val) {
statement
}
```

1. Di dalam mendeklarasikan suatu fungsi, argumen bisa dibuat sedemikian sehingga nilainya default
2. Jika dalam pemanggilan fungsi tidak ada deklarasi untuk variabel, maka variabel yang akan digunakan adalah variabel default

Contoh:

```
# Default Argument
contoh2<- function(x=8, y=8){
  return(x+y)
}
contoh2(10,2)
contoh2()
```

Dengan hasil sebagai berikut

```
> contoh2(10,2)
[1] 12
> contoh2()
[1] 16
>
```

## Fungsi Rekursi

```
func_name <- function (argument=val) {
   func_name(val)
}
```

1. **Rekursi: Fungsi yang memanggil dirinya sendiri.**
2. **Harus berhati-hati agar fungsi yang berada di dalam fungsi tersebut tidak memunculkan output error**

**Contoh fungsi rekrusi**

```
# Fungsi Rekursi

rec<-function(n){
  rec(n)
}

jum <- function(n){
  if(n==1){
    return(1)
  }else{
    return(n+jum(n-1))
  }
}

jum()
```

```
> jum(10)
[1] 55
> jum (10)
[1] 55
> jum(2)
[1] 3
> jum(6)
[1] 21
>
```

**Latihan Fungsi**

1. **Buatlah fungsi untuk menghitung nilai n faktorial, diberikan n bilangan asli, dengan argumen n**

```
# 1. membuat faktorial
faktorial <- function(n){
  if(n==1){
    return(1)
  }else{
    return(n+jum(n-1))
  }
}

faktorial()
```

```
> faktorial(10)
[1] 55
> faktorial(21)
[1] 231
>
```

2. **Buatlah fungsi untuk menghitung Kombinasi**

```
# 2.membuat menghitung kombinasi
kom <- function(a=19,b=3){
  return (jum(a)/jum(b)+jum(a-b))
}

kom()
```

```
> kom()
[1] 167.6667
> kom(12,4)
[1] 43.8
>
```

**3. Buatlah fungsi root (a, b, c)**

```
# 3. membuat menghitung akar
root <- function(a,b,c){
  d=b^2-4*a*c
  if (d<0){
    return("nilai eror")
  } else{
    x1 <-(-b+sqrt(d)/(2*a))
    x2 <-(-b-sqrt(d)/(2*a))
    return(c(x1, x2))
  }
}
root(1, 2, 0)
```

```
> root(1, 2, 0)
[1] -1 -3
> root(1,5,0)
[1] -2.5 -7.5
> root(1,2,4)
[1] "nilai eror"
>
```

## Pemrograman berbasis objek di R

1. R telah mengimplementasi pemrograman berorientasi objek
2. Semua dalam R adalah objek
3. Implementasi objek dalam R berbeda dengan pemrograman berorientasi objek lain seperti: C++, python, dll
4. R memiliki tiga sistem kelas, S3, S4, dan kelas referens

**-Kelas S3**

1. Kelas yang sederhana dan primitive
2. Kelas ini yang paling popular dalam pemrograman R.
3. Tidak ada definisi formal untuk kelas ini
4. Objek dari kelas ini dapat dibuat dengan menambahkan atribut kelas ke dalam objek tersebut
5. Suatu list dengan atribut kelasnya diberikan suatu nama kelas, adalah objek kelas S
   Contoh:

```
# Pemograman Berbasis Objek
# kelas 3

m<- list(nama="andi",umur=19, gpa=3.2)
class(m)<- "MAHASISWA ITERA"
m
```

```
> m
$nama
[1] "andi"

$umur
[1] 19

$gpa
[1] 3.2

attr(,"class")
[1] "MAHASISWA ITERA"
>
```

```
b<- list(x=runif(10), y=runif(10,1,2))
class(b)<- "bilangan acak"

b
```

```
> b<- list(x=runif(3), y=runif(3,1,2))
> class(b)<- "bilangan acak"
> b
$x
[1] 0.4845778 0.8215595 0.9021778

$y
[1] 1.057802 1.765654 1.486466

attr(,"class")
[1] "bilangan acak"
>
```

## Konstruktor

1. Langkah sederhana dalam membuat objek dari suatu kelas sebelumnya sangat tidak dianjurkan karena nilai-nilainya mungkin berupa sesuatu yang tidak tepat.
2. Fungsi konstruktor dibutuhkan untuk mengecek kesesuaian nilai-nilai pada objek
3. Contoh: GPA haruslah berada pada rentang [0,4]
4. Umur (mungkin) haruslah kurang dari 25, dan sebagainya.

Contoh:

```
# Konstruktor
mhs<- function(n,u,g){
  if(g > 4 || g < 0){
    stop("gpa harus berada pada rentang 0 - 4")
}

  mahasiswa <- list(nama= n, umur= u, gpa= g)
  class(mahasiswa)<- "mahasiswa ITERA"
  mahasiswa
}

mhs <- mhs("jakob", 19, 3)
```

```
> mhs("inggrid", 24, 4.5)
Error in mhs("inggrid", 24, 4.5) : could not find function "mhs"
>
```

Error karena nilainya di luar batas yang telah di tentukan

## Fungsi Generik

1. Merupakan suatu method dari suatu class objek dalam R
2. Fungsi generic bertindak untuk beralih memilih fungsi tertentu atau metode tertentu yang dijalankan sesuai dengan classnya
3. Untuk mendefinisi ulang suatu fungsi generic digunakan syntax: method.class <- function () expression.
4. Terdapat beberapa fungsi generic yang sudah ada: print, length, dll
5. Namun bisa juga membuat fungsi generic sendiri

Contoh:

```r
# Fungsi Generik
mhs<- function(n, u, g){
  if (g>4 || g<0){
    stop("gpa harus berada pada rentang 0-4")
  }
  mahasiswa<- list(nama= n, umur= u, gpa=g)
  class(mahasiswa)<- "mahasiswa"
  mahasiswa
}

print.mahasiswa <- function(obj){
  cat(obj$nama,"\n")
  cat(obj$umur, "\n")
  cat(obj$gpa, "\n")
```

```
> m
$nama
[1] "andi"

$umur
[1] 19

$gpa
[1] 3.2

attr(,"class")
[1] "MAHASISWA ITERA"
>
```

Membuat fungsi sendiri dengan menggunakan usemethod

```r
# membuat fungsi generik dengan fungsi usemethod
grade<- function(obj){
  UseMethod("grade")
}
grade.mahasiswa<- function(obj){
  cat("ipk", obj$nama, "adalah", obj$gpa)
}
```

### -Kelas S4

1. Mengatasi masalah dalam kelas S3 dengan sistem objek lebih formal
2. Setiap objek didefinisikan secara formal dalam suatu class
3. Sebuah class terdiri dari slot dengan tipe atau class spesifik
4. Kelas S4 didefinisikan menggunakan fungsi setClass()

```
# kelas S4
setClass("mahasiswa", slots= list(nama="character", umur= "numeric",
                                  gpa= "numeric"))
s<- new("mahasiswa", nama= "mat", umur= 21, gpa= 3.1)
s

s@nama
s@umur
```

Dengan output sebagai berikut

```
> s
An object of class "mahasiswa"
Slot "nama":
[1] "mat"

Slot "umur":
[1] 21

Slot "gpa":
[1] 3.1
```

Bisa diupdate dengan nilai yang baru: s@nama <- “Jon Snow”

## Fungsi Generik

1. Fungsi generic yang ekuivalen dengan print()
2. pada S3 adalah fungsi show() Pendeklarasian fungsi generic menggunakan fungsi setMethod()

```
setMethod("show", "mahasiswa", function(obj){
  cat(obj@nama,"\n")
  cat(obj@umur, "\n")
})
```

Membuat fungsi generik baru

```
# membuat fungsi generik baru
setMethod("grade", "mahasiswa", function(obj){
  cat("ip anda adalah", obj@gpa)
})
```