

Modul 4 Pengembangan Pemrograman Fungsi R

2025-10-21

PENDAHULUAN

Fungsi (function) adalah kode-kode yang disusun untuk melakukan tugas tertentu, seperti perhitungan matematis, pembacaan data, analisis statistik, dan lain-lain. dalam komputasi statistik fungsi mempermudah reproduktibilitas analisis, mempercepat eksperimen, dan mempermudah kolaborasi.

Mendefinisikan Fungsi

Mendefinisikan suatu fungsi pada R gunakan kata kunci `function()`.

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

Contoh fungsi menghitung luas persegi panjang

```
luas_persegi_panjang <- function(p, l) {  
  p * l  
}  
luas_persegi_panjang(5, 3)  
## [1] 15
```

Kode R di atas menunjukkan bagaimana cara membuat fungsi sederhana untuk menghitung luas persegi panjang. Pertama, didefinisikan sebuah fungsi bernama `luas_persegi_panjang` yang memiliki dua parameter, yaitu `p` (panjang) dan `l` (lebar). Di dalam fungsi tersebut, operasi perkalian `p * l` digunakan untuk menghitung luas dari persegi panjang sesuai rumus dasar matematika. Setelah fungsi dibuat, dilakukan pemanggilan `luas_persegi_panjang(5, 3)` untuk menghitung luas dengan panjang 5 dan lebar 3. Hasil dari perhitungan tersebut adalah 15, yang ditampilkan sebagai output di konsol R. Melalui contoh ini, dapat dipahami bahwa fungsi dalam R memungkinkan kita untuk menyusun logika perhitungan menjadi kode yang ringkas, terstruktur, dan dapat digunakan berulang kali.

Contoh fungsi menghitung luas persegi panjang dengan menggunakan default argument

```
luas_persegi_panjang <- function(p=5, l=3) {  
  p * l
```

```
}  
luas_persegi_panjang()  
## [1] 15
```

Kode R di atas menunjukkan cara membuat fungsi dengan nilai default pada argumennya. Fungsi bernama `luas_persegi_panjang` memiliki dua parameter, yaitu `p` (panjang) dan `l` (lebar), yang masing-masing diberi nilai default 5 dan 3. Di dalam fungsi, operasi $p * l$ digunakan untuk menghitung luas persegi panjang. Ketika fungsi dipanggil tanpa argumen seperti pada `luas_persegi_panjang()`, R secara otomatis menggunakan nilai default yang sudah ditentukan, yaitu $p = 5$ dan $l = 3$, sehingga menghasilkan output 15. Konsep ini sangat berguna ketika ingin membuat fungsi yang tetap dapat dijalankan meskipun pengguna tidak memberikan nilai input secara eksplisit, sehingga kode menjadi lebih fleksibel dan efisien.

Fungsi Rekursi

Fungsi rekursi adalah fungsi yang memanggil dirinya sendiri secara langsung atau tidak langsung untuk menyelesaikan suatu permasalahan. Fungsi rekursi biasanya digunakan ketika masalah bisa dipecah menjadi submasalah yang lebih kecil dan serupa dan terdapat kondisi berhenti (base case) agar rekursi tidak berjalan tanpa henti.

Contoh penggunaan fungsi rekursi untuk menghitung faktorial

```
faktorial <- function(n) {  
  if (n == 0) {          # base case  
    return(1)  
  } else {  
    return(n * faktorial(n - 1)) # recursive call  
  }  
}  
faktorial(5)  
## [1] 120
```

Kode R di atas menunjukkan contoh penerapan fungsi rekursi untuk menghitung faktorial dari sebuah bilangan. Fungsi bernama `faktorial()` menerima satu parameter `n`. Di dalam fungsi, terdapat kondisi dasar (base case) yaitu ketika $n == 0$, fungsi akan mengembalikan nilai 1, karena secara matematis faktorial dari 0 adalah 1. Jika n tidak sama dengan 0, maka fungsi akan memanggil dirinya sendiri dengan argumen $n - 1$ melalui pernyataan $n * faktorial(n - 1)$. Proses ini akan terus berulang hingga mencapai kondisi dasar. Saat fungsi `faktorial(5)` dijalankan, perhitungannya menjadi $5 * 4 * 3 * 2 * 1$, dan menghasilkan output 120. Kode ini menggambarkan konsep rekursi dengan jelas, yaitu fungsi yang memanggil dirinya sendiri untuk menyelesaikan masalah yang lebih kecil hingga mencapai hasil akhir.

Pemrograman Berbasis Objek Pada R

Bahasa pemrograman R telah mengimplementasikan paradigma pemrograman berorientasi objek (Object-Oriented Programming/OOP), di mana hampir semua elemen di dalam R diperlakukan sebagai objek. Setiap objek memiliki tipe (class) dan atribut yang menentukan bagaimana objek tersebut berperilaku terhadap fungsi tertentu. Pada awal pengembangannya, R menggunakan sistem kelas S3, yaitu sistem objek yang bersifat dinamis, fleksibel, dan tidak terlalu ketat karena tidak memerlukan deklarasi formal atas struktur atau tipe data yang dimiliki suatu objek—cukup dengan memberi atribut class pada objek. Namun, untuk kebutuhan yang lebih kompleks dan agar struktur data lebih terdefinisi secara formal, R kemudian mengembangkan sistem kelas S4. Sistem S4 memiliki pendefinisian yang ketat dan eksplisit, di mana pengguna harus mendefinisikan class, slot (atribut internal), serta metode (method) yang berkaitan dengan class tersebut. Pendekatan ini membuat S4 lebih terstruktur, kuat dalam validasi data, dan lebih sesuai digunakan dalam proyek besar atau pengembangan paket R yang memerlukan konsistensi tipe dan dokumentasi yang jelas.

Kelas S3

Kelas S3 dalam R merupakan sistem kelas yang sederhana dan bersifat primitif, serta menjadi kelas yang paling populer dan banyak digunakan dalam pemrograman R. Sistem ini tidak memiliki definisi formal atau ketat mengenai bagaimana suatu kelas harus dideklarasikan. Sebuah objek dapat dianggap sebagai objek S3 hanya dengan menambahkan atribut class pada objek tersebut, tanpa perlu mendefinisikan strukturnya secara eksplisit. Umumnya, objek S3 dibuat dari tipe data list yang diberi atribut nama kelas tertentu menggunakan fungsi `class()`. Kesederhanaan ini membuat S3 sangat mudah digunakan dan fleksibel.

Contoh penggunaan kelas S3 pada R

```
# Membuat objek biasa
x <- 1:5
# Menambahkan atribut class agar menjadi objek S3
class(x) <- "angka"
# Membuat fungsi print khusus untuk class 'angka'
print.angka <- function(obj) {
  cat("Ini adalah objek dari kelas 'angka'\n")
  cat("Nilainya adalah:", obj, "\n")
}
# Memanggil objek
x

## Ini adalah objek dari kelas 'angka'
## Nilainya adalah: 1 2 3 4 5
```

Kode R di atas menunjukkan bagaimana cara membuat objek S3 dan mendefinisikan metode khusus untuk kelas tersebut. Pertama, sebuah vektor `x` dibuat dengan nilai 1 sampai 5 menggunakan operator `:`. Selanjutnya, vektor ini diubah menjadi objek dari kelas angka dengan menambahkan atribut class melalui `class(x) <- "angka"`. Kemudian, dibuat fungsi `print.angka()` yang merupakan metode khusus untuk mencetak objek bertipe angka. Fungsi ini menerima sebuah objek `obj` dan menampilkan pesan "Ini adalah objek dari kelas 'angka'", diikuti dengan isi dari objek tersebut menggunakan `cat()`. Terakhir, ketika objek `x` dipanggil, R akan mengenali bahwa objek ini memiliki kelas angka dan secara otomatis menggunakan metode `print.angka()` untuk menampilkan informasi, sehingga outputnya berbeda dari vektor biasa. Dengan cara ini, kita bisa menyesuaikan perilaku fungsi tertentu sesuai kelas objek, yang merupakan konsep dasar dari pemrograman berorientasi objek di R menggunakan sistem S3.

Fungsi Generik

Fungsi generik dalam R adalah fungsi yang dapat menyesuaikan perilakunya berdasarkan kelas objek yang diberikan sebagai argumen. Artinya, satu nama fungsi bisa memiliki metode berbeda tergantung pada tipe atau kelas objek yang diproses. Ketika fungsi generik dipanggil, R akan secara otomatis memilih dan menjalankan method yang sesuai dengan class objek tersebut melalui proses yang disebut method dispatch. Fungsi generik biasanya didefinisikan menggunakan perintah `UseMethod()` pada sistem S3 atau `setGeneric()` pada sistem S4.

Contoh penggunaan fungsi generik pada kelas S3

```
# Membuat objek biasa
x <- 1:5

# Menambahkan atribut class agar menjadi objek S3
class(x) <- "angka"

# Membuat fungsi generik 'info'
info <- function(obj) {
  UseMethod("info") # Menentukan bahwa 'info' adalah fungsi generik
}

# Membuat method khusus untuk class 'angka'
info.angka <- function(obj) {
  cat("Ini adalah objek dari kelas 'angka'\n")
  cat("Nilainya adalah:", obj, "\n")
  cat("Panjang objek:", length(obj), "\n")
}

# Memanggil fungsi generik
info(x)
```

```
## Ini adalah objek dari kelas 'angka'  
## Nilainya adalah: 1 2 3 4 5  
## Panjang objek: 5
```

Kode R di atas menunjukkan cara membuat objek S3 dan menggunakan fungsi generik dengan method khusus. Pertama, dibuat vektor x berisi 1–5 dan kelasnya diubah menjadi “angka” menggunakan `class(x) <- “angka”`. Kemudian dibuat fungsi generik `info()` yang memanggil method spesifik berdasarkan kelas objek melalui `UseMethod(“info”)`. Untuk kelas “angka”, dibuat method `info.angka()` yang menampilkan informasi bahwa objek termasuk kelas “angka”, nilai-nilainya, dan panjang objek menggunakan `length()`. Saat `info(x)` dipanggil, R mengenali kelas x dan menjalankan `info.angka()`, sehingga outputnya lebih informatif dibanding mencetak vektor biasa, menunjukkan prinsip pemrograman berorientasi objek S3 di R.

Kelas S4

Sistem kelas S4 dalam R dikembangkan untuk mengatasi berbagai keterbatasan yang ada pada kelas S3 dengan menyediakan pendekatan yang lebih formal dan terstruktur terhadap pemrograman berorientasi objek. Dalam sistem ini, setiap objek harus didefinisikan secara eksplisit dalam sebuah class yang memiliki struktur jelas dan terkontrol. Sebuah kelas S4 terdiri dari slot-slot yang masing-masing memiliki tipe data atau class tertentu, sehingga validasi terhadap isi objek dapat dilakukan secara ketat. Pendefinisian kelas S4 dilakukan menggunakan fungsi `setClass()`, yang memungkinkan pengguna untuk menetapkan atribut, tipe data, dan hubungan antarobjek dengan lebih sistematis serta memastikan konsistensi dalam pengelolaan data di R.

Contoh penggunaan kelas S4 pada R

```
# Membuat class S4 bernama 'Mahasiswa'  
setClass(  
  "Mahasiswa",  
  slots = list(  
    nama = "character",  
    nilai = "numeric"  
  )  
)  
  
# Membuat objek dari class 'Mahasiswa'  
mhs1 <- new("Mahasiswa", nama = "Andi", nilai = 85)  
  
# Membuat fungsi generik 'info'  
setGeneric("info", function(obj) standardGeneric("info"))  
  
## [1] "info"  
  
# Membuat method 'info' khusus untuk class 'Mahasiswa'  
setMethod(  
  "info",  
  "Mahasiswa",  
  function(obj) {  
    paste0("Nama: ", obj@nama, ", Nilai: ", obj@nilai)  
  }  
)
```

```

"info",
"Mahasiswa",
function(obj) {
  cat("Nama Mahasiswa :", obj@nama, "\n")
  cat("Nilai Mahasiswa:", obj@nilai, "\n")
}
)
info(mhs1)

## Nama Mahasiswa : Andi
## Nilai Mahasiswa: 85

```

Kode R di atas menunjukkan penggunaan sistem S4 untuk membuat class dan method secara berorientasi objek. Pertama, dibuat class S4 bernama “Mahasiswa” menggunakan `setClass()` dengan dua slot, yaitu nama bertipe character dan nilai bertipe numeric, yang berfungsi menyimpan data khusus untuk setiap objek mahasiswa. Kemudian dibuat objek `mhs1` dari class “Mahasiswa” menggunakan `new()`, dengan nama = “Andi” dan nilai = 85, sehingga `mhs1` menjadi instance dari class tersebut. Selanjutnya dibuat fungsi generik `info` menggunakan `setGeneric()`, yang memungkinkan pemanggilan method spesifik sesuai class objek yang diterimanya. Untuk class “Mahasiswa”, dibuat method `info` menggunakan `setMethod()`, yang mengambil objek `obj` dan menampilkan isi slot nama dan nilai menggunakan operator `@`. Saat `info(mhs1)` dipanggil, R mengenali bahwa `mhs1` adalah objek dari class “Mahasiswa” dan mengeksekusi method yang sesuai, sehingga informasi nama dan nilai mahasiswa ditampilkan secara terstruktur, memperlihatkan prinsip pemrograman berorientasi objek S4 di R.

Kelas R6

R6 adalah sistem OOP modern di R yang mendukung enkapsulasi, pewarisan dan metode dengan sintaks yang mirip dengan bahasa pemrograman lain seperti Python. R6 menggunakan kelas referensi secara bawaan. Untuk menggunakan R6 bisa menggunakan pustaka “R6” dan menggunakan `R6Class`.

Contoh penggunaan kelas R6 pada R

```

# Memanggil Library R6
library(R6)

# Membuat class R6 bernama 'Mahasiswa'
Mahasiswa <- R6Class("Mahasiswa",
  public = list(
    nama = NULL,
    nilai = NULL,

    # Konstruktor (fungsi yang dijalankan saat objek dibuat)
    initialize = function(nama, nilai) {
      self$nama <- nama
    }
  )
)

```

```

        self$nilai <- nilai
    },

    # Method untuk menampilkan data
    info = function() {
        cat("Nama Mahasiswa :", self$nama, "\n")
        cat("Nilai Mahasiswa:", self$nilai, "\n")
    },

    # Method untuk mengubah nilai
    ubahNilai = function(nilaiBaru) {
        self$nilai <- nilaiBaru
        cat("Nilai berhasil diubah menjadi:", self$nilai, "\n")
    }
)
)

# Membuat objek dari class Mahasiswa
m1 <- Mahasiswa$new("Andi", 85)

# Memanggil method info()
m1$info()

## Nama Mahasiswa : Andi
## Nilai Mahasiswa: 85

# Mengubah nilai mahasiswa
m1$ubahNilai(90)

## Nilai berhasil diubah menjadi: 90

# Cek Lagi
m1$info()

## Nama Mahasiswa : Andi
## Nilai Mahasiswa: 90

```

Kode R di atas menunjukkan penggunaan class R6 untuk membuat objek berorientasi objek yang stateful di R. Pertama, library R6 dipanggil untuk mengaktifkan fungsionalitas R6, lalu dibuat class Mahasiswa menggunakan R6Class() dengan field nama dan nilai, konstruktor initialize() untuk menginisialisasi objek saat dibuat, serta method info() untuk menampilkan data mahasiswa dan ubahNilai() untuk mengubah nilai secara langsung. Objek m1 dibuat dari class ini menggunakan \$new("Andi", 85), sehingga m1 memiliki nama "Andi" dan nilai 85. Saat m1\$info() dipanggil, method info() menampilkan nama dan nilai mahasiswa, kemudian m1\$ubahNilai(90) dijalankan untuk memperbarui nilai menjadi 90 dan pemanggilan m1\$info lagi menampilkan nilai terbaru, menunjukkan bahwa objek R6 bersifat stateful dan bisa dimodifikasi setelah dibuat.