MODUL 5
CONDITIONAL EXPRESSIONS

A. Tujuan Praktikum

- Memahami penggunaan pernyataan IF-ELSE, IFELSE, ANY, dan ALL\
- Memahami langkah pembuatan dan pemanggilan fungsi pada R
- Memahami penggunaan pernyataan FOR-LOOP
- Memahami penggunaan Vektorisasi dan Fungsi `sapply`

B. Alokasi Waktu

1 x pertemuan = 120 menit

C. Dasar Teori

Conditional Expressions merupakan salah satu fitur dasar pemrograman yang digunakan untuk *flow control*. Ekspresi bersyarat yang paling umum digunakan adalah pernyataan IF-ELSE. Di R, kita dapat melakukan analisis data tanpa menggunakan ekspresi persyaratan. Namun, kita biasanya akan membutuhkan ekspresi bersyarat untuk menulis fungsi dan paket pada *script* yang kita buat.

Pernyataan IF-ELSE, IFELSE, ANY, dan ALL

Berikut ini adalah contoh sederhana penggunaan pernyataan IF-ELSE. Pada *script* berikut, kita akan menampilkan kebalikan dari a (1/a) kecuali jika a bernilai 0:

```
a <- 0
if(a!=0){
print(1/a)
} else{
print("Nilai a tidak boleh 0.")
}
#> [1] " Nilai a tidak boleh 0."
```

Contoh selanjutnya, kita akan menggunakan data "US *murders*" untuk menghitung '*murder_rate*':

```
library(dslabs)
data(murders)
murder_rate <- murders$total / murders$population*100000
```

Selanjutnya, kita akan coba mencari tahu negara bagian mana yang memiliki tingkat pembunuhan atau '*murder_rate*'-nya kurang dari 0,5.

```
ind <- which.min(murder_rate)
if(murder_rate[ind] < 0.5){
print(murders$state[ind])
} else{
print("Tidak ada")
}
#> [1] "Vermont"
```

Dengan pernyataan IF, kita juga dapat memanfaatkan penggunaan ELSE untuk menampilkan hasil berupa keterangan jika tidak ada negara yang memenuhi syarat yang diberikan. Kita akan coba lagi menggunakan IF-ELSE dengan mengubah nilai '*murder_rate*' menjadi 0,25, dan mendapat hasil yang berbeda:

```
if(murder_rate[ind] < 0.25){
print(murders$state[ind])
} else{
print("Tidak ada.")
}
#> [1] " Tidak ada."
```

Fungsi *Conditional Expressions* lain yang dapat digunakan adalah IFELSE. Fungsi ini membutuhkan tiga argumen: logika yang diinginkan dan dua kemungkinan jawaban. Jika logika adalah BENAR, nilai dalam argumen kedua yang akan ditampilkan sebagai *output*, sebaliknya jika SALAH, nilai dalam argumen ketiga yang akan ditampilkan. Contohnya adalah sebagai berikut:

```
a <- 0
ifelse(a > 0, 1/a, NA)
#> [1] NA
```

Fungsi ini biasanya digunakan untuk mengevaluasi data dengan tipe vektor. IFELSE akan memeriksa setiap elemen dari vektor logika dan menampilkan argumen kedua dari hasil evaluasi per-elemen vektor jika elemen bernilai BENAR, atau argumen ketiga jika elemen dari vektor yang disediakan bernilai SALAH.

```
a <- c(0, 1, 2, -4, 5)
result <- ifelse(a > 0, 1/a, NA)
```

Tabel dibawah ini akan membantu kita melihat tiap proses yang terjadi pada pernyataan IFELSE:

a	Argumen 1	Argumen 2	Argumen 2	Hasil
0	SALAH	Inf	NA	NA
1	BENAR	1.00	NA	1.0
2	BENAR	0.50	NA	0.5
-4	SALAH	-0.25	NA	NA
5	BENAR	0.20	NA	0.2

Selanjutnya, kita akan membahas mengenai fungsi ANY and ALL. Fungsi ANY mengevaluasi vektor logika dan mengembalikan BENAR jika salah satu entri bernilai BENAR. Sedangkan fungsi ALL mengevaluasi vektor logika dan mengembalikan BENAR jika semua entri bernilai BENAR. Berikut contohnya:

```
z <- c(TRUE, TRUE, FALSE)
any(z)
#> [1] TRUE
all(z)
#> [1] FALSE
```

Membuat Fungsi

Dalam melakukan analisis data, ada saatnya kita merasa perlu untuk melakukan operasi yang sama berulang kali. Contoh sederhana adalah menghitung rata-rata. Kita dapat menghitung rata-

rata vektor x menggunakan fungsi `sum` dan `length` dengan rumus: `sum(x)/length(x)`. Jika kita membutuhkan komputasi berulang kali, akan lebih efisien jika kita menuliskan fungsi baru yang akan melakukan komputasi tersebut secara otomatis saat kita panggil fungsi yang berkaitan. Sebetulnya, menghitung rata-rata merupakan komputasi yang sangat umum, sehingga fungsi rata-rata sudah termasuk dalam fungsi komputasi dasar yang disediakan R. Namun, jika fungsi yang kita inginkan belum ada, R mengizinkan kita untuk mendefinisikan sendiri fungsi yang kita inginkan. Sebagai contoh, versi sederhana dari fungsi menghitung rata-rata yang akan kita buat sendiri, yaitu '*avg*' didefinisikan seperti ini:

```
avg <- function(x){
s <- sum(x)
n <- length(x)
s/n
}
```

Dengan mendefiniskan fungsi baru `avg` seperti diatas, kita dapat menghitung rata-rata dengan memanggil fungsi `avg` seperti contoh berikut:

```
s <- 3
avg(1:10)
#> [1] 5.5
s
#> [1] 3
```

Yang perlu diperhatikan adalah: variabel yang didefinisikan di dalam suatu fungsi tidak disimpan di *workspace*. Sehingga, kita harus menyimpan nilai `s` dan `n` terlebih dahulu sebelum memanggil fungsi `avg`.

Secara umum, fungsi merupakan objek, sehingga kita perlu menetapkan nama variabel untuk fungsi dengan menggunakan (`<-`).Bentuk umum dari definisi fungsi dapat dilihat pada contoh dibawah ini:

```
my_function <- function(VARIABLE_NAME){
perform operations on VARIABLE_NAME and calculate VALUE
VALUE
}
```

Fungsi yang kita definisikan dapat terdiri dari beberapa argumen serta nilai *default*. Sebagai contoh, kita dapat mendefinisikan fungsi untuk menghitung aritmatika atau rata-rata geometrik tergantung pada variabel yang dimasukkan pengguna seperti contoh:

```
avg <- function(x, arithmetic = TRUE){
n <- length(x)
ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

FOR-LOOP

Rumus untuk menjumlahkan rangkaian angka 1 + 2 + · · · · n adalah n (n + 1) / 2. Bagaimana jika kita kurang yakin jika rumus yang akan kita gunakan tersebut adalah benar? Bagaimana kita bisa memeriksanya? Dengan menggunakan fungsi yang telah kita pelajari pada bagian sebelumnya, kita dapat membuat fungsi baru yang akan kita gunakan untuk menghitung Sn

```
compute_s_n <- function(n){
x <- 1:n
sum(x)
}
```

Selanjutnya, bagaimana kita dapat menghitung Sn untuk berbagai nilai n, misl: n = 1,.., 25? Apakah kita harus menuliskan 25 baris pemanggilan fungsi `compute_s_n`? Disinilah kita dapat menggunakan fungsi FOR-LOOP untuk pemrograman. Contoh sederhana: implementasi FOR-LOOP untuk melakukan tugas yang sama persis berulang-ulang, dan satu-satunya hal yang berubah adalah nilai '*n*'.

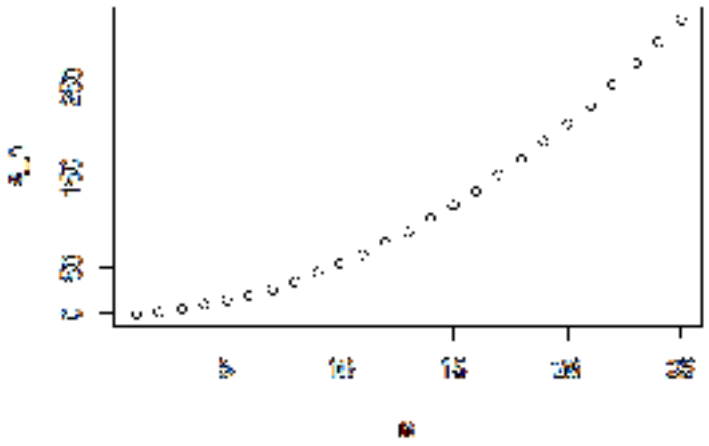
```
for(i in 1:5){
print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
```

Penggunaan FOR-LOOP untuk komputasi '*Sn*' adalah sebagai berikut:

```
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
s_n[n] <- compute_s_n(n)
}
```

Dalam setiap iterasi n = 1, n = 2, dll ..., kita melakukan penghitungan Sn dan menyimpannya sebagai data ke- n dari `s_n`. Untuk keperluan analisis data lebih lanjut, kita dapat membuat plot untuk mencari pola `s_n`:

```
n <- 1:m
plot(n, s_n)
```



Vektorisasi dan Fungsi `sapply`

Meskipun FOR-LOOP merupakan salah satu konsep penting untuk dipahami, dalam R kita jarang menggunakannya, karena vektorisasi lebih disukai daripada FOR-LOOP. Hal ini dikarenakan vektorisasi dapat digunakan dengan penulisan *script* yang lebih pendek dan lebih jelas. Dengan *vectorized function*, kita dapat menerapkan fungsi operasi yang sama pada setiap anggota vektor seperti contoh berikut.

```
x <- 1:10
sqrt(x)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
y <- 1:10
x*y
#> [1] 1 4 9 16 25 36 49 64 81 100
```

Komputasi yang dicontohkan diatas tidak menggunakan FOR-LOOP. Namun, tidak semua fungsi dapat menerapkan *vectorized function*. Misalnya, fungsi yang telah kita buat sebelumnya, `compute_s_n`. Fungsi tersebut tidak dapat melakukan komputasi secara *element-wise*, karena input yang diharapkan bukan berupa vektor, melainkan skalar. Contoh, potongan *script* dibawah ini juga tidak akan melakukan komputasi fungsi pada setiap '*n*':

```
n <- 1:25
compute_s_n(n)
```

Functionals adalah fungsi yang membantu kita menerapkan fungsi yang sama untuk setiap entri dalam vektor, matriks, *data frame*, atau *list*. Pada bagian ini kita akan mencoba mengimplementasikan *functionals* yang beroperasi pada vektor numerik, logika, dan karakter, yaitu: `sapply`.

Fungsi `sapply` memungkinkan kita untuk melakukan operasi secara *element-wise* pada fungsi apa pun. Berikut cara kerjanya:

```
x <- 1:10
sapply(x, sqrt)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Setiap elemen '*x*' akan diteruskan ke fungsi `sqrt` dan hasil komputasinya akan disimpan sementara dan digabungkan hingga nilai '*x*'terakhir. Hasil komputasi yang dilakukan adalah berupa vektor yang panjangnya sama dengan nilai '*x*'.
Maka dari itu, agar FOR-LOOP yang kita miliki dapat beroperasi secara *element-wise*, kita dapat mengubah *script* fungsi `s_n` menjadi sebagai berikut:

```
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

D. Latihan

1. Fungsi `nchar` dapat digunakan untuk menghitung jumlah karakter dari suatu vektor karakter. Buatlah satu baris kode yang akan menyimpan hasil komputasi pada variabel '*new_names*' dan berisi singkatan nama negara ketika jumlah karakternya lebih dari 8 karakter.
2. Buat fungsi `sum_n` yang dapat digunakan untuk menghitung jumlah bilangan bulat dari 1 hingga n. Gunakan pula fungsi ini untuk menentukan jumlah bilangan bulat dari 1 hingga 5.000.
3. Buat fungsi `compute_s_n` yang dapat digunakan untuk menghitung jumlah Sn = $1^2 + 2^2 + 3^2 + \ldots n^2$. Tampilkan hasil penjumlahan ketika $n = 10$.
4. Buat vektor numerik kosong dengan nama: `s_n` dengan ukuran:25 menggunakan `s_n <- vector ("numeric", 25)`.
   Simpan di hasil komputasi $S_1, S_2, \ldots S_{25}$ menggunakan FOR-LOOP.
5. Ulangi langkah pada soal no. 4 dan gunakan fugsi `sapply`.