

Struktur Kontrol

OBJEKTIF:

1. Mahasiswa mampu memahami Struktur Kontrol, Ekspresi Boolean, Struktur Seleksi, Operator Logis, Struktur Iterasi, dan Pola-pola Loop pada Python.
 2. Mahasiswa mampu mengimplementasikan statement if, if-else, if-elif-else menggunakan bahasa pemrograman Python.
 3. Mahasiswa mampu mengimplementasikan Loop while, Infinite Loop, dan Loop for menggunakan bahasa pemrograman Python.
 4. Mahasiswa mampu menggunakan Operator and, Operator or, dan Operator not pada python.
-

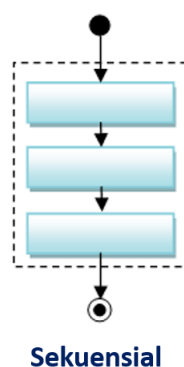
3.1 Struktur Kontrol

Algoritma adalah proses langkah per langkah untuk menyelesaikan suatu persoalan. Kita membuat algoritma dengan menentukan aksi-aksi yang dikerjakan dan urutan pengerjaan aksi-aksi tersebut. Kita menentukan urutan pengerjaan aksi-aksi dengan menggunakan **struktur kontrol**. Terdapat tiga struktur kontrol untuk menentukan urutan pengekseskuan aksi-aksi dalam algoritma:

- **Struktur Sekuensial:** rangkaian aksi-aksi dikerjakan secara berurutan.
- **Struktur Seleksi:** rangkaian aksi-aksi tertentu dikerjakan jika memenuhi suatu kondisi.
- **Struktur Iterasi:** rangkaian aksi-aksi dikerjakan berulang kali.

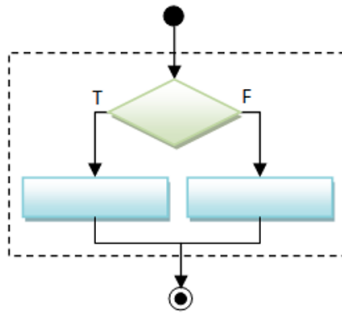
Struktur Sekuensial

Struktur sekuensial adalah struktur *default* program dimana statement-statement dieksekusi secara berurutan dari baris atas ke baris berikutnya dan selanjutnya. Flowchart dari struktur sekuensial diilustrasikan pada gambar berikut:



Struktur Seleksi

Struktur seleksi memungkinkan program untuk mengeksekusi statement-statement berbeda berdasarkan suatu kondisi. Struktur seleksi sering disebut dengan struktur keputusan/percabangan karena program membuat keputusan dengan menguji kondisi terlebih dahulu dan mengeksekusi statement-statement tertentu berdasarkan hasil uji kondisi tersebut (yang membuat alur program bercabang)



Seleksi

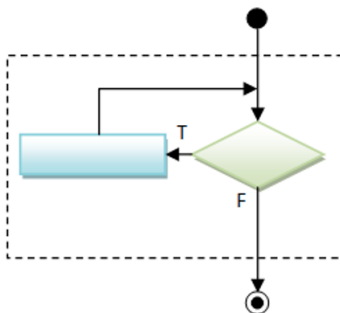
Berikut adalah contoh penulisan struktur seleksi pada Python:

```

if nilai >= 60:
    # Kerjakan sesuatu
    <statement>
    ...
    <statement>
  
```

Struktur Iterasi

Struktur iterasi atau perulangan memungkinkan statement-statement dieksekusi secara berulang kali berdasarkan suatu kondisi atau jumlah tertentu. Struktur ini sering disebut juga sebagai struktur loop. Kondisi-kondisi dalam struktur seleksi maupun iterasi dituliskan dengan menuliskan **ekspresi Boolean**.



Iterasi

Berikut adalah contoh penulisan struktur iterasi pada Python:

```

while count <= 10:
    # Kerjakan sesuatu
    <statement>
    ...
    <statement>
  
```

3.2 Ekspresi Boolean

Ekspresi Boolean adalah ekspresi yang dievaluasi ke tipe Boolean `bool`. Tipe `bool` hanya dapat bernilai `True` atau `False`. Kita dapat menuliskan ekspresi Boolean menggunakan operator relasional. Tabel berikut mendaftar operator-operator relasional yang tersedia dalam Python:

Operator Relasional	Contoh	Arti
<	<code>x < y</code>	Apakah <code>x</code> lebih kecil dari <code>y</code> ?
<=	<code>x <= y</code>	Apakah <code>x</code> lebih kecil dari atau sama dengan <code>y</code> ?
>	<code>x > y</code>	apakah <code>x</code> lebih besar dari <code>y</code> ?
>=	<code>x >= y</code>	Apakah <code>x</code> lebih besar dari atau sama dengan <code>y</code> ?
==	<code>x == y</code>	Apakah <code>x</code> sama dengan <code>y</code> ?
!=	<code>x != y</code>	Apakah <code>x</code> tidak sama dengan <code>y</code> ?

Catatan. Kesalahan yang sering dilakukan programmer pemula adalah menggunakan simbol `=` untuk ekspresi Boolean sama dengan yang seharusnya menggunakan `==`.

Sesi interaktif berikut mencontohkan sejumlah ekspresi Boolean:

```
>>> 3 != 5
True
>>> 3 == 4
False
>>> 3 < 5
True
>>> 3 > 5
False
>>> 3 == 5
False
>>> 3 >= 3
True
>>> 3 <= 3
True
```

Selain membandingkan antara dua integer, kita dapat membuat perbandingan antara dua floating point atau antara integer dengan floating point. Sesi interaktif berikut mencontohkannya:

```
>>> 23.1 >= 23
True
>>> 23.1 >= 23.1
True
>>> 23.1 <= 23.1
True
>>> 23.1 <= 23
False
>>> 67.3 == 87
False
>>> 67.0 == 67
True
>>> 67.0 != 67
False
>>> 67.0 != 23
True
```

Kita juga dapat mempunyai ekspresi Boolean yang terdiri dari operator aritmatika dan operator relasional:

```
>>> 1 + 3 > 7
False
```

Pada contoh di atas, ekspresi aritmatika, `1 + 3`, dievaluasi terlebih dahulu sebelum hasilnya dibandingkan dengan nilai `7`.

Kita juga dapat membentuk ekspresi Boolean menggunakan variabel seperti berikut:

```
>>> x = 4
>>> y = 2
>>> x + 2 > y * 4 / 2
True
```

Selain membandingkan tipe numerik, kita juga dapat membandingkan dua string dengan operator relasional. Umumnya kita hanya menggunakan operator relasional `==` dan `!=` pada perbandingan dua string. Operator `==` digunakan untuk mengetahui apakah kedua string sama dan operator relasional `!=` digunakan untuk mengetahui apakah kedua string berbeda. Sesi interaktif berikut mencontohkan perbandingan dua string:

```
>>> nama1 = 'Alan'
>>> nama2 = 'Alan'
>>> nama1 == nama2
False
>>> nama1 != nama2
True
```

Kita juga dapat menuliskan ekspresi Boolean sebagai perbandingan berantai dengan dua operator relasional. Sebagai contoh, ekspresi:

```
1 < x <= 5
```

membandingkan apakah nilai variabel `x` lebih besar dari 1 dan lebih kecil atau sama dengan 5. Sesi interaktif berikut mencontohkan perbandingan berantai:

```
>>> x = 3
>>> 1 < x <= 5
True
```

Umumnya kita menggunakan perbandingan berantai untuk menguji apakah suatu nilai berada dalam suatu interval seperti pada contoh di atas.

3.3 Struktur Seleksi

Struktur seleksi memungkinkan sebuah program memilih (menyeleksi) pengeksekusian statement-statement berbeda berdasarkan kondisi. Pilihan statement-statement yang dieksekusi ini disebut dengan cabang, karena menyebabkan alur eksekusi program bercabang.

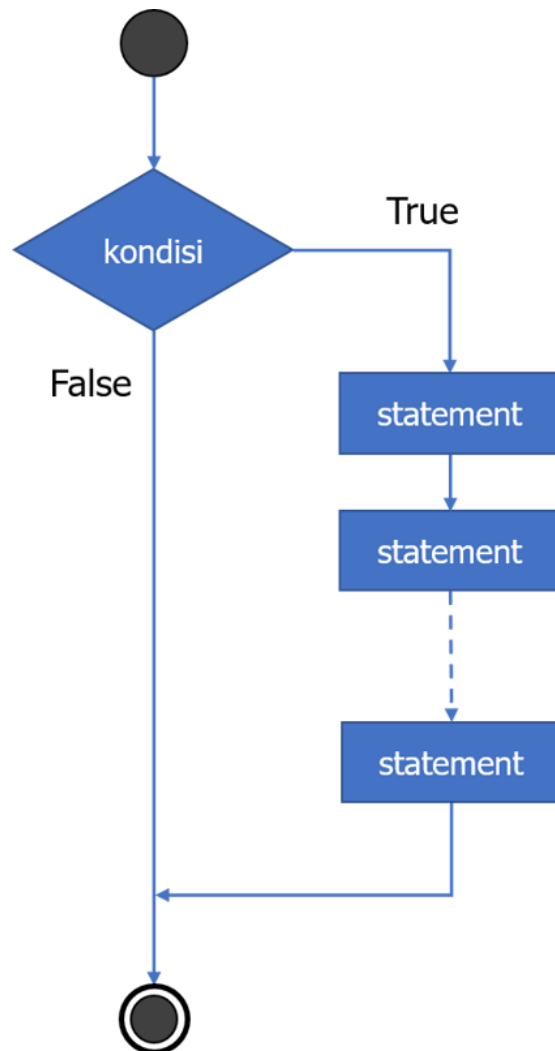
Terdapat tiga statement yang dapat digunakan dalam Python untuk membentuk struktur seleksi:

- **Statement `if`**: digunakan untuk membentuk struktur seleksi satu cabang.
- **Statement `if-else`**: digunakan untuk membentuk struktur seleksi dua cabang.

- **Statement** `if-elif-else`: digunakan untuk membentuk struktur seleksi multi cabang (lebih dari dua cabang).

3.3.1 Statement `if`

Statement `if` digunakan untuk membuat struktur seleksi yang mengeksekusi statement-statement tertentu jika suatu kondisi terpenuhi dan melewati pengeksekusian statement-statement tersebut jika kondisi tersebut tidak terpenuhi. Gambar berikut mengilustrasikan flowchart dari statement `if`:



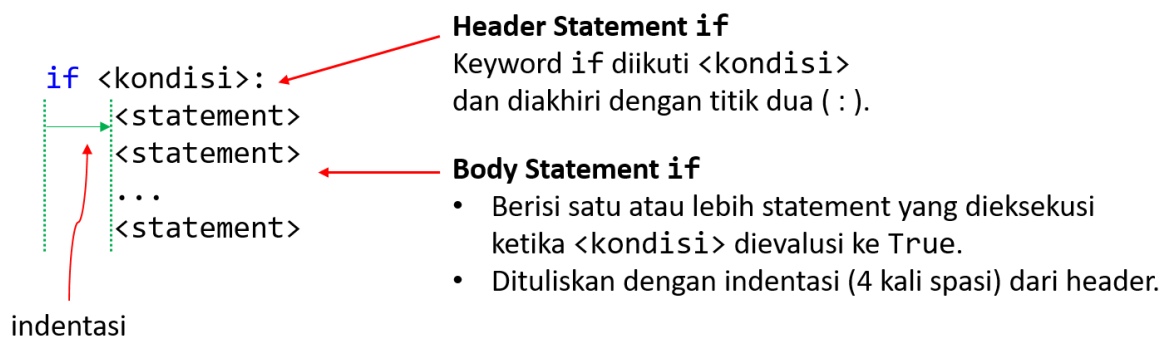
Pada gambar flowchart struktur seleksi di atas dapat dilihat, alur program mempunyai satu cabang. Jika kondisi terpenuhi (bernilai True), maka alur program menempuh cabang ke kanan lalu mengeksekusi sejumlah statement. Jika kondisi tidak terpenuhi (bernilai False), maka alur program menempuh jalur utama ke bawah yang melewati statement-statement pada cabang.

Bentuk umum penulisan statement `if` adalah sebagai berikut:

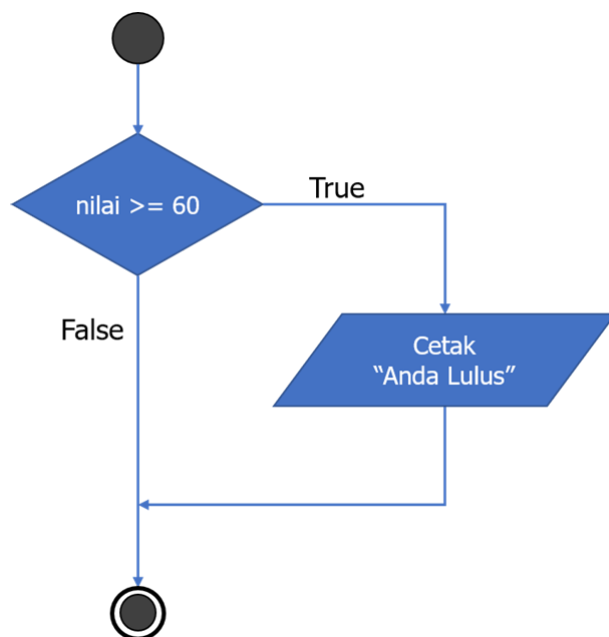
```
if <kondisi>:  
    <statement>  
    <statement>  
    ...  
    <statement>
```

Dimana `<kondisi>` yang dituliskan setelah keyword `if` berupa **ekspresi Boolean**. `<statement>` - `<statement>` dieksekusi jika ekspresi Boolean tersebut dievaluasi ke `True` dan dilewati jika ekspresi Boolean tersebut dievaluasi ke `False`. Perhatikan `<statement>` - `<statement>` dituliskan dengan indentasi dari keyword `if`.

Gambar berikut menjelaskan bagian-bagian dari statement `if`:



Kita akan melihat sebuah program yang menggunakan statement `if`. Misalkan kita membuat sebuah program kelulusan ujian yang menampilkan `'Anda Lulus!'` jika nilai ujian lebih besar atau sama dengan 60. Flowchart dari program ini dapat digambarkan seperti gambar berikut:



Pseudocode struktur seleksi dari program dapat ditulis seperti berikut:

```
Jika nilai >= 60  
    cetak "Anda Lulus!"
```

Kita dapat menuliskan pseudocode di atas dalam kode Python seperti berikut:

```
if nilai >= 60:
    print('Anda lulus!')
```

Kode Python di atas akan mencetak `Anda lulus!` ke layar, jika kondisi `nilai >= 60` dievaluasi ke `True`.

Berikut adalah kode lengkap dari program kelulusan ujian:

```
# kelulusan1.py
# Program yang menampilkan 'Anda lulus!' jika nilai yang dimasukkan pengguna
# lebih besar atau sama dengan 60

def main():
    nilai = float(input('Masukkan nilai ujian: '))
    if nilai >= 60:
        print('Anda lulus!')

main()
```

Contoh output dari program `kelulusan1.py` di atas:

Output program jika input nilai kurang dari 60

```
Masukkan nilai ujian: 58
```

Output program jika input nilai lebih besar atau sama dengan 60

```
Masukkan nilai ujian: 70
Anda lulus!
```

Perhatikan pada output di atas, jika pengguna memasukkan input kurang dari 60 (pada contoh di atas, pengguna memasukkan nilai 58), program tidak menampilkan teks `Anda lulus!`. Ini berarti statement `print('Anda lulus!')` yang berada di dalam body statement `if` tidak dieksekusi. Namun, jika pengguna memasukkan input nilai lebih besar atau sama dengan 60, program menampilkan teks `Anda lulus!` yang berarti statement `print('Anda lulus!')` dieksekusi.

Kita dapat mempunyai lebih dari satu statement dalam body statement `if`. Sebagai contoh, program berikut mempunyai dua statement `print` di dalam body statement `if`:

```
# kelulusan2.py
# Program yang mendemonstrasikan statement if dengan lebih dari satu statement
# di dalam body-nya.

def main():
    nilai = float(input('Masukkan nilai ujian: '))
    if nilai >= 60:
        print('Selamat!')
        print('Anda lulus!')

main()
```

Berikut adalah contoh output dari program `kelulusan2.py` di atas:

```
Masukkan nilai ujian: 90
Selamat!
Anda lulus!
```

Satu hal yang perlu diperhatikan ketika menulis statement `if` yang di dalam body-nya terdapat lebih dari satu statement adalah statement-statement tersebut harus dituliskan dengan indentasi yang sama. Atau dengan kata lain, indentasi pada penulisan statement-statement dalam body statement `if` harus konsisten. Misalkan kita tidak menuliskan statement-statement dalam body statement `if` secara konsisten, seperti contoh berikut:

```
# kelulusan2_error.py
# Program yang menghasilkan error karena statement-statement
# di dalam body statement if tidak dituliskan dengan indentasi yang sama

def main():
    nilai = float(input('Masukkan nilai ujian: '))
    if nilai >= 60:
        print('Selamat!')
        print('Anda lulus!')

main()
```

Perhatikan pada kode program `kelulusan2_error.py` di atas, dua statement yang berada di dalam body statement `if`, yaitu statement pada baris 8 dan baris 9 tidak dituliskan dengan indentasi yang sama. Pada baris 9 terdapat tambahan sebuah spasi sebelum statement. Penulisan seperti ini akan menghasilkan error. Jika kita menjalankan program di atas, maka kita akan mendapatkan pesan error berikut:

```
ERROR!!
File ".\kelulusan2_error.py", line 10
    print('Anda lulus!')
    ^
IndentationError: unexpected indent
```

Catatan: Indentasi statement-statement dalam body statement `if` harus konsisten (sesuai konvensi indentasi adalah 4 spasi). Beda indentasi akan menyebabkan error.

Jika di dalam program terdapat statement-statement yang dieksekusi setelah statement `if` dieksekusi, maka statement-statement tersebut harus dituliskan di luar struktur statement `if` dengan indentasi sejajar dengan header statement `if`. Sebagai contoh, perhatikan program berikut:


```
# kelulusan3.py
# Program dengan statement setelah statement if

def main():
    nilai = float(input('Masukkan nilai ujian: '))

    if nilai >= 60:
        print('Selamat!')
        print('Anda lulus!')

    print('Terima kasih!')

main()
```

Catatan: Statement di luar struktur `if` dituliskan dengan indentasi sama seperti header `if`.

Berikut adalah contoh output dari program `kelulusan3.py`:

Output program jika input nilai kurang dari 60

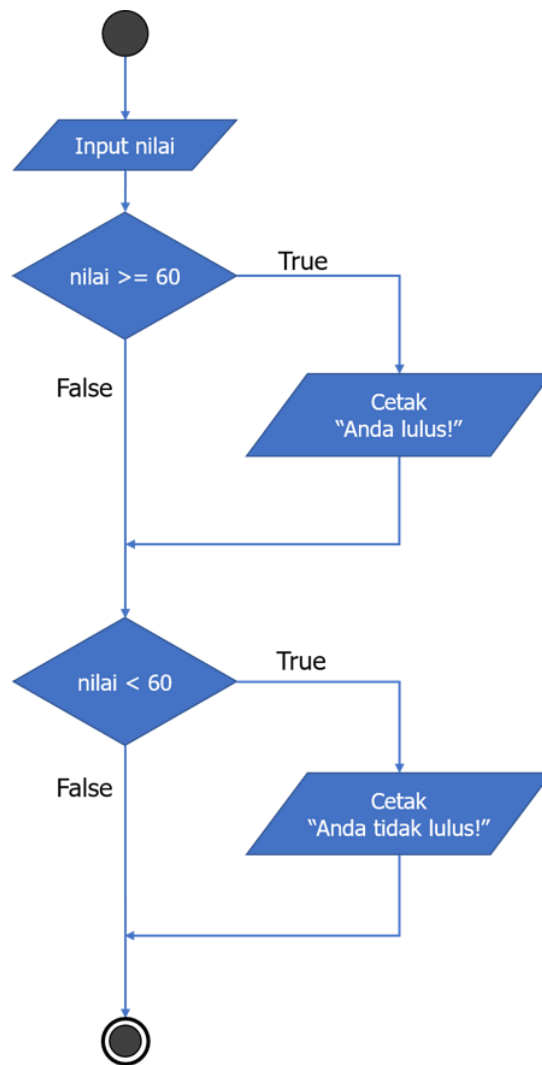
```
Masukkan nilai ujian: 56
Terima kasih!
```

Output program jika input nilai lebih besar atau sama dengan 60

```
Masukkan nilai ujian: 88
Selamat!
Anda lulus!
Terima kasih!
```

Rangkaian Statement `if`

Kita dapat menuliskan rangkaian statement `if` untuk menguji beberapa kondisi. Misalkan kita ingin melanjutkan program `kelulusan.py` sebelumnya sehingga program tersebut menampilkan teks `Anda tidak lulus!` jika pengguna memasukkan input berupa nilai kurang dari 60. Flowchart dari lanjutan program `kelulusan ujian` dapat digambarkan seperti gambar berikut:



Pseudocode program lengkap dapat kita tuliskan seperti berikut:

```
Input nilai
Jika nilai >= 60
    Cetak "Anda lulus!"
Jika nilai < 60
    Cetak "Anda tidak lulus!"
```

Pseudocode di atas dapat dituliskan dalam program Python seperti berikut:

```
# kelulusan4.py
# Program ini menampilkan 'Anda lulus!' jika nilai yang dimasukkan pengguna
# lebih besar dari 60 dan menampilkan 'Anda tidak lulus!' jika nilai
# yang dimasukkan lebih kecil dari 60

def main():
    nilai = float(input('Masukkan nilai ujian: '))
    if nilai >= 60:
        print('Anda lulus!')
    if nilai < 60:
        print('Anda tidak lulus!')

main()
```

Contoh output dari program `kelulusan4.py` di atas:

Input nilai kurang dari 60

```
Masukkan nilai ujian: 58
Anda tidak lulus!
```

Input nilai lebih dari 60

```
Masukkan nilai ujian: 92
Anda lulus!
```

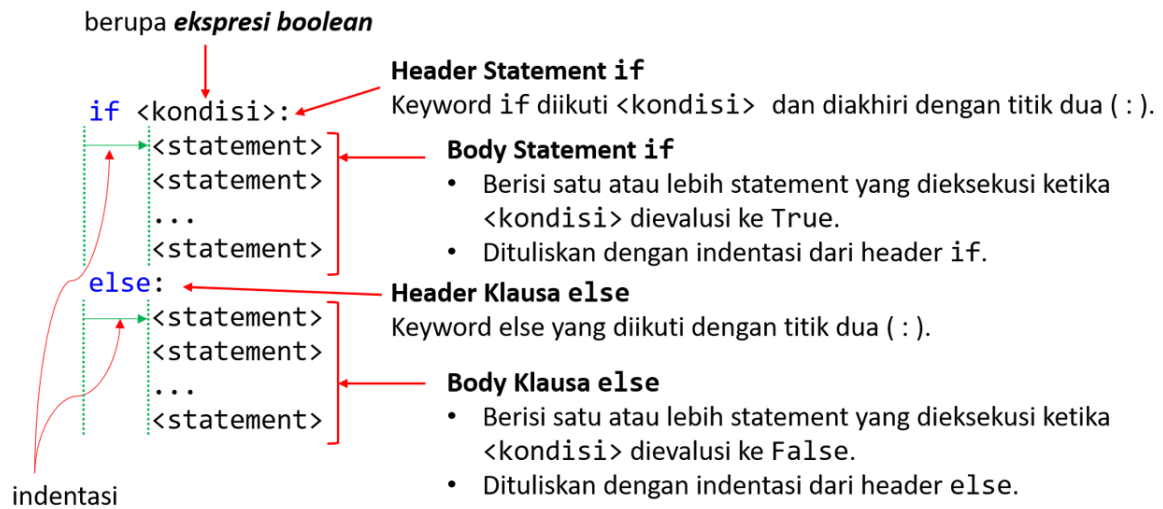
Perhatikan bahwa kondisi `nilai >= 60` dan kondisi `nilai < 60` adalah dua kondisi yang *mutually exclusive*. *Mutually exclusive* berarti jika satu kondisi terpenuhi (`True`) maka kondisi yang lain tidak terpenuhi (`False`). Untuk dua kondisi yang *mutually exclusive*, kita dapat menggunakan statement `if-else` yang akan kita bahas selanjutnya dibandingkan menggunakan dua statement `if`.

3.3.2 Statement `if-else`

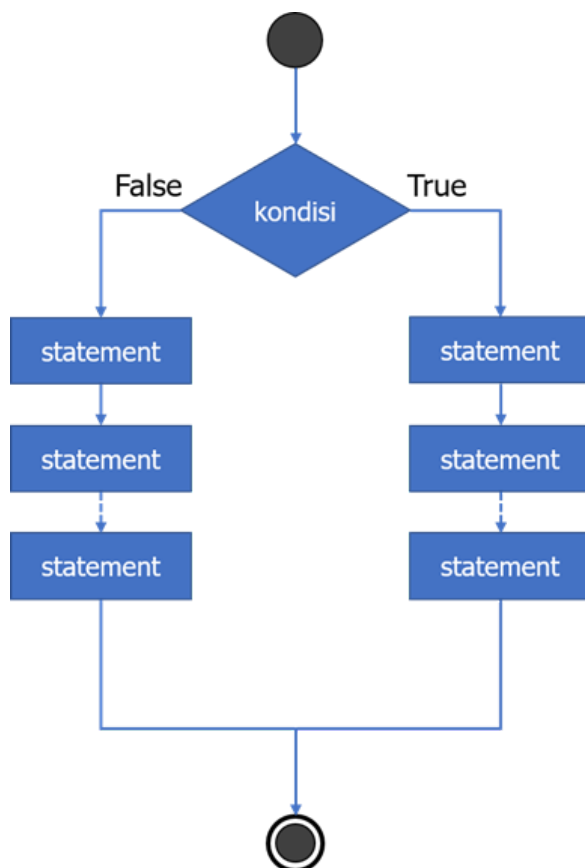
Statement `if` dapat diekstensi dengan menambahkan klausa `else` untuk membuat program mengeksekusi statement-statement lain ketika kondisi tidak terpenuhi. Struktur statement `if` dengan klausa `else` disebut dengan statement `if-else` dengan bentuk umum penulisan sebagai berikut:

```
if <kondisi>:
    <statement>
    <statement>
    ...
    <statement>
else:
    <statement>
    <statement>
    ...
    <statement>
```

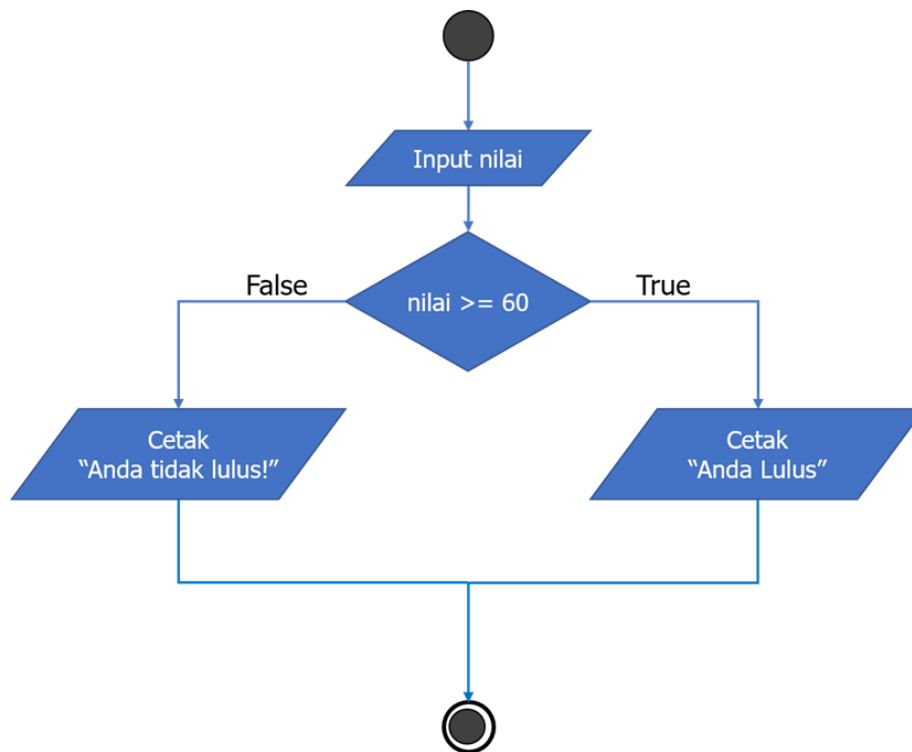
Gambar berikut menjelaskan bagian-bagian dari statement `if-else`:



Flowchart dari statement `if-else` dapat digambarkan seperti gambar berikut:



Pada contoh program kelulusan dengan dua statement `if`, terdapat dua kondisi: `nilai >= 60` dan `nilai < 60`. Kedua kondisi ini disebut *mutually exclusive* karena jika nilai ≥ 60 adalah benar, maka nilai < 60 pasti salah, dan sebaliknya. Sehingga, sebenarnya hanya salah satu kondisi yang perlu diuji. Logika program kelulusan dengan dua statement `if` sebelumnya dapat kita ubah dengan menggunakan struktur seleksi dua cabang seperti pada flowchart di bawah ini:



Kita dapat menuliskan program kelulusan dengan statement `if-else` seperti berikut:

```
# kelulusan_if_else.py
# Program yang menampilkan Anda lulus jika nilai yang dimasukkan
# lebih besar dari 60 dan menampilkan Anda tidak lulus jika
# nilai yang dimasukkan lebih kecil dari 60

def main():
    nilai = float(input('Masukkan nilai ujian: '))
    if nilai >= 60:
        print('Anda lulus!')
    else:
        print('Anda tidak lulus!')

main()
```

Catatan: Statement `if-else` lebih efisien dan lebih cepat dibandingkan rangkaian statement `if` karena pengujian kondisi hanya terjadi satu kali.

Contoh output dari program `kelulusan_if_else.py` di atas:

Input nilai kurang dari 60

```
Masukkan nilai ujian: 58
Anda tidak lulus!
```

Input nilai lebih dari 60

Masukkan nilai ujian: 92
Anda lulus!

3.3.3 Struktur Seleksi Tersarang

Kita dapat menuliskan struktur seleksi di dalam struktur seleksi. Misalkan kita membuat program kelulusan yang menentukan kelulusan berdasarkan dua nilai ujian (ujian1 dan ujian2) dengan struktur seleksi: Jika nilai ujian1 > 80 dan ujian2 > 75 maka program menampilkan "Anda lulus!" dan jika kedua kondisi tidak terpenuhi maka program menampilkan "Anda tidak lulus!".

Kita dapat menuliskan struktur seleksi dari program tersebut seperti berikut:

```
if ujian1 > 80:
    if ujian2 > 75:
        print('Anda lulus!')
    else:
        print('Anda tidak lulus!')
else:
    print('Anda tidak lulus!')
```

Penulisan struktur seleksi di dalam struktur seleksi seperti di atas disebut dengan struktur seleksi tersarang (*nested*). Berikut ini kode program lengkap dari program kelulusan dengan kondisi dua nilai ujian:

```
# kelulusan_dua_ujian.py
# Program ini meminta pengguna memasukkan dua nilai ujian (ujian1 dan ujian2)
# dan menentukan kelulusan berdasarkan kondisi kedua nilai ujian tersebut

def main():
    # Ambil dua nilai ujian
    ujian1 = float(input('Masukkan nilai ujian1: '))
    ujian2 = float(input('Masukkan nilai ujian2: '))

    # Tampilkan "Anda lulus" jika ujian1 > 80 dan ujian2 > 75
    # dan tampilkan "Anda tidak lulus" jika lainnya
    if ujian1 > 80:
        if ujian2 > 75:
            print('Anda lulus!')
        else:
            print('Anda tidak lulus!')
    else:
        print('Anda tidak lulus!')

# Panggil fungsi main
main()
```

Berikut adalah contoh-contoh output dengan berbagai kombinasi input:

Output 1:

```
Masukkan nilai ujian1: 88
Masukkan nilai ujian2: 80
Anda lulus!
```

Output 2:

```
Masukkan nilai ujian1: 85
Masukkan nilai ujian2: 60
Anda tidak lulus!
```

Output 3:

```
Masukkan nilai ujian1: 78
Masukkan nilai ujian2: 70
Anda tidak lulus!
```

Output 4:

```
Masukkan nilai ujian1: 75
Masukkan nilai ujian2: 80
Anda tidak lulus!
```

Contoh: Menampilkan Nilai Huruf Berdasarkan Skala Nilai

Persoalan lain yang dapat diselesaikan menggunakan statement `if` tersarang adalah persoalan untuk menampilkan nilai huruf berdasarkan skala nilai seperti tabel di bawah ini:

Nilai Ujian	Nilai Huruf
≥ 90	A
80 - 90	B
70 - 80	C
60 - 70	D
≤ 60	E

Program berikut adalah program untuk menyelesaikan persoalan menampilkan nilai huruf berdasarkan skala nilai:

```
# nilai_huruf.py
# Program yang menampilkan nilai huruf dengan ketentuan:
# A:  $\geq 90$ 
# B: 80 - 90 (tidak termasuk 90)
# C: 70 - 80 (tidak termasuk 80)
# D: 60 - 70 (tidak termasuk 70)
# E:  $< 60$ 

def main():
    # Ambil nilai ujian
    nilai = float(input('Masukkan nilai ujian: '))

    # Tampilkan nilai huruf berdasarkan skala
    if nilai  $\geq 90$ :
        print('A')
    else:
        if nilai  $\geq 80$ :
            print('B')
        else:
```

```

    if nilai >= 70:
        print('C')
    else:
        if nilai >= 60:
            print('D')
        else:
            print('E')

# Panggil fungsi main
main()

```

Berikut adalah contoh-contoh output dari program `nilai_huruf.py` di atas:

Output 1:

```

Masukkan nilai ujian: 88
B

```

Output 2:

```

Masukkan nilai ujian: 60
D

```

Perhatikan bagian struktur seleksi dari program di atas:

```

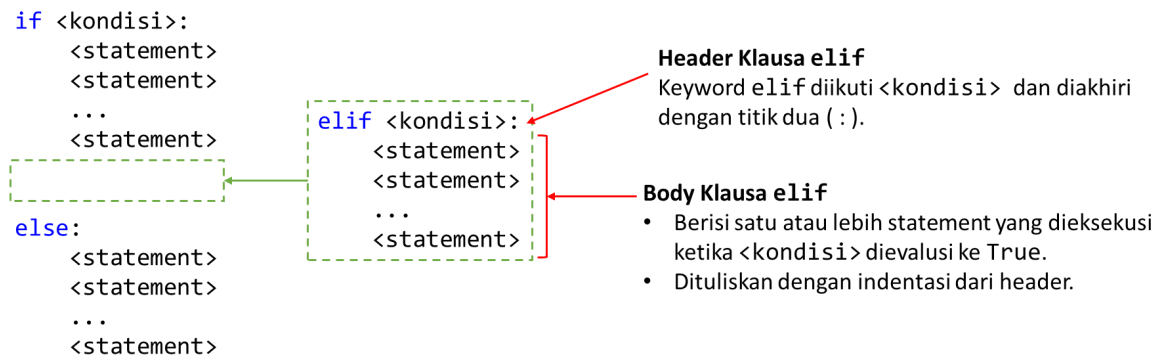
if nilai >= 90:
    print('A')
else:
    if nilai >= 80:
        print('B')
    else:
        if nilai >= 70:
            print('C')
        else:
            if nilai >= 60:
                print('D')
            else:
                print('E')

```

Pada struktur seleksi tersarang di atas kita menguji rangkaian kondisi-kondisi. Namun, penulisan struktur seleksi tersarang ini membuat kode sulit dibaca. Untuk menyederhanakan penulisan struktur seleksi yang menguji rangkaian kondisi-kondisi kita dapat menggunakan statement `if-elif-else`.

3.3.4 Statement `if-elif-else`

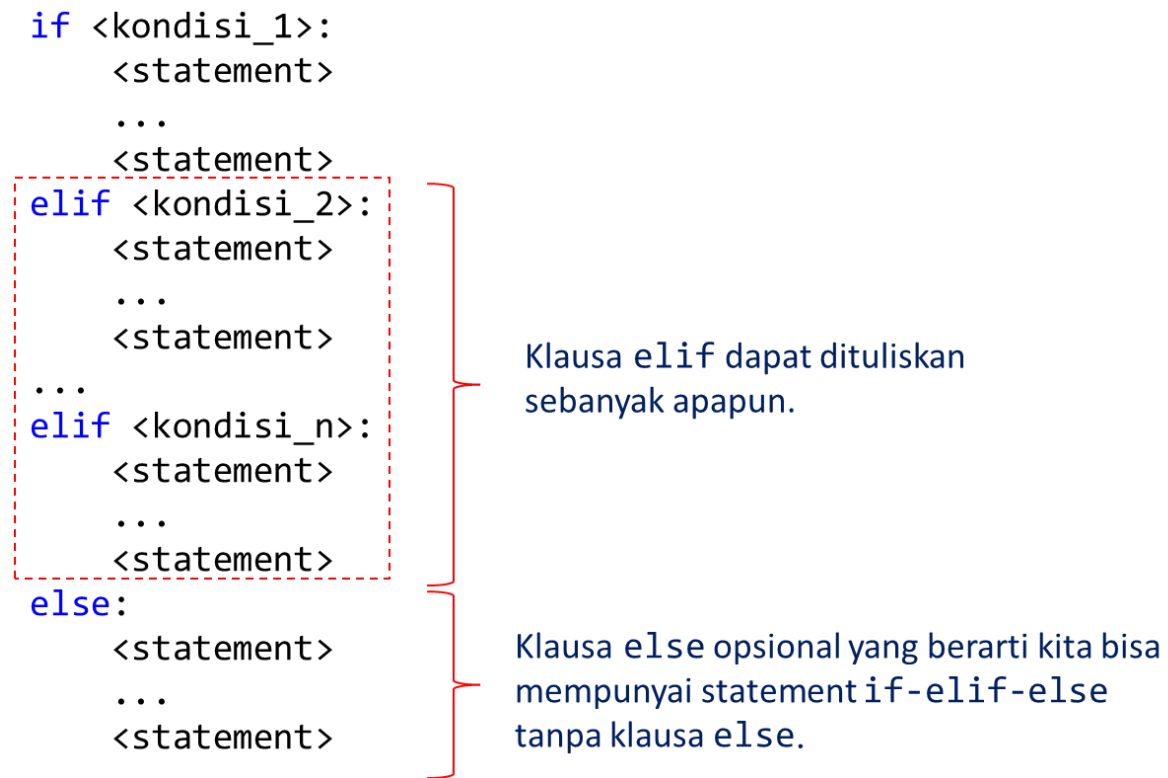
Python menyediakan struktur seleksi yang digunakan untuk menguji rangkaian kondisi-kondisi yang mutually exclusive yaitu statement `if-elif-else`. Statement `if-elif-else` dibentuk dengan menambahkan klausa `elif` (singkatan dari `else if`) di antara `if` dan `else`:



Berikut ini bentuk umum penulisan statement `if-elif-else`:

```
if <kondisi_1>:
    <statement>
    ...
    <statement>
elif <kondisi_2>:
    <statement>
    ...
    <statement>
...
elif <kondisi_n>:
    <statement>
    ...
    <statement>
else:
    <statement>
    ...
    <statement>
```

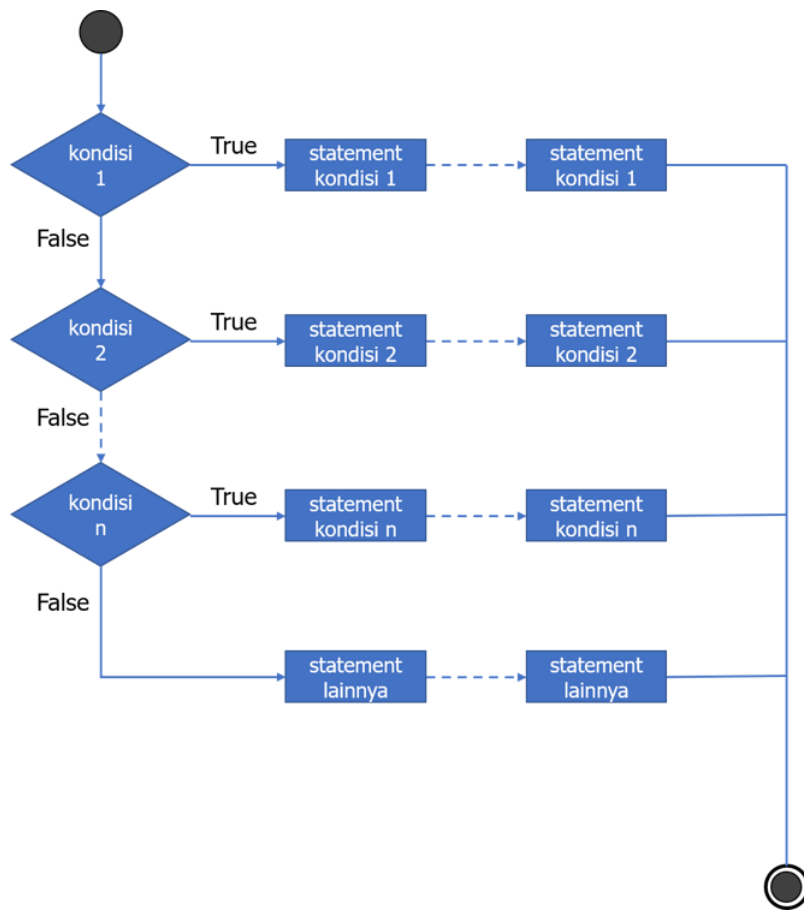
Kita dapat menuliskan sebanyak apapun klausa `elif`. Sedangkan klausa `else` adalah opsional, yang berarti kita bisa mempunyai statement `if-elif-else` tanpa klausa `else`:



Statement `if-elif-else` dieksekusi seperti berikut:

- Interpreter menguji `<kondisi_1>` terlebih dahulu. Jika `<kondisi_1>` bernilai `True` maka statement-statement di dalam klausa `if` dieksekusi lalu program keluar dari struktur `if-elif-else`. Jika `<kondisi_1>` bernilai `False` maka interpreter melanjutkan pengujian kondisi pada klausa `elif` pertama, `<kondisi_2>`.
- Interpreter melanjutkan pengujian kondisi berikutnya. Jika `<kondisi_2>` bernilai `True` maka statement-statement di dalam klausa `elif` pertama ini dieksekusi lalu program keluar dari struktur `if-elif-else`. Namun, jika `<kondisi_2>` bernilai `False`, maka interpreter melanjutkan pengujian kondisi pada klausa `elif` berikutnya.
- Proses pengujian ini berlanjut sampai dengan klausa `elif` terakhir. Jika kondisi pada klausa `elif` terakhir (`<kondisi_n>`) bernilai `True` maka statement-statement dalam klausa `elif` terakhir ini dieksekusi dan program keluar dari struktur `if-elif-else`. Namun, jika kondisi tersebut bernilai `False` maka statement-statement di dalam klausa `else` dieksekusi.

Gambar di bawah ini adalah flowchart dari statement `if-elif-else`:



Program yang menampilkan nilai huruf berdasarkan suatu skala nilai yang dituliskan menggunakan struktur seleksi tersarang, dapat dituliskan menggunakan statement `if-elif-else` menjadi seperti berikut:

```
# nilai_huruf_elif.py
# Program ini menampilkan nilai huruf berdasarkan
# skala nilai menggunakan statement if-elif-else

def main():
    nilai = float(input("Masukkan nilai: "))

    # Tampilkan nilai huruf berdasarkan skala
    if nilai >= 90:
        print('A')
    elif nilai >= 80:
        print('B')
    elif nilai >= 70:
        print('C')
    elif nilai >= 60:
        print('D')
    else:
        print('E')

main()
```

Berikut adalah contoh output dari program `nilai_huruf_elif.py` di atas:

Output 1:

```
Masukkan nilai: 88
B
```

Output 2:

```
Masukkan nilai: 55
E
```

Klausula `else` pada statement `if-elif-else` adalah opsional yang berarti kita dapat mempunyai statement `if-elif-else` tanpa klausula `else`. Sebagai contoh, kita bisa menuliskan program kita sebelumnya tanpa klausula `else` seperti berikut:

```
# nilai_huruf_elif_2.py
# Program ini menampilkan nilai huruf berdasarkan
# skala nilai menggunakan statement if-elif-else tanpa klausula else

def main():
    nilai = float(input("Masukkan nilai: "))

    # Tampilkan nilai huruf berdasarkan skala
    if nilai >= 90:
        print('A')
    elif nilai >= 80:
        print('B')
    elif nilai >= 70:
        print('C')
    elif nilai >= 60:
        print('D')
    elif nilai < 60:
        print('E')

main()
```

Perhatikan pada kode program di atas, kita tidak menuliskan klausula `else`. Contoh output dari program di atas adalah sebagai berikut:

Output 1:

```
Masukkan nilai: 45
E
```

Output 2:

```
Masukkan nilai: 88
B
```

3.4 Operator Logis

Python menyediakan operator logis (disebut juga sebagai operator Boolean) yang dapat digunakan untuk membentuk kondisi dengan ekspresi Boolean yang kompleks. Terdapat tiga operator logis dalam Python yaitu: operator `and`, `or`, dan `not`.

Table berikut menjelaskan operator `and`, `or`, dan `not` dan contoh ekspresi yang menggunakan operator tersebut:

Operator	Arti	Contoh Ekspresi	Arti Ekspresi
<code>and</code>	Mengkombinasikan dua ekspresi Boolean menjadi satu gabungan ekspresi. Kedua subekspresi harus <code>True</code> agar gabungan ekspresi <code>True</code> .	<code>x > y and a < b</code>	Apakah x lebih besar dari y DAN apakah a lebih kecil dari b?
<code>or</code>	Mengkombinasikan dua ekspresi Boolean menjadi satu gabungan ekspresi. Salah satu atau kedua subekspresi harus <code>True</code> agar gabungan ekspresi <code>True</code> .	<code>x == y or a == b</code>	Apakah x sama dengan y ATAU apakah a sama dengan b?
<code>not</code>	Merupakan operator unary (membutuhkan hanya satu operand). Operator ini membalik nilai kebenaran dari operand. Jika operand bernilai <code>True</code> , maka operator ini menjadikannya bernilai <code>False</code> , dan sebaliknya.	<code>not (x > y)</code>	Apakah ekspresi <code>x > y</code> TIDAK benar?

3.4.1 Operator `and`

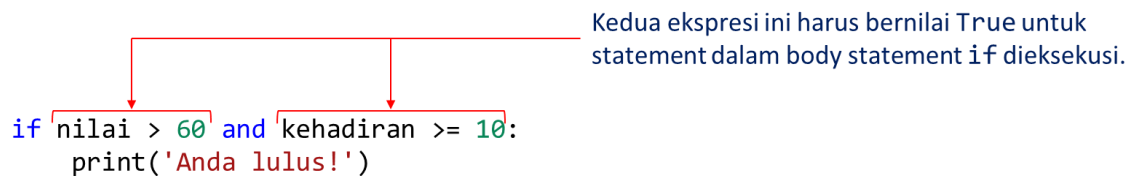
Operator `and` bekerja menurut tabel kebenaran berikut:

eksp1	eksp2	eksp1 and eksp2
<code>True</code>	<code>True</code>	<code>True</code>
<code>True</code>	<code>False</code>	<code>False</code>
<code>False</code>	<code>True</code>	<code>False</code>
<code>False</code>	<code>False</code>	<code>False</code>

Gabungan dua ekspresi dengan operator `and` hanya bernilai `True` jika kedua ekspresi bernilai `True`. Sebagai contoh:

```
if nilai > 60 and kehadiran >= 10:
    print('Anda lulus!')
```

Kedua ekspresi, `nilai > 60` dan `kehadiran >= 10`, harus bernilai `True` untuk statement dalam body statement `if` dieksekusi:



```
if nilai > 60 and kehadiran >= 10:
    print('Anda lulus!')
```

Berikut adalah sesi interaktif yang mendemonstrasikan ekspresi-ekspresi yang menggunakan operator `and`:

```
>>> x = 5
>>> y = 10
>>> z = 20
>>> x < y and y < z
True
>>> x < y and y > z
False
>>> x > y and y < z
False
>>> x > y and y > z
False
```

3.4.2 Operator `or`

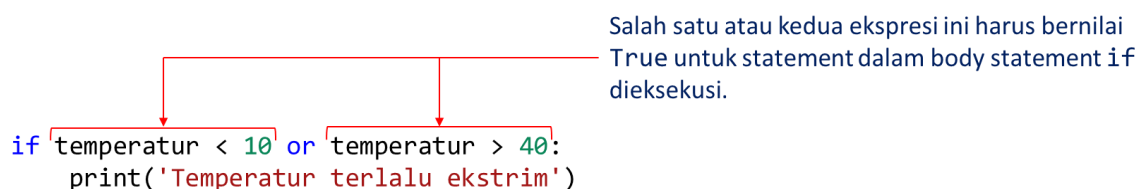
Operator `or` bekerja menurut tabel kebenaran berikut:

eksp1	eksp2	eksp1 and eksp2
True	True	True
True	False	True
False	True	True
False	False	False

Gabungan dua ekspresi dengan operator `or` bernilai `True` jika salah satu atau kedua ekspresi bernilai `True`. Berikut adalah contoh penggunaan operator `or`:

```
if temperatur < 10 or temperatur > 40:
    print('Temperatur terlalu ekstrim')
```

Salah satu atau kedua ekspresi pada kondisi statement `if` di atas harus bernilai `True` untuk statement dalam body statement `if` dieksekusi:



```
if temperatur < 10 or temperatur > 40:
    print('Temperatur terlalu ekstrim')
```

Berikut adalah sesi interaktif yang mendemonstrasikan ekspresi-ekspresi yang menggunakan operator `or`:

```
>>> x = 5
>>> y = 10
>>> z = 20
>>> x < y or y < z
True
>>> x < y or y > z
True
>>> x > y or y < z
True
>>> x > y or y > z
False
```

3.4.3 Operator not

Operator `not` memerlukan satu operand berupa ekspresi Boolean dan bekerja dengan membalik nilai kebenaran dari operand tersebut.


Operator `not` bekerja menurut tabel kebenaran berikut:

eksp	not eksp
True	False
False	True

Berikut adalah contoh penerapan operator `not`:

```
if not (temperatur > 100):
    print('Air tidak mendidih.')
```

Ekspresi `temperatur > 100` pada statement `if` di atas harus bernilai `False` untuk statement dalam body statement `if` dieksekusi:

 Ekspresi ini harus bernilai `False` untuk statement dalam body statement `if` dieksekusi.

```
if not (temperatur > 100):
    print('Air tidak mendidih.')
```

3.4.4 Menggunakan Operator Logis

Pada bagian ini kita akan membahas contoh-contoh penggunaan operator logis.

Contoh 1: Program Kelulusan Berdasarkan Dua Nilai Ujian

Program kelulusan yang menguji dua ujian pada contoh sebelumnya pada pembahasan struktur seleksi tersarang:

```

if ujian1 > 80:
    if ujian2 > 75:
        print('Anda lulus!')
    else:
        print('Anda tidak lulus!')
else:
    print('Anda tidak lulus!')

```

Program di atas dapat dituliskan ulang menggunakan operator logis `and` seperti berikut:

```

if ujian1 > 80 and ujian2 > 75:
    print('Anda lulus!')
else:
    print('Anda tidak lulus!')

```

Berikut adalah kode lengkap program kelulusan dua ujian menggunakan operator `and`:

```

# kelulusan_dua_ujian2.py
# Program ini meminta pengguna memasukkan dua nilai ujian (ujian1 dan ujian2)
# dan menentukan kelulusan berdasarkan kondisi kedua nilai ujian tersebut.
# Kondisi pada struktur seleksi menggunakan operator and.

def main():
    # Ambil dua nilai ujian
    ujian1 = float(input('Masukkan nilai ujian1: '))
    ujian2 = float(input('Masukkan nilai ujian2: '))

    # Tampilkan "Anda lulus" jika ujian1 > 80 dan ujian2 > 75
    # dan tampilkan "Anda tidak lulus" jika lainnya
    if ujian1 > 80 and ujian2 > 75:
        print('Anda lulus!')
    else:
        print('Anda tidak lulus!')

# Panggil fungsi main
main()

```

Output 1:

```

Masukkan nilai ujian1: 88
Masukkan nilai ujian2: 80
Anda lulus!

```

Output 2:

```

Masukkan nilai ujian1: 85
Masukkan nilai ujian2: 60
Anda tidak lulus!

```

Output 3:

Masukkan nilai ujian1: 78
Masukkan nilai ujian2: 70
Anda tidak lulus!

Output 4:

Masukkan nilai ujian1: 75
Masukkan nilai ujian2: 80
Anda tidak lulus!

Contoh 2: Menentukan Tingkat Risiko Penyakit Jantung Berdasarkan IMT

Misalkan kita membuat sebuah program yang menginformasikan tingkat risiko penyakit jantung seseorang menggunakan aturan berdasarkan umur dan Indeks Massa Tubuh (IMT) seperti pada tabel berikut:

		Umur	
		< 45	≥ 45
IMT	< 22.0	Rendah	Sedang
	≥ 22.0	Sedang	Tinggi

Indeks Massa Tubuh (IMT) dihitung rumus berikut:

$$IMT = \frac{BeratBadan(kg)}{TinggiBadan(m) \times TinggiBadan(m)}$$

Struktur seleksi dari program dapat kita tuliskan seperti berikut:

```
if umur < 45 and imt < 22.0:  
    print('Tingkat resiko penyakit jantung Anda: RENDAH')  
elif (umur < 45 and imt >= 22.0) or (umur >= 45 and imt < 22.0):  
    print('Tingkat resiko penyakit jantung Anda: SEDANG')  
else:  
    print('Tingkat resiko penyakit jantung Anda: TINGGI')
```

Kode program lengkap yang mengkalkulasi tingkat resiko penyakit jantung seseorang dapat dituliskan seperti berikut:

```
# resiko_jantung.py  
# Program ini menentukan tingkat resiko jantung seseorang  
# berdasarkan umur dan indeks massa tubuh  
  
def main():  
    # Ambil umur, berat badan, dan tinggi badan  
    umur = int(input('Masukkan umur: '))  
    bb = int(input('Masukkan berat badan (kg): '))  
    tb = int(input('Masukkan tinggi badan (cm): '))  
  
    # Kalkulasi imt  
    imt = bb / ((tb/100) * (tb/100))
```

```

# Tentukan resiko penyakit jantung berdasarkan umur dan imt
if umur < 45 and imt < 22.0:
    print('Tingkat resiko penyakit jantung Anda: RENDAH')
elif (umur < 45 and imt >= 22.0) or (umur >= 45 and imt < 22.0):
    print('Tingkat resiko penyakit jantung Anda: SEDANG')
else:
    print('Tingkat resiko penyakit jantung Anda: TINGGI')

# Panggil fungsi main
main()

```

Berikut adalah contoh output dari program `resiko_jantung.py` di atas:

```

Masukkan umur: 22
Masukkan berat badan (kg): 90
Masukkan tinggi badan (cm): 165
Tingkat resiko penyakit jantung Anda: SEDANG

```

Untuk membuat program lebih mudah dibaca, kita dapat menyimpan kondisi dalam variabel. Sebagai contoh, pada program sebelumnya dapat kita dapat menyimpan kondisi `umur < 45` dan `imt < 22.0` dalam variabel seperti berikut:

```

muda = umur < 45
kurus = imt < 22.0

```

Sehingga struktur seleksi program sebelumnya dapat kita tulis ulang sebagai berikut:

```

if muda and kurus:
    print('Tingkat resiko penyakit jantung Anda: RENDAH')
elif (muda and not kurus) or (not muda and kurus):
    print('Tingkat resiko penyakit jantung Anda: SEDANG')
else:
    print('Tingkat resiko penyakit jantung Anda: TINGGI')

```

Dengan menyimpan kondisi dalam variabel maka program lebih mudah dibaca dan penulisan kondisi dapat kita persingkat. Berikut adalah kode alternatif dari kode program resiko penyakit jantung:

```

# resiko_jantung2.py
# Program ini menentukan tingkat resiko jantung seseorang
# berdasarkan umur dan indeks massa tubuh

def main():
    # Ambil umur, berat badan, dan tinggi badan
    umur = int(input('Masukkan umur: '))
    bb = int(input('Masukkan berat badan (kg): '))
    tb = int(input('Masukkan tinggi badan (cm): '))

    # Kalkulasi imt
    imt = bb / ((tb/100) * (tb/100))

    # Simpan kondisi dalam variabel
    muda = umur < 45

```

```

kurus = imt < 22.0

# Tentukan resiko penyakit jantung berdasarkan umur dan imt
if muda and kurus:
    print('Tingkat resiko penyakit jantung Anda: RENDAH')
elif (muda and not kurus) or (not muda and kurus):
    print('Tingkat resiko penyakit jantung Anda: SEDANG')
else:
    print('Tingkat resiko penyakit jantung Anda: TINGGI')

# Panggil fungsi main
main()

```

Berikut adalah contoh output dari program `resiko_jantung2.py` di atas:

```

Masukkan umur: 25
Masukkan berat badan (kg): 95
Masukkan tinggi badan (cm): 165
Tingkat resiko penyakit jantung Anda: SEDANG

```

3.4.5 Evaluasi *Short Circuit*

Gabungan ekspresi Boolean dengan operator `and` dan `or` dievaluasi dari kiri ke kanan. Sebagai contoh, misalkan terdapat ekspresi seperti berikut:

```

x >= 2 and x/y > 2

```

Interpreter mengevaluasi ekspresi di atas dengan mengevaluasi ekspresi pada ruas kiri, `x >= 2`, terlebih dahulu. Setelah mengevaluasi ekspresi pada ruas kiri, interpreter melanjutkan dengan mengevaluasi ekspresi pada ruas kanan, `x/y > 2`. Lalu, kedua nilai Boolean dari hasil evaluasi dievaluasi berdasarkan tabel kebenaran `and`. Namun, jika dari evaluasi ekspresi pada ruas kiri sudah dapat disimpulkan nilai dari gabungan ekspresi Boolean, maka ekspresi pada ruas kanan tidak dievaluasi. Evaluasi sebagian ini disebut dengan evaluasi **short-circuit**.

Evaluasi **short-circuit** terjadi dalam dua kasus berikut:

- Untuk gabungan ekspresi dengan operator `and`, jika hasil ekspresi ruas kiri bernilai `False`, maka ekspresi ruas kanan tidak dievaluasi karena gabungan ekspresi Boolean pasti bernilai `False`.
- Untuk gabungan ekspresi dengan operator `or`, jika hasil ekspresi ruas kiri bernilai `True`, maka ekspresi ruas kanan tidak dievaluasi karena gabungan ekspresi Boolean pasti bernilai `True`.

Untuk melihat terjadinya evaluasi short-circuit, perhatikan ekspresi berikut:

```

x >= 2 and x/y > 2

```

Jika ekspresi pada ruas kiri, `x >= 2`, dievaluasi ke `False`, maka proses evaluasi berhenti disini dan gabungan ekspresi menghasilkan `False`. Sesi interaktif berikut memperlihatkan terjadinya evaluasi short-circuit:

```
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
```

Pada sesi interaktif di atas, ekspresi ruas kanan `(x/y) > 2` tidak dievaluasi, karena ekspresi ruas kiri `x >= 2` menghasilkan `False` sehingga sudah dapat disimpulkan keseluruhan ekspresi menghasilkan `False`. Jika, misalkan, ruas kanan dievaluasi maka keseluruhan ekspresi akan menghasilkan error, karena ekspresi `x/y` membagi dengan 0 yang menghasilkan error.

Sesi interaktif berikut mendemonstrasikan contoh kasus tidak terjadinya evaluasi short-circuit pada evaluasi ekspresi di atas:

```
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<pyshe11#298>", line 1, in <module>
    x >= 2 and (x/y) > 2
ZeroDivisionError: division by zero
```

Evaluasi ruas kiri adalah `True`, maka hasil evaluasi gabungan ekspresi tidak dapat disimpulkan hanya dari evaluasi ruas kiri. Karena itu, ruas kanan dievaluasi. Dan karena ekspresi ruas kanan, `x/y > 2`, dengan `y = 0` melakukan pembagian dengan 0, maka evaluasi ruas kanan tersebut menghasilkan error.

3.5 Struktur Iterasi

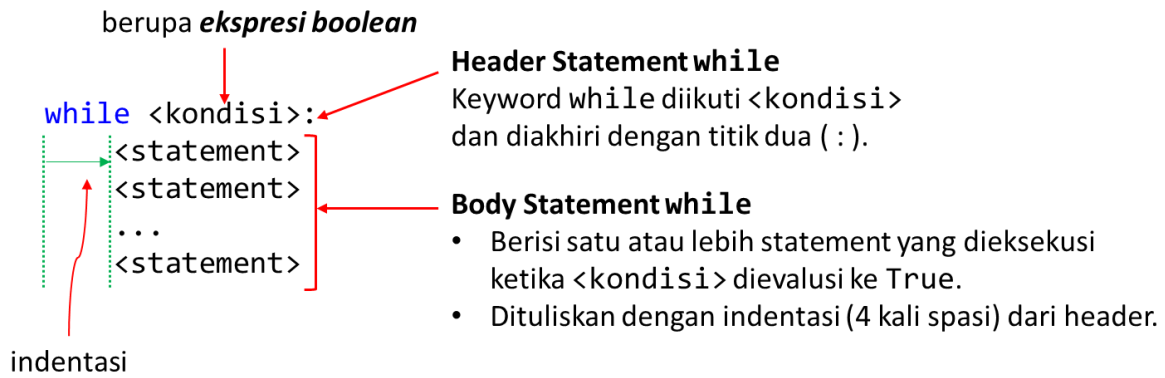
Struktur iterasi atau perulangan memungkinkan sebuah program untuk mengeksekusi sekumpulan statement-statement secara berulang kali. Struktur iterasi sering disebut dengan **loop**. Python menyediakan dua statement loop yaitu loop `while` dan loop `for`. Loop `while` digunakan untuk melakukan iterasi berdasarkan kondisi sedangkan loop `for` digunakan untuk melakukan iterasi berdasarkan jumlah tertentu.

3.5.1 Loop `while`

Statement loop `while` digunakan untuk membuat iterasi berdasarkan suatu kondisi. Penulisan statement loop `while` mirip dengan penulisan statement `if`, hanya saja pada `loop while` kita menggunakan keyword `while` menggantikan keyword `if`. Bentuk umum syntax penulisan statement `while` adalah sebagai berikut:

```
while <kondisi>:
    <statement>
    <statement>
    ...
    <statement>
```

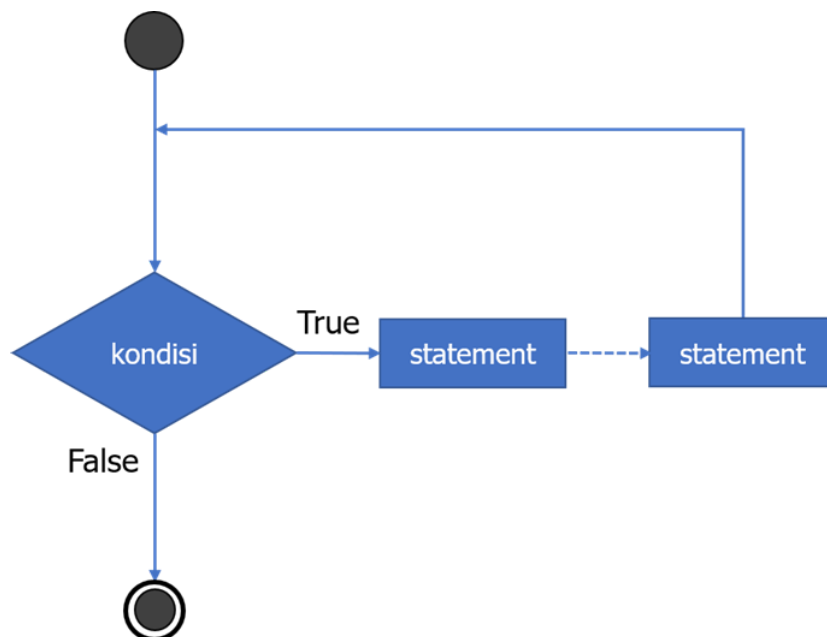
Gambar berikut menjelaskan bagian-bagian dari syntax statement `while`:



Statement loop `while` bekerja seperti berikut:

- Eksekusi loop dimulai dengan menguji `<kondisi>`, jika menghasilkan `False` maka loop berhenti dan jika menghasilkan `True` maka statement-statement di dalam body `while` dieksekusi.
- Setelah selesai mengeksekusi statetement-statement di dalam body `while`, loop berulang dengan menguji `<kondisi>` kembali dan jika menghasilkan `False` maka loop berhenti, dan jika menghasilkan `True`, maka statement-statement di dalam body kembali dieksekusi.
- Loop terus berulang sampai dengan hasil pengujian `<kondisi>` bernilai `False`.

Flowchart dari statement loop `while` dapat dilihat pada gambar berikut:



Kondisi pada statement loop `while` berbentuk ekspresi Boolean dan umumnya menguji nilai dari satu atau lebih variabel, sehingga umumnya sebelum statement loop `while` kita menuliskan statemet-statement yang menginisialisasi variabel-variabel yang diuji nilainya di dalam kondisi. Kemudian, salah satu dari statement-statement dalam body loop harus mengubah nilai dari satu atau lebih variabel yang diuji sehingga pada akhirnya menyebabkan kondisi dievaluasi ke `False` dan menghentikan loop.

Berikut adalah contoh program dengan statement `while`:

```
# tujuh_hello.py
# Menampilkan hello sebanyak 7 kali
# menggunakan loop while

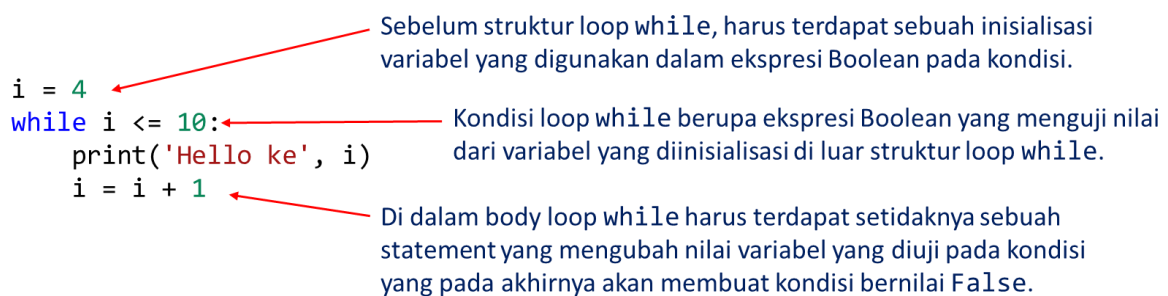
def main():
    i = 4
    while i <= 10:
        print('Hello ke', i)
        i = i + 1

main()
```

Output dari program `tujuh_hello.py` di atas:

```
Hello ke 4
Hello ke 5
Hello ke 6
Hello ke 7
Hello ke 8
Hello ke 9
Hello ke 10
```

Perhatikan pada kode di atas, sebelum statement loop `while`, kita mendefinisikan sebuah variabel `i` dan menginisiasikannya dengan nilai 4 (statement baris 6). Variabel `i` ini kita gunakan sebagai variabel yang diuji nilainya pada kondisi dari statement loop `while`, `i <= 10`. Di dalam body statement loop `while` di atas, selain terdapat statement yang mencetak teks `Hello ke` yang diikuti nilai variabel `i`, kita mempunyai statement pada baris 9 yang menginkrementasi variabel `i` dengan nilai 1, `i = i + 1`. Statement ini mengubah nilai variabel `i` sehingga pada akhirnya akan membuat kondisi bernilai `False` dan menghentikan loop. Gambar berikut menjelaskan struktur loop `while` contoh di atas:



The diagram shows the code snippet from the previous block with three red arrows pointing to specific parts, each with a text annotation:

- Arrow 1 points to `i = 4`: "Sebelum struktur loop while, harus terdapat sebuah inisialisasi variabel yang digunakan dalam ekspresi Boolean pada kondisi." (Before the while loop structure, there must be an initialization of a variable used in the Boolean expression of the condition.)
- Arrow 2 points to `while i <= 10:`: "Kondisi loop while berupa ekspresi Boolean yang menguji nilai dari variabel yang diinisialisasi di luar struktur loop while." (The while loop condition is a Boolean expression that tests the value of the variable initialized outside the while loop structure.)
- Arrow 3 points to `i = i + 1`: "Di dalam body loop while harus terdapat setidaknya sebuah statement yang mengubah nilai variabel yang diuji pada kondisi yang pada akhirnya akan membuat kondisi bernilai False." (Inside the while loop body, there must be at least one statement that changes the value of the variable tested in the condition, which will eventually make the condition False.)

Infinite Loop

Apa yang terjadi jika kita tidak mempunyai statement di dalam body loop `while` yang dapat menghentikan loop? Jika tidak terdapat statement yang mengubah nilai variabel yang diuji nilainya sehingga hasil pengujian menghasilkan `False`, maka loop akan berjalan terus menerus dan tidak akan pernah berakhir. Loop tanpa akhir ini disebut dengan ***infinite loop***.

Berikut adalah contoh kode yang menghasilkan ***infinite loop***:

```
i = 4
while i <= 10:
    print('Hello ke', i)
```

Pada struktur `while` di atas, nilai variabel `i` akan selalu sama dengan 4, sehingga kondisi, `i <= 10` akan selalu bernilai `True`. Kode di atas akan menghasilkan infinite loop karena tidak ada statement dalam body loop `while` yang membuat kondisi menjadi `False`.

Catatan: Untuk menghentikan infinite loop, kita harus menginterupsi/menghentikan paksa program dengan menekan CTRL+C.

Selalu hindari infinite loop. Hanya dalam kasus tertentu yang jarang sekali kita memerlukan infinite loop.

3.5.2 Loop `for`

Statement loop `for` digunakan untuk melakukan perulangan dengan jumlah tertentu. Misalkan kita ingin menampilkan "Hello, world!" sebanyak lima kali, kita dapat menuliskan statement `for` seperti berikut:

```
for count in range(5):  
    print('Hello, world!')
```

Statement loop `for` di atas memberikan output seperti berikut:

```
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!
```

Statement loop `for` memerlukan:

- Keyword `for` dan keyword `in`;
- Sebuah variabel yang digunakan sebagai counter (pencacah). Pada contoh variabel `count` adalah counter;
- Sebuah barisan nilai yang digunakan untuk memberikan nilai ke variabel counter. Pada contoh fungsi built-in `range(5)` digunakan untuk menggenerasi barisan nilai 0, 1, 2, 3, 4;
- Statement-statement yang dieksekusi berulang yang dituliskan di dalam body statement `for`.

Statement `for` bekerja sebagai berikut:

- Loop dimulai dengan variabel counter ditugaskan dengan nilai pertama dari barisan nilai, lalu statement-statement dalam body loop dieksekusi
- Setelah itu variabel counter ditugaskan dengan nilai kedua dari barisan nilai, dan statement-statement dalam body loop kembali dieksekusi.
- Hal ini berlanjut sampai dengan variabel counter menerima nilai terakhir dari barisan nilai.

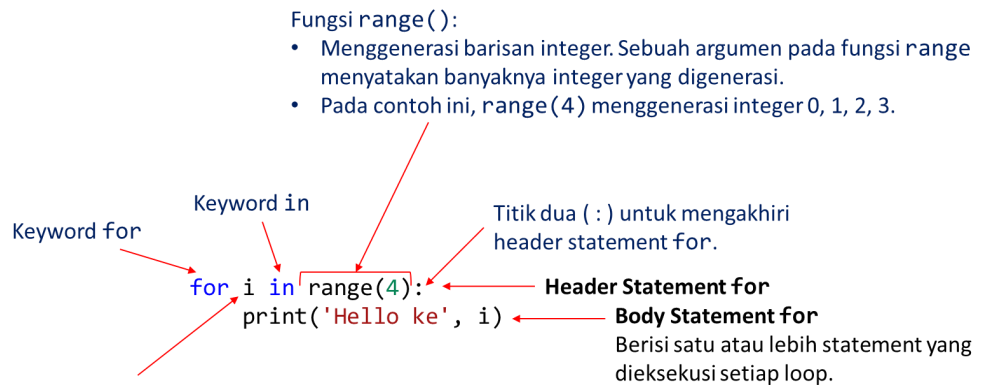
Loop `for` berikut memperlihatkan perubahan nilai variabel counter:

```
for i in range(4):  
    print('Hello, world! ke', i)
```

Statement loop `for` di atas memberikan output:

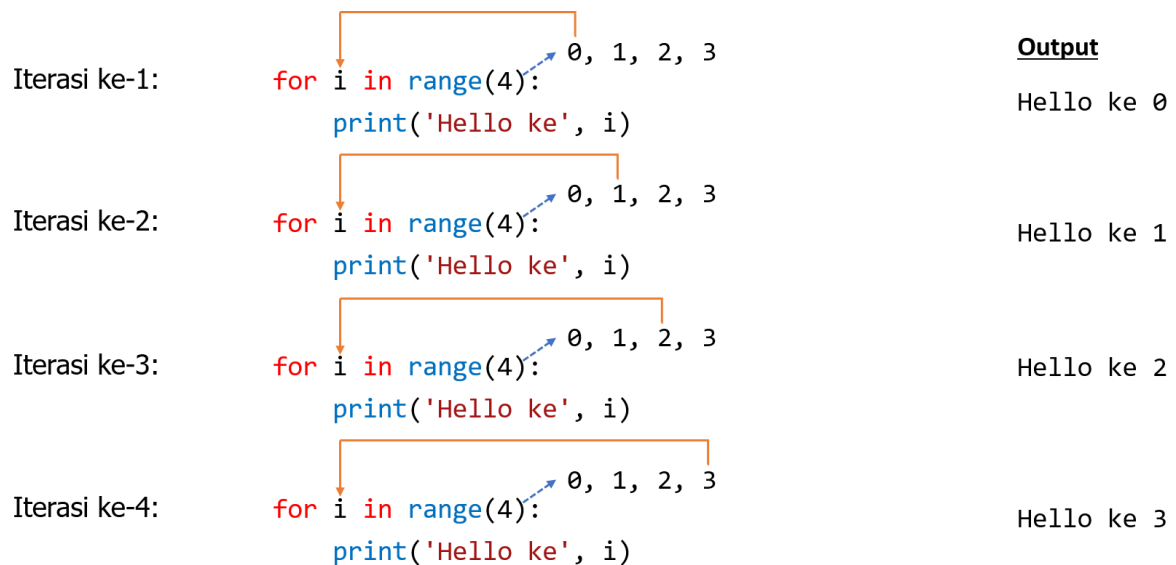
```
Hello ke 0
Hello ke 1
Hello ke 2
Hello ke 3
```

Gambar berikut menjelaskan bagian-bagian dari statement loop `for` pada contoh di atas:



Loop `for` memerlukan sebuah variabel yang pada setiap loopnya ditugaskan dengan masing-masing nilai pada barisan nilai.
Pada contoh ini, variabel `i` ditugaskan dengan barisan nilai yang digenerasi oleh fungsi `range` dimulai dari nilai pertama pada loop pertama, nilai kedua pada loop kedua, dan seterusnya.

Gambar berikut mengilustrasikan tahapan iterasi loop `for` di atas:



Fungsi `range`

Fungsi `range` digunakan untuk menghasilkan barisan bilangan dalam *range* (jangkauan) tertentu. Fungsi `range` dapat menerima satu argumen, dua argumen, atau tiga argumen.

Fungsi `range` dengan satu argumen dituliskan dengan syntax berikut:

```
range(<stop>)
```

Fungsi `range` dengan satu argumen menghasilkan integer dari 0 sampai dengan `<stop> - 1`. Sebagai contoh, `range(10)` menghasilkan 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Fungsi `range` dengan dua argumen dituliskan dengan syntax berikut:

```
range(<start>, <stop>)
```

Fungsi `range` dengan dua argumen menghasilkan barisan integer dari `<start>` sampai dengan `<stop> - 1` dengan inkrementasi 1. Sebagai contoh, `range(2, 10)` menghasilkan 2, 3, 4, 5, 6, 7, 8, 9.

Fungsi `range` dengan tiga argumen dituliskan dengan syntax berikut:

```
range(<start>, <stop>, <step>)
```

Fungsi `range` dengan tiga argument ini menghasilkan barisan integer dari `<start>` sampai dengan `<stop> - 1` dengan inkrementasi `<step>`. Sebagai contoh, `range(1, 10, 2)` menghasilkan 1, 3, 5, 7, 9.

Kode berikut adalah contoh loop `for` dengan fungsi `range()` dua argumen:

```
for i in range(0, 5):  
    print(i)
```

Kode di atas akan menghasilkan output:

```
0  
1  
2  
3  
4
```

Kode berikut adalah contoh loop `for` dengan fungsi `range()` tiga argumen:

```
for j in range(1, 10, 2):  
    print(j)
```

Kode di atas menghasilkan output:

```
1  
3  
5  
7  
9
```

Bentuk Umum Penulisan Loop `for`

Secara umum, penulisan statement loop `for` adalah sebagai berikut:

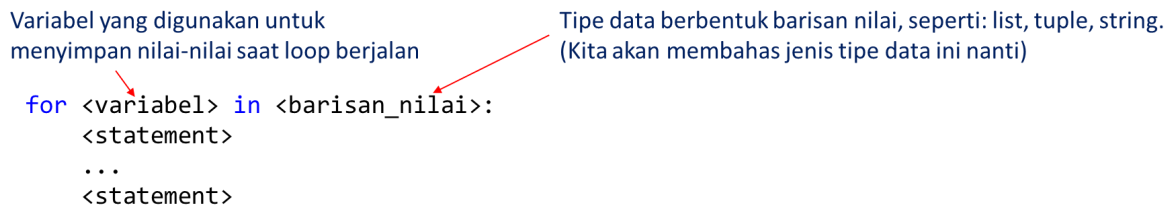
```
for <variabel> in <barisan_nilai>:  
    <statement>  
    ...  
    <statement>
```

`<variabel>` adalah variabel counter yang digunakan untuk menyimpan nilai-nilai saat loop berjalan. Sedangkan `<barisan_nilai>` merupakan tipe data yang berbentuk barisan nilai seperti list, tuple, string.

Gambar berikut menjelaskan bagian-bagian syntax di atas:

Variabel yang digunakan untuk menyimpan nilai-nilai saat loop berjalan

Tipe data berbentuk barisan nilai, seperti: list, tuple, string. (Kita akan membahas jenis tipe data ini nanti)



```
for <variabel> in <barisan_nilai>:  
    <statement>  
    ...  
    <statement>
```

Berikut adalah contoh `loop for` menggunakan barisan nilai (bertipe `list`):

```
for num in [1, 3, 5, 7, 9]:  
    print(num)
```

Kode di atas memberikan output:

```
1  
3  
5  
7  
9
```

Perhatikan pada kode di atas, `[1, 3, 5, 7, 9]` adalah sebuah list. Kita akan membahas list nanti.

Berikut adalah contoh program lain yang menggunakan `loop for`. Program ini menampilkan nilai kuadrat dari 1 s.d 10:

```
# kuadrat.py  
# Program ini menampilkan tabel nilai kuadrat  
# dari 1 sampai dengan 10  
  
def main():  
  
    # Print baris judul  
    print('Angka\tKuadrat')  
    print('-----')  
  
    # Print angka 1 s.d 10  
    # dan nilai kuadratnya  
    for num in range(1, 11):  
        kuadrat = num ** 2
```

```
print(f'{num}\t{kuadrat}')

main()
```

Output dari program `kuadrat.py` di atas:

Angka	Kuadrat
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

3.6 Pola-pola Loop

Pada bagian ini kita akan melihat sejumlah pola-pola loop yang sering digunakan pada program.

3.6.1 Loop yang Menghitung Total Berjalan

Salah satu penerapan loop yang banyak digunakan adalah untuk menghitung total penjumlahan dari serangkaian angka-angka. Program yang menghitung total penjumlahan dari serangkaian angka-angka memerlukan sebuah loop yang mengiterasi setiap angka dan sebuah variabel yang mengakumulasi total jumlah dari angka dengan total sebelumnya pada setiap iterasi. Variabel yang digunakan untuk mengakumulasi total jumlah disebut dengan variabel **akumulator**. Dan karena nilai total diakumulasi setiap iterasi, maka nilai total ini sering disebut sebagai total berjalan.

Misalkan kita ingin menghitung jumlah total bilangan asli dari 1 s.d 10, maka kita dapat menuliskan kode berikut:

```
total = 0
for num in range(1, 11):
    total = total + num
```

Pada baris 1 kode di atas, kita mendefinisikan sebuah variabel bernama `total` dan menginisialisasinya dengan 0. Variabel `total` ini digunakan sebagai variabel akumulator. Baris 2 dan 3 adalah loop `for` mengiterasi barisan bilangan asli dari 1 s.d 10. Setiap iterasinya nilai bilangan asli disimpan dalam variabel `num`. Kemudian di setiap iterasinya nilai `num` ini dijumlahkan ke variabel `total` sebelumnya. Sehingga, saat loop berakhir variabel `total` akan menyimpan nilai 55, yang merupakan jumlah dari $1+2+3+4+5+6+7+8+9+10$.

Kita dapat mengetahui bagaimana loop yang menghitung total berjalan bekerja dengan menelusurinya dengan tabel perubahan nilai variabel seperti tabel berikut:

num	total
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

Pada tabel kita dapat melihat, saat `num = 1`, maka `total` bernilai 1. Saat `num = 2`, `total` bernilai 3 dan seterusnya sampai pada saat `num = 10` yang merupakan iterasi terakhir, nilai `total` akan bernilai 55.

Berikut adalah kode program yang menghitung total berjalan:

```
# loop_total_berjalan.py
# Program ini mendemonstrasikan penggunaan loop
# untuk menghitung total berjalan

def main():
    # Tetapkan variabel akumulator dan inisialisasi
    total = 0

    # Hitung total jumlah angka 1 s.d 10 menggunakan for loop
    for num in range(1, 11):
        total += num

    # Tampilkan total
    print('Total:', total)

# Panggil fungsi main
main()
```

Output dari program di atas:

```
Total: 55
```

Menghitung Bunga Majemuk

Contoh persoalan pemrograman yang memerlukan penerapan loop total berjalan adalah persoalan menghitung bunga majemuk pada rekening deposito seperti persoalan berikut:

Suatu rekening deposito dengan bunga majemuk berkembang pertahunnya dengan rumus berikut:

$$P(t+1)=P(t)+rP(t)$$

dimana t adalah tahun, r adalah bunga deposito per tahun dan $P(t)$ adalah saldo rekening deposito pada tahun t .

Misalkan, suatu rekening deposito dengan bunga 8% dan saldo awal 10.000.000,-, hitung pada tahun berapa saldo deposito menjadi dua kali dari saldo awal!

Perhitungan manual dari persoalan di atas dapat dilakukan dengan menghitung saldo setiap tahunnya dari tahun 1 sampai dengan tahun dimana saldo deposito menjadi lebih besar dari dua kali saldo awal. Kita dapat melakukan perhitungan manual seperti berikut:

Saldo tahun 1:

$$P(1) = P(0) + 0,08P(0) = 10.000.000 + 0,08(10.000.000) = 10.800.000,-$$

Saldo tahun 2:

$$P(2) = P(1) + 0,08P(1) = 10.800.000 + 0,08(10.800.000) = 11.664.000,-$$

... sampai dengan $P(n) > 2 * \text{Rp.}10.000.000$

Program yang menyelesaikan persoalan penghitungan bunga majemuk seperti di atas dapat dituliskan seperti berikut:

```
# deposito.py
# Program ini mensimulasikan saldo deposito dengan
# bunga majemuk sebesar 8% dan menghitung berapa lama
# saldo deposito menjadi dua kali lipat

def main():
    # Constant bunga 8% setahun
    BUNGA = 0.08
    # Inisialisasi akumulator saldo
    saldo = 0.0
    # Variabel tahun berjalan
    tahun = 0
    # Minta saldo awal ke pengguna
    saldo_awal = float(input('Masukkan saldo awal: '))
    # Tetapkan saldo ke saldo_awal
    saldo = saldo_awal

    # Hitung saldo per tahun dan berhenti ketika
    # saldo akumulasi lebih besar dari 2 x saldo_awal
    while saldo <= saldo_awal * 2:
        # Saldo tahun ke n = saldo tahun ke (n-1) +
        #                               BUNGA * saldo tahun ke (n-1)
        saldo += BUNGA * saldo
        # Inkrementasi tahun berjalan
        tahun += 1
        # Tampilkan saldo per tahun
        print(f'Saldo tahun ke-{tahun}: {saldo:,.2f}')

    # Tampilkan lama tahun untuk mendapatkan
    # saldo akhir 2 x saldo awal
    print(f'\nwaktu yang dibutuhkan untuk melipatgandakan saldo awal: {tahun}
tahun')
    print(f'Saldo akhir: {saldo:,.2f} ')

# Panggil fungsi main
main()
```

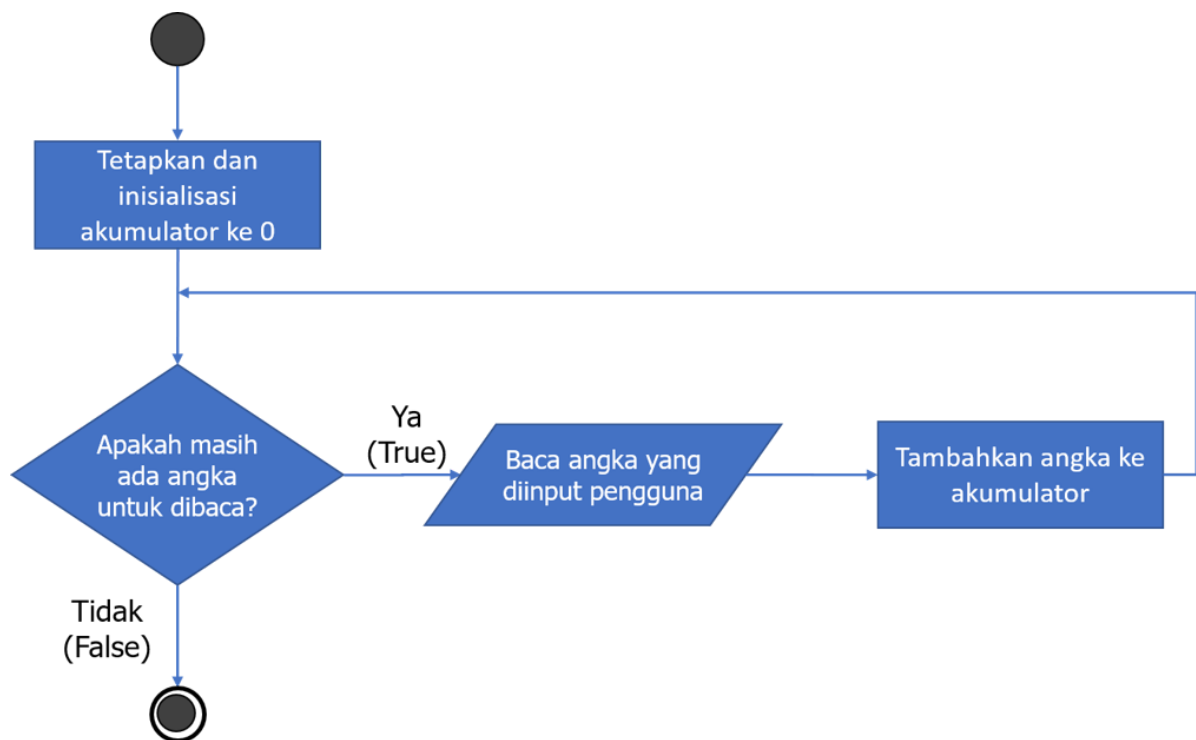
Contoh output dari program `deposito.py` di atas:

```
Masukkan saldo awal: 250000
Saldo tahun ke-1: 270,000.00
Saldo tahun ke-2: 291,600.00
Saldo tahun ke-3: 314,928.00
Saldo tahun ke-4: 340,122.24
Saldo tahun ke-5: 367,332.02
Saldo tahun ke-6: 396,718.58
Saldo tahun ke-7: 428,456.07
Saldo tahun ke-8: 462,732.55
Saldo tahun ke-9: 499,751.16
Saldo tahun ke-10: 539,731.25

waktu yang dibutuhkan untuk melipatgandakan saldo awal: 10 tahun
Saldo akhir: 539,731.25
```

Menghitung Total Berjalan Input Pengguna

Contoh lain penerapan loop yang menghitung total berjalan adalah loop yang menghitung jumlah dari angka-angka yang di-input pengguna. Logika dari loop ini digambarkan oleh flowchart berikut:



Program berikut menghitung total berjalan dari lima angka yang di-input pengguna:

```
# jumlah_input_angka.py
# Program ini menghitung total berjalan dari lima angka
# yang dimasukkan pengguna

# Constant dari banyak angka maksimum
```

```

MAKS = 5

def main():
    # Inisialisasi variabel akumulator
    total = 0.0

    # Tampilkan penjelasan program
    print(f'Program ini menghitung jumlah dari {MAKS} angka yang Anda masukkan.')

    # Ambil angka dari pengguna dan akumulasikan
    for counter in range(MAKS):
        num = int(input('Masukkan sebuah angka: '))
        total += num

    # Tampilkan total dari angka-angka yang dimasukkan
    print('Total:', total)

# Panggil fungsi main
main()

```

Contoh output dari program `jumlah_input_angka.py` di atas:

```

Program ini menghitung jumlah dari 5 angka yang Anda masukkan.
Masukkan sebuah angka: 49
Masukkan sebuah angka: 55
Masukkan sebuah angka: 89
Masukkan sebuah angka: 34
Masukkan sebuah angka: 24
Total: 251.0

```

3.6.2 Loop Sentinel

Pada contoh program `jumlah_input_angka.py`, kita meminta pengguna memasukkan angka sebanyak jumlah tertentu (sebanyak 5 nilai input). Bagaimana jika kita ingin membuat program yang memungkinkan pengguna memasukkan angka-angka sebanyak berapapun ? Kita dapat meminta pengguna mengetikkan suatu nilai khusus yang menandakan akhir dari penginputan. Nilai khusus yang menandakan akhir dari penginputan ini disebut dengan nilai **sentinel**.

Kode berikut mencontohkan penggunaan nilai sentinel untuk mengakhiri loop yang meminta input pengguna:

```

total = 0
num = int(input('Masukkan angka (0 untuk mengakhiri): '))
while num != 0:
    total += num
    num = int(input('Masukkan angka (0 untuk mengakhiri): '))

```

Loop meminta pengguna memasukkan angka sampai dengan pengguna memasukkan nilai 0 dan menjumlahkan angka-angka yang dimasukkan. Kita menggunakan nilai 0 sebagai nilai sentinel. Jika pengguna memasukkan nilai 0, maka program berhenti meminta input:

Kita menggunakan nilai 0 sebagai sentinel.
Jika pengguna memasukkan nilai 0, program berhenti meminta input.

```
total = 0
num = int(input('Masukkan angka (0 untuk mengakhiri): '))
while num != 0:
    total += num
    num = int(input('Masukkan angka (0 untuk mengakhiri): '))
```

Meminta input pertama kali di luar struktur loop.

Loop meminta pengguna memasukkan angka sampai dengan pengguna memasukkan nilai 0 dan menjumlahkan angka-angka yang dimasukkan.

Kode program lengkap dari contoh loop di atas:

```
# total_sentinel.py
# Program ini meminta pengguna menginput angka
# dan menghitung total jumlah angka yang di-input.
# Untuk menghentikan input, pengguna menginput 0.

def main():
    # Variabel akumulator
    total = 0
    # Ambil angka pertama
    num = int(input('Masukkan angka (0 untuk mengakhiri): '))
    # Jika angka yang dimasukkan 0, hentikan loop,
    # jika tidak lanjutkan meminta angka
    while num != 0:
        total += num
        num = int(input('Masukkan angka (0 untuk mengakhiri): '))

    print('Total angka yang dimasukkan:', total)

# Panggil fungsi main
main()
```

Output dari program di atas:

```
Masukkan angka (0 untuk mengakhiri): 34
Masukkan angka (0 untuk mengakhiri): 22
Masukkan angka (0 untuk mengakhiri): 56
Masukkan angka (0 untuk mengakhiri): 77
Masukkan angka (0 untuk mengakhiri): 0
Total angka yang dimasukkan: 189
```

Berikut adalah contoh lain penerapan loop sentinel. Program ini menghitung rata-rata dari nilai yang dimasukkan pengguna dan menggunakan karakter string kosong sebagai sentinel yang menghentikan input pengguna:

```
# average_sentinel.py
# Program ini meminta pengguna menginput angka
# dan menghitung total jumlah dan
# rata-rata angka-angka yang di-input.
# Untuk menghentikan input, pengguna menekan ENTER.

def main():
    # Variabel akumulator
    total = 0

    # Variabel untuk menyimpan banyaknya angka yang diinput
```



```

counter = 0

# Ambil angka pertama (tidak dikonversi ke int)
num = input('Masukkan angka (ENTER untuk mengakhiri): ')

# Jika karakter yang dimasukkan karakter kosong '',
# hentikan loop, jika bukan lanjutkan meminta angka
while num != '':
    # Inkrementasi penghitung input
    counter += 1
    # Akumulasikan total
    total += int(num)
    # Ambil angka selanjutnya (tidak dikonversi ke int)
    num = input('Masukkan angka (ENTER untuk mengakhiri): ')

# Tampilkan total jumlah dan rata-rata
print('Total angka yang dimasukkan:', total)
print('Rata-rata dari angka yang dimasukkan:', total/counter)

# Panggil fungsi main
main()

```

Contoh output dari program `average_sentinel.py` di atas:

```

Masukkan angka (ENTER untuk mengakhiri): 34
Masukkan angka (ENTER untuk mengakhiri): 59
Masukkan angka (ENTER untuk mengakhiri): 87
Masukkan angka (ENTER untuk mengakhiri):
Total angka yang dimasukkan: 180
Rata-rata dari angka yang dimasukkan: 60.0

```

Pada program di atas, kita menggunakan nilai string kosong (``) sebagai nilai sentinel. Ketika pengguna menekan pada prompt, fungsi `input` akan mengembalikan string kosong, yang menyebabkan loop berakhir.

3.6.3 Loop Tersarang

Kita dapat menuliskan loop di dalam loop. Loop di dalam loop ini disebut dengan loop tersarang (nested loop).

Program berikut menerapkan loop tersarang untuk menghitung nilai rata-rata dari ujian-ujian setiap mahasiswa dalam suatu kelas:

```

# rataan_nilai_ujian.py
# Prgram ini menghitung rata-rata nilai ujian
# setiap mahasiswa dalam satu kelas

def main():
    # Ambil banyak mahasiswa dan banyak ujian
    num_mhs = int(input('Banyak mahasiswa yang ingin diinput: '))
    num_ujian = int(input('Banyak ujian per mahasiswa: '))

    # Hitung rata-rata nilai ujian setiap mahasiswa
    for mhs in range(num_mhs):
        # Inisialisasi akumulator
        total = 0.0

```

```

# Ambil nilai-nilai ujian per mahasiswa
print(f'Mahasiswa ke-{mhs+1}')
print('-----')
for ujian in range(num_ujian):
    print(f'Ujian ke-{ujian+1}', end='')
    nilai_ujian = float(input(': '))
    # Akumulasikan nilai_ujian
    total += nilai_ujian

# Hitung rata-rata nilai ujian
rataan = total / num_ujian

# Tampilkan rata-rata
print(f'Rata-rata ujian untuk mahasiswa ke-{mhs+1}: {rataan}')
print()

# Panggil fungsi main
main()

```

Contoh output dari program `rataan_nilai_ujian.py` di atas:

```

Banyak mahasiswa yang ingin diinput: 3
Banyak ujian per mahasiswa: 3
Mahasiswa ke-1
-----
Ujian ke-1: 100
Ujian ke-2: 95
Ujian ke-3: 90
Rata-rata ujian untuk mahasiswa ke-1: 95.0

Mahasiswa ke-2
-----
Ujian ke-1: 80
Ujian ke-2: 81
Ujian ke-3: 82
Rata-rata ujian untuk mahasiswa ke-2: 81.0

Mahasiswa ke-3
-----
Ujian ke-1: 75
Ujian ke-2: 85
Ujian ke-3: 80
Rata-rata ujian untuk mahasiswa ke-3: 80.0

```

Contoh lain penerapan loop tersarang adalah loop yang mencetak pola segitiga:

```

*
**
***
****
*****
*****
*****
*****

```

Program berikut mencetak pola segitiga seperti di atas:

```
# pola_segitiga.py
# Program ini mencetak pola segitiga

def main():
    ALAS = 8
    for baris in range(ALAS):
        for kolom in range(baris + 1):
            print('*', end='')
        print()

main()
```

Perhatikan kode program di atas. Pada loop bagian luar, baris 6, pemanggilan fungsi `range(ALAS)` menghasilkan sebuah barisan bilangan: 0, 1, 2, 3, 4, 5, 6, 7. Sehingga, variabel `baris` ditugaskan nilai 0 s.d 7 saat loop bagian luar beriterasi. Pemanggilan fungsi `range` pada loop bagian dalam, pada baris 7, adalah `range(baris + 1)`. Loop bagian dalam dieksekusi sebagai berikut:

- Saat iterasi pertama loop bagian luar, variabel `baris` ditugaskan dengan nilai 0, sehingga `range(baris + 1)`, menyebabkan loop bagian dalam beriterasi 1 kali dan mencetak satu karakter bintang.
- Saat iterasi kedua loop bagian luar, variabel `baris` ditugaskan dengan nilai 1, sehingga `range(baris + 1)` menyebabkan loop bagian dalam beriterasi 2 kali dan mencetak dua karakter bintang.
- Saat iterasi ketiga loop bagian luar, variabel `baris` ditugaskan dengan nilai 2, sehingga `range(baris + 1)` menyebabkan loop bagian dalam beriterasi 3 kali dan mencetak tiga karakter bintang.
- Proses ini berlanjut sampai iterasi terakhir loop bagian luar, yaitu ketika variabel `baris` ditugaskan dengan nilai 7 dan menyebabkan loop bagian dalam beriterasi 7 kali dan mencetak tujuh karakter bintang.

REFERENSI

[1] Gaddis, Tony. 2012. Starting Out With Python Second Edition. United States of America: Addison-Wesley