Bab 5. Statement Control

Pada Bab 2, khususnya pada subbab logika bahasa pemrograman telah dijelaskan tentang beberapa jenis alur proses instruksi. Alur-alur proses instruksi tersebut merupakan mekanisme untuk mengatur aliran proses perintah yang dijalankan. Istilah lain untuk menyebut pengaturan alur proses instruksi adalah statement kontrol. Pada bab ini, khusus akan dijelaskan dua jenis statement kontrol secara lebih detil, yaitu statement kontrol percabangan atau kondisional (bersyarat) dan perulangan (*looping*).

Statement Kondisional/Percabangan

Statement kondisional digunakan untuk menyatakan pernyataan bersyarat. Di dalam Python hanya terdapat satu statemen yang menyatakan kondisional, yaitu IF dengan sintaks:

```
if syarat :
    ekspresi...
    ekspresi...
```

Pada sintaks tersebut, `syarat` harus bernilai boolean (`True` atau `False`). Jika `syarat` bernilai `True`, maka ekspresi akan dijalankan. Sedangkan jika bernilai `False` maka tidak melakukan apa-apa.

Catatan: Sebagai penanda blok ekspresi yang akan dilakukan ketika syarat bernilai `True` adalah adanya indentasi (tab).

Adapun contoh penggunaan statemen IF adalah:

```
In [1]: bil = 30
        if bil > 10 :
            print('lebih besar dari 10')

        lebih besar dari 10
```

Ketika kode program di atas dijalankan, maka program akan mencetak '`lebih besar dari 10`'.

Pada contoh yang diberikan, nilai variabel `bil` adalah 30. Sehingga dalam hal ini, syarat `bil > 10` bernilai `True` dan akibatnya akan mencetak `'lebih besar dari 10'`.

Perhatikan contoh selanjutnya berikut ini:

```
In [5]: bil = 20
        if bil > 30 :
            print('lebih besar dari 10')
```

Hasil dari contoh di atas tidak menampilkan apa-apa ketika dijalankan, karena nilai syarat nya adalah `False` sehingga tidak ada ekspresi yang akan dijalankan.

Statement IF dapat pula berbentuk sebagaimana sintaks berikut ini:

```
if syarat :
    ekspresi1
else :
    ekspresi2
```

Bentuk sintaks IF tersebut, `ekspresi2` akan dijalankan ketika syarat bernilai `False`. Sedangkan `ekspresi1` akan dijalankan ketika syarat bernilai `True`.

Dalam hal ini, penanda blok ekspresi yang akan dieksekusi ketika syarat bernilai `True` atau ketika syarat bernilai `False` adalah dengan adanya indentasi (tab).

Adapun contoh penggunaannya adalah:

```
In [6]: bil = 20
        if bil < 10:
              print('lebih kecil dari 10')
        else:
              print('tidak lebih kecil dari 10')

        tidak lebih kecil dari 10
```

Ketika kode program tersebut dijalankan, maka akan dihasilkan tampilan '`tidak lebih kecil dari 10`'. Hal ini disebabkan karena syarat bernilai `False`, sehingga yang dijalankan adalah ekspresi pada bagian `else`.

Selain dua bentuk statement IF yang telah diberikan, terdapat juga bentuk statement IF dengan sintaks:

```
if syarat1 :
    ekspresi1
elif syarat2 :
    ekspresi2
elif syarat3 :
    ekspresi3
.
.
else :
    ekspresiN
```

Berdasarkan sintaks di atas, apabila `syarat1` bernilai `True` maka `ekspresi1` yang akan dijalankan, sedangkan `ekspresi2`, `ekspresi3` dst tidak akan dijalankan. Namun, jika `syarat1` bernilai `False` maka akan dilakukan pengecekan untuk `syarat2`. Apabila `syarat2` bernilai `True` maka `ekspresi2` yang akan dijalankan. Sedangkan apabila `syarat2` bernilai `False` maka akan dilakukan pengecekan pada `syarat3` demikian seterusnya. Adapun apabila tidak ada syarat yang bernilai `True`, maka ekspresi yang akan dijalankan adalah `ekspresiN`.

Berikut ini contoh penggunaannya:

```
In [7]: bil = 10
        if bil < 10:
              print('lebih kecil dari 10')
        elif bil > 10:
              print('lebih besar dari 10')
        else:
              print('sama dengan 10')
```

```
sama dengan 10
```

Apabila program di atas dijalankan, maka tampilan yang muncul adalah '`sama dengan 10`'. Hal ini disebabkan karena kedua syarat sebelumnya, yaitu bil < 10 dan bil > 10, semuanya bernilai `False`.

Sekarang, perhatikan contoh berikut ini:

```
In [8]: bil = 10
        if bil < 10:
              print('lebih kecil dari 10')
        elif bil == 10:
              print('sama dengan 10')
        else:
              print('lebih kecil dari 10')
```

```
sama dengan 10
```

Dari program yang diberikan akan diperoleh hasil tampilan '`sama dengan 10`'. Tampilan ini dihasilkan pada saat syarat ke-2 (`bil == 10`) bernilai `True`. Dalam hal ini bagian else tidak dilakukan karena syarat ke-2 sudah terpenuhi.

Studi Kasus 5.1

Buatlah program Python untuk mengkonversi nilai angka 0-100 ke dalam nilai huruf, dengan ketentuan sebagai berikut

Nilai Angka	Nilai Huruf
80 – 100	A
70 – 79	B
60 – 69	C
40 – 59	D
0 – 39	E

Jawab:

Berdasarkan kasus yang diberikan, maka untuk kasus ini terdapat satu buah input yaitu nilai angka. Adapun outputnya adalah nilai huruf. Sehingga salah satu solusi programnya adalah sebagai berikut:

```
# program konversi nilai angka 0-100
# ke nilai huruf

print('Masukkan nilai angka  : ')
nAngka = int(input())

if nAngka >= 80 and nAngka <=100:
    nHuruf = 'A'
elif nAngka >= 70 and nAngka <= 79:
    nHuruf = 'B'
elif nAngka >= 60 and nAngka <= 69:
    nHuruf = 'C'
elif nAngka >= 40 and nAngka <= 59:
    nHuruf = 'D'
elif nAngka >= 0 and nAngka <= 39:
    nHuruf = 'E'
else:
    print('Nilai angka tidak valid')

print('Nilai hurufnya adalah:' + nHuruf)
```

Program tersebut apabila dijalankan dan kemudian diberikan input angka 0 s/d 100 maka bisa memberikan hasil yang benar. Misalkan apabila diberikan input angka 78 maka akan memberikan output nilai hurufnya B

```
Masukkan nilai angka  :
78
Nilai hurufnya adalah :  B
```

Namun, apabila dimasukkan nilai angka di luar 0 s/d 100, misalkan 200 maka akan muncul error sebagai berikut

```
Masukkan nilai angka  :
200
Nilai angka tidak valid

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-10958fe83d66> in <module>()
     18     print('Nilai angka tidak valid')
     19
---> 20 print('Nilai hurufnya adalah : ', nHuruf)

NameError: name 'nHuruf' is not defined
```

Error tersebut muncul dikarenakan kegagalan pada proses menjalankan perintah `print()`. Di dalam perintah `print()` terdapat `nHuruf` yang akan dicetak, sedangkan untuk kasus nilai angka 200 nilai `nHuruf` tidak ada. Oleh karena itu sebaiknya program tersebut dimodifikasi, misalnya menjadi berikut ini

```
# program konversi nilai angka 0-100
# ke nilai huruf

print('Masukkan nilai angka  : ')
nAngka = int(input())

if nAngka >= 80 and nAngka <=100:
    nHuruf = 'A'
elif nAngka >= 70 and nAngka <= 79:
    nHuruf = 'B'
elif nAngka >= 60 and nAngka <= 69:
    nHuruf = 'C'
elif nAngka >= 40 and nAngka <= 59:
    nHuruf = 'D'
elif nAngka >= 0 and nAngka <= 39:
    nHuruf = 'E'
else:
    print('Nilai angka tidak valid')

#modifikasi pada bagian cetak nilai huruf
if nAngka >= 0 and nAngka <= 100:
    print('Nilai hurufnya adalah : ', nHuruf)
```

```
Masukkan nilai angka  :
200
Nilai angka tidak valid
```

Modifikasi yang dilakukan pada program tersebut adalah dengan memberi kondisional (syarat) pada proses `print()` yaitu hanya akan dilakukan apabila nilai angkanya 0 s/d 100 saja.

Alternatif lainnya dapat pula menjadi berikut ini:

```
# program konversi nilai angka 0-100
# ke nilai huruf

print('Masukkan nilai angka  : ')
nAngka = int(input())

if nAngka >= 0 and nAngka <= 100:
    if nAngka >= 80 and nAngka <=100:
        nHuruf = 'A'
    elif nAngka >= 70 and nAngka <= 79:
        nHuruf = 'B'
    elif nAngka >= 60 and nAngka <= 69:
        nHuruf = 'C'
    elif nAngka >= 40 and nAngka <= 59:
        nHuruf = 'D'
    elif nAngka >= 0 and nAngka <= 39:
        nHuruf = 'E'

    print('Nilai hurufnya adalah : ', nHuruf)
else:
    print('Nilai angka tidak valid')
```

Pada modifikasi tersebut, rangkaian perintah IF untuk menentukan nilai huruf dimasukkan ke dalam syarat `if nAngka >= 0 and nAngka <= 100`. Hal ini berarti bahwa proses konversi penentuan

nilai angka hanya akan dilakukan ketika nilai angka dari input hanyalah 0 s/d 100 saja. Sedangkan apabila tidak, maka nilai angkanya tidak valid.

■

Statement Perulangan (Looping)

Statement perulangan digunakan untuk mengulang ekspresi atau perintah hingga mencapai batas atau kondisi tertentu. Dalam hal ini, batas perulangan bisa berupa banyaknya perulangan, artinya bahwa perulangan akan berhenti apabila banyaknya perulangan yang sudah dilakukan telah mencapai banyaknya perulangan yang diinginkan. Selain itu batas perulangan juga bisa berupa syarat kapan perulangan harus berhenti. Analogi kedua jenis batas perulangan tersebut adalah pada tiga kalimat sehari-hari sebagai berikut:

1. Ali makan bakso sampai dengan 10 kali
2. Ali terus makan bakso selama dia masih belum kenyang
3. Ali terus makan bakso sampai kenyang

Kalimat nomor 1, perulangan yang dilakukan oleh Ali adalah makan bakso. Proses makan bakso yang dilakukan oleh Ali baru akan berhenti apabila dia telah makan 10 kali. Sehingga dalam contoh ini perulangan akan berhenti ketika telah mencapai jumlah perulangan tertentu yang diinginkan. Sedangkan pada kalimat nomor 2, proses makan yang dilakukan oleh Ali tidak diketahui banyak perulangannya. Dalam hal ini, yang diketahui kapan dia berhenti makan bakso yaitu ketika dia merasa kenyang. Artinya selama Ali masih belum merasa kenyang maka dia akan terus makan bakso.

Di dalam Python terdapat dua struktur kontrol yang digunakan untuk melakukan perulangan, yaitu WHILE dan FOR.

Statement WHILE dapat digunakan untuk kedua jenis keadaan perulangan di atas. Dengan kata lain, WHILE dapat digunakan untuk perulangan yang diketahui banyaknya perulangan dan tidak diketahui banyaknya perulangan (hanya diketahui syarat selama apa perulangan tersebut dilakukan).

Sedangkan untuk FOR hanya dapat digunakan untuk menyatakan perulangan yang sudah diketahui banyaknya perulangan saja.

Struktur While

Struktur sintaks dari perulangan WHILE adalah sebagai berikut:

```
while syarat:
   ekspresi...
   ekspresi...
```

Berdasarkan sintaks tersebut, selama syarat bernilai `True` maka ekspresi dalam blok while akan terus dilakukan. Dengan kata lain, perulangan while baru akan berhenti ketika syarat bernilai `False`.

Berikut ini adalah contoh penggunaannya:

```
n = 1
while n <= 3:
    print('Ali makan bakso')
    n = n + 1
```

```
Ali makan bakso
Ali makan bakso
Ali makan bakso
```

Kode program tersebut mencetak 'Ali makan bakso' sebanyak 3 kali. Adapun penjelasan dari contoh kode tersebut adalah sebagai berikut:

Nilai awal untuk variabel `n` adalah 1. Variabel `n` di sini dimaksudkan sebagai penghitung banyaknya perulangan yang akan dilakukan (biasanya disebut variabel *counter*). Adapun syarat perulangan adalah `n ≤ 3`, yang artinya bahwa selama nilai `n` nya memenuhi `n ≤ 3` maka perulangan akan tetap terus dilakukan. Ekspresi `n = n + 1` digunakan untuk menambah 1 nilai `n` sebelumnya. Sehingga proses yang terjadi ketika kode program dijalankan adalah

```
n = 1
apakah n ≤ 3? → True
      cetak 'Ali makan bakso'
      n = n + 1 → n = 1 + 1 = 2
apakah n ≤ 3? → True
      cetak 'Ali makan bakso'
      n = n + 1 → n = 2 + 1 = 3
apakah n ≤ 3? → True
      cetak 'Ali makan bakso'
      n = n + 1 → n = 3 + 1 = 4
apakah n ≤ 3? → False
perulangan WHILE berhenti
```

Contoh WHILE yang diberikan sebelumnya merupakan contoh perulangan yang diketahui banyaknya perulangan.

Selanjutnya perhatikan contoh kode program dan outputnya berikut ini:

```
bil = 0
while bil != -1:
    print('Masukkan sebuah bilangan :')
    bil = int(input())
```

```
Masukkan sebuah bilangan :
10
Masukkan sebuah bilangan :
21
Masukkan sebuah bilangan :
9
Masukkan sebuah bilangan :
-1
```

Kode program tersebut ketika dijalankan akan terus meminta input bilangan selama bilangan yang diinputkan bukan -1. Dalam hal ini banyaknya perulangan tidak diketahui, namun hanya kapan syarat perulangan itu berhenti yang diketahui yaitu selama inputnya bukan -1.

Penggunaan ekspresi `bil = 0` sebelum WHILE adalah untuk nilai awal supaya perulangan WHILE nya bisa berjalan. Nilai awal yang dipilih bisa sembarang nilai asalkan bukan -1.

Statement '*break*'

Sebuah perulangan dapat dihentikan meskipun syarat perulangan masih memenuhi (bernilai `True`) dengan menggunakan statement `break`. Berikut ini adalah contoh penggunaannya dalam bentuk kode program dan outputnya:

```
while True:
    print('Masukkan bilangan : ')
    bil = int(input())
    if bil == -1:
        break
print('Terimakasih')
```

```
Masukkan bilangan :
10
Masukkan bilangan :
4
Masukkan bilangan :
-1
Terimakasih
```

Pada contoh di atas, perulangan WHILE selalu akan terus dilakukan karena syaratnya adalah `True`. Namun jika input bilangannya adalah -1 maka perulangannya akan berhenti (akibat dari penggunaan `break`). Selanjutnya setelah perulangan WHILE berhenti akan mencetak '`Terimakasih`'.

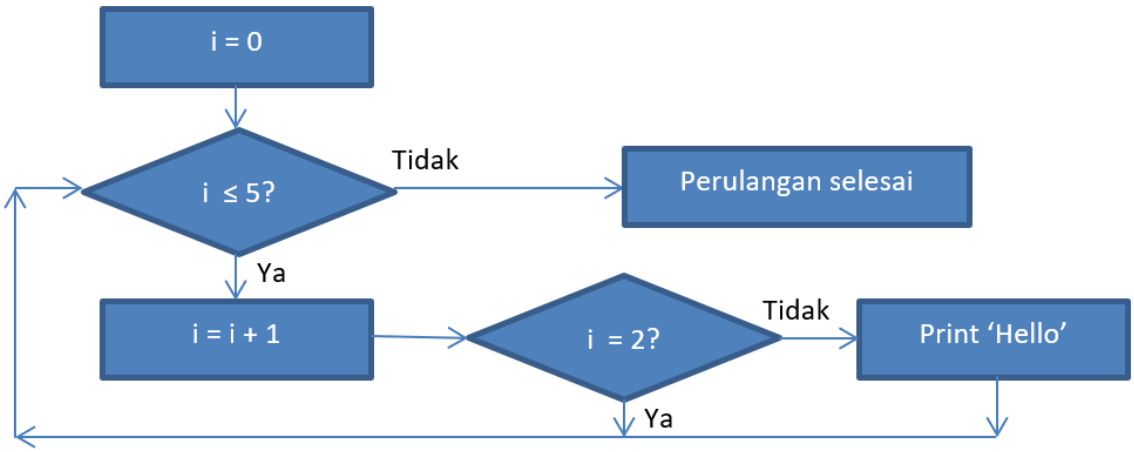
Statement '*continue*'

Di dalam perulangan, bisa ditambahkan statement `continue`. Statement ini digunakan untuk mengulangi kembali ke bagian awal looping tanpa menyelesaikan satu perulangan utuh. Perhatikan contoh berikut ini beserta outputnya:

```
i = 0;
while i<=5:
    i = i+1
    if i==2:
        continue
    print('i=' + str(i) + ', Hello')
```

```
i=1, Hello
i=3, Hello
i=4, Hello
i=5, Hello
i=6, Hello
```

Nilai `i` mula-mula (sebelum masuk ke WHILE adalah 0). Selanjutnya di dalam WHILE, perulangan akan dilakukan selama nilai `i` memenuhi $i \leq 5$. Di dalam WHILE, nilai `i` bertambah 1 dari sebelumnya, kemudian mencetak 'hello'. Namun ketika i=2, ada statement `continue` sehingga pengaruhnya adalah ketika `i=2` tidak mencetak 'hallo' karena ketika `continue` aliran proses program kembali ke awal WHILE. Secara diagram proses ini digambarkan sebagai berikut:



Studi Kasus 5.2

Buatlah sebuah program untuk menjumlahkan dua buah bilangan bulat yang berasal dari input. Setiap kali setelah menampilkan hasil penjumlahannya, munculkan pertanyaan (Y/N) apakah user ingin mengulanginya lagi (menjumlahkan untuk bilangan lainnya). Jika user memilih Y, maka program meminta user memasukkan kembali kedua bilangan yang akan dijumlahkan kemudian program menampilkan hasilnya. Namun jika user memilih N, maka program berhenti.

Jawab:

Pada studi kasus yang diberikan, terdapat perulangan yang banyaknya perulangan tidak ditentukan. Dalam hal ini, perulangan akan terus dilakukan selama user memilih Y untuk mengulangi penjumlahan kedua bilangan. Sehingga kode program yang menjadi solusi dari studi kasus ini adalah

```
jawab = 'y'
while jawab == 'y':
    print('Masukkan bil pertama : ')
    bil1 = int(input())
    print('Masukkan bil kedua   : ')
    bil2 = int(input())
    hasil = bil1 + bil2
    print('Hasil penjumlahannya : ', hasil)
    print('Ulangi lagi (y/n)?   : ')
    jawab = input()
```

Dalam kode program tersebut, yang merupakan syarat bagi perulangan WHILE adalah selama nilai `jawab` adalah `'y'` atau `jawab == 'y'`. Oleh karena itu, perlu diberikan nilai awal variabel `jawab` sebelum WHILE yaitu `'y'` (`jawab = 'y'`) supaya perulangan WHILE dapat berjalan ketika pertama kali. Selanjutnya nilai `jawab` tergantung input yang dilakukan user.

Sehingga output dari program di atas adalah:

```
Masukkan bil pertama :
4
Masukkan bil kedua   :
5
Hasil penjumlahannya :  9
Ulangi lagi (y/n)?   :
y
Masukkan bil pertama :
3
Masukkan bil kedua   :
6
Hasil penjumlahannya :  9
Ulangi lagi (y/n)?   :
n
```

Alternatif solusi dari permasalahan yang diberikan tersebut, dapat pula ditulis kode program sebagai berikut:

```
while True:
    print('Masukkan bil pertama : ')
    bil1 = int(input())
    print('Masukkan bil kedua   : ')
    bil2 = int(input())
    hasil = bil1 + bil2
    print('Hasil penjumlahannya : ', hasil)
    print('Ulangi lagi (y/n)?   : ')
    jawab = input()
    if jawab == 'n':
        break
```

Pada kode program tersebut, menggunakan break untuk mengakhiri perulangan WHILE nya, yaitu ketika `jawab == 'n'`.

■

Studi Kasus 5.3

Buatlah kode program Python untuk mengenerate syair lagu 'anak ayam' sebagai berikut

Anak ayam turun 3, mati satu tinggalah 2
Anak ayam turun 2, mati satu tinggalah 1
Anak ayam turun 1, mati satu tinggal induknya

Adapun jumlah anak ayam mula-mula adalah n buah yang merupakan input program. Contoh syair anak ayam di atas adalah untuk n = 3.

Jawab:

Studi kasus yang diberikan di atas terdapat perulangan untuk mencetak '*Anak ayam turun ..., mati satu tinggallah ...*'. Perulangan tersebut diulang sebanyak n kali, namun ada pengecualian bentuk teks yang dicetak yaitu ketika jumlah anaknya tinggal satu. Selain itu, jumlah anak ayam (n) pada setiap kali perulangan nilainya berkurang satu. Sehingga kode program yang bisa menjadi solusi dari studi kasus ini adalah:

```
print('Masukkan jumlah anak ayam mula-mula: ')
n = int(input())
while n >= 1:
    if n > 1:
        print('Anak ayam turun ', n, ', mati satu tinggallah ', n-1)
    else:
        print('Anak ayam turun ', n, ', mati satu tinggal induknya')
    n = n-1
```

Dalam kode program di atas, perulangan WHILE diberikan syarat yaitu selama nilai `n` memenuhi `n ≥ 1` yang artinya bahwa perulangan akan terus dilakukan selama jumlah anak ayam (`n`) masih `≥ 1`. Selanjutnya di dalam WHILE terdapat sebuah kondisi di mana akan mencetak '`Anak ayam turun n, mati satu tinggallah n-1`' atau '`Anak ayam turun n, mati satu tinggal induknya`'.

Sesuai dengan kasusnya, perintah untuk mencetak '`Anak ayam turun n, mati satu tinggallah n-1`' dilakukan ketika anak ayamnya lebih dari 1. Namun ketika `n = 1` maka akan mencetak '`Anak ayam turun n, mati satu tinggal induknya`'. Adapun ekspresi `n=n-1` digunakan untuk mengurangi nilai `n` sebelumnya. Sehingga output dari kode program di atas adalah

```
Masukkan jumlah anak ayam mula-mula:
5
Anak ayam turun  5 , mati satu tinggalah  4
Anak ayam turun  4 , mati satu tinggalah  3
Anak ayam turun  3 , mati satu tinggalah  2
Anak ayam turun  2 , mati satu tinggalah  1
Anak ayam turun  1 , mati satu tinggal induknya
```

■

Infinite Loop

Di dalam sebuah perulangan, bisa terjadi perulangan terus menerus dilakukan tanpa berhenti (*infinite loop*) yang disebabkan kesalahan logika. Sebagai contoh perhatikan kode program berikut ini

```
n = 1
while n < 5:
    print('Hello world')
```

Apabila kode di atas dijalankan, maka perulangan untuk mencetak 'Hello world' tidak akan berhenti dikarenakan nilai `n` (yaitu 1) selalu memenuhi `n < 5`. Dalam hal ini, nilai `n` selalu tetap tidak pernah berubah. Sehingga seterusnya nilai `n` yang sama dengan 1 pasti memenuhi `n < 5`.

Jika terjadi *infinite loop* di dalam Python, maka cara untuk menghentikan perulangan adalah dengan menekan tombol `CTRL + C` pada bagian console jika menggunakan Spyder, atau bisa dengan cara lain yaitu merestart kernel dengan mengklik menu `Kernel > Restart/Interrupt` (apabila menggunakan Jupyter Notebook). Selanjutnya kode program harus dimodifikasi supaya tidak terjadi *infinite loop* kembali.

Statement For

Statement perulangan berikutnya yang akan dibahas adalah FOR. Statement FOR hanya bisa digunakan untuk menyatakan perulangan yang sudah ditentukan banyaknya perulangan.

Sintaks dari perulangan FOR adalah sebagai berikut

```
for i in range(n):
    ...
    ...
```

Berdasarkan sintaks di atas, perulangan FOR dilakukan sebanyak `n` kali. Dalam hal ini, `i` merupakan sebuah variabel yang berfungsi sebagai pencacah (*counter*). Untuk setiap perulangan, nilai `i` ini akan berubah nilainya mulai dari 0, 1, 2, ..., `n-1` (total `n` perulangan). Setelah `i` nilainya sama dengan `n-1` maka perulangan berhenti.

Perhatikan contoh berikut ini:

```
for i in range(4):
    print('i=' + str(i) + ', Hello world')
```

Contoh di atas akan menghasilkan output:

```
i=0, Hello world
i=1, Hello world
i=2, Hello world
i=3, Hello world
```

Berdasarkan contoh tersebut tampak bahwa nilai `i` berubah pada setiap perulangannya mulai dari 0 sampai dengan 3 atau dalam hal ini sebanyak 4 kali perulangan. Selain mencetak nilai `i`, di dalam perulangan juga mencetak '`Hello world`'.

Penggunaan `range(n)` pada perulangan FOR dapat juga dalam bentuk yang lain. Selain itu, juga bisa menggunakan `range(a, n)`. Misalnya pada program berikut ini

```
for i in range(1, 5):
    print(i)
```

Output dari program tersebut adalah sebagai berikut:

```
1
2
3
4
```

Sehingga dalam hal ini, penggunaan `range(a, b)` akan memberi nilai variabel counternya mulai dari a, a+1, a+2, ... dan berhenti hingga sebelum mencapai b.

Selain itu, `range()` dapat pula berbentuk `range(a, b, step)`. Misalnya adalah:

```
for i in range(1, 15, 2):
    print(i)
```

Program tersebut jika dijalankan akan dihasilkan output:

```
1
3
5
7
9
11
13
```

Berdasarkan output tersebut, dapat disimpulkan bahwa nilai `step` di dalam `range(a, b, step)` memiliki makna besarnya kenaikan nilai variabel counternya. Mula-mula nilai variabel counternya adalah `a`, kemudian setiap kali perulangan nilainya bertambah sejumlah `step`, dan perulangan baru akan berhenti hingga nilainya sebelum mencapai `b`.

Studi Kasus 5.4

Buatlah program untuk menampilkan bilangan bulat 1 s/d 100 yang merupakan kelipatan 3.

Jawab:

Untuk menyelesaikan kasus di atas, dapat digunakan perulangan FOR dengan membuat nilai i = 0, 1, 2, ..., 99. Selanjutnya karena pada kasus yang diberikan bilangan bulatnya mulai dari 1, maka untuk setiap i tersebut perlu ditambah 1 supaya diperoleh bilangan 1, 2, 3, ..., 100. Kemudian setiap dari bilangan yang diperoleh ini dicek apakah merupakan kelipatan 3 atau tidak. Jika ya, maka ditampilkan. Sedangkan jika tidak, maka diabaikan (tidak dicetak). Sehingga berdasarkan ide penyelesaian ini diperoleh program sebagai berikut:

```
for i in range(100):
    bil = i+1
    if bil % 3 == 0:
        print(bil)
```

Dalam hal ini, untuk mengecek apakah `i+1` nya merupakan kelipatan 3 atau tidak digunakan modulo (%) dengan 3.

Alternatif program lainnya, bisa juga dituliskan kode programnya sebagai berikut

```
for i in range(100):
    if (i+1) % 3 == 0:
        print(i+1)
```

di mana perbedaan program tersebut dengan sebelumnya yaitu tidak membuat variabel `bil` untuk menyimpan `i+1` nya, namun langsung digunakan di dalam IF nya.

■

Studi Kasus 5.5

Buatlah program untuk menghitung banyaknya bilangan bulat 1 s/d 100 yang merupakan kelipatan 3.

Jawab:

Kasus ini merupakan pengembangan dari Kasus 8.4 sebelumnya. Ide untuk menghitung banyaknya bilangan bulat yang merupakan kelipatan 3 ini adalah dengan membuat proses perhitungan yang nilainya bertambah 1 (*increment* dengan faktor 1) setiap kali ditemukan bilangan yang merupakan kelipatan 3. Sebelum mulai perhitungan, terlebih dahulu diset nilai awal nya 0.

```
sum = 0
for i in range(100):
    if (i+1) % 3 == 0:
        sum = sum + 1
print('Banyaknya bilangan : ' + str(sum))
```

Dalam program tersebut, variabel `sum` digunakan untuk menyimpan hasil perhitungan banyaknya bilangan kelipatan 3. Nilai mula-mula variabel ini adalah 0 (sebelum mulai perhitungan). Output dari program adalah 33.

■

Studi Kasus 5.6

Buatlah program untuk menghitung hasil dari 1 + 2 + 3 + ... + 100.

Jawab:

Kasus di atas dapat diselesaikan dengan menggunakan looping. Ide penyelesaiannya adalah sebagai berikut:

- Mula-mula sebelum mulai menjumlahkan, nilai awal variabel `sum = 0` diberikan. Variabel `sum` ini digunakan untuk menyimpan nilai hasil penjumlahannya.
- Selanjutnya dimulai proses penjumlahannya. Satu persatu untuk setiap suku yang dijumlahkan dengan cara nilai `sum` sebelumnya dijumlahkan dengan suku pertama dan hasilnya disimpan kembali ke dalam `sum`

  `sum = sum + 1 = 0 + 1 = 1`

- Selanjutnya dilakukan lagi penjumlahan nilai `sum` dengan suku kedua, dan hasilnya disimpan ke dalam `sum` lagi.

  `sum = sum + 2 = 1 + 2 = 3`

- Proses penjumlahan ini dilakukan terus hingga semua sukunya terjumlahkan.

Berdasarkan ide penyelesaian di atas, maka hal ini bisa dinyatakan ke dalam bentuk looping FOR karena banyaknya perulangan di atas sudah jelas. Sehingga kode program penyelesaiannya adalah

```
sum = 0
for i in range(100):
    suku = i+1
    sum = sum + suku
print('Hasil penjumlahannya : ' + str(sum))
```

Dalam hal ini nilai `i` berubah dimulai dari 0, 1, 2, ..., 99. Sedangkan karena suku yang dijumlahkan 1, 2, 3, ..., 100 maka nilai `i` perlu ditambahkan 1 terlebih dahulu (`suku = i + 1`) baru kemudian dijumlahkan.

Selain menggunakan kode program di atas, bisa juga menyelesaikan kasusnya dengan looping sebanyak 101 kali sebagai berikut:

```
sum = 0
for i in range(101):
    sum = sum + i
print('Hasil penjumlahannya : ' + str(sum))
```

Jika menggunakan kode tersebut, maka looping terjadi untuk i = 0 sampai dengan 100, dengan kata lain akan menghitung penjumlahan 0 + 1 + 2 + ... + 100. Sama saja bukan??

Adapun output dari kedua program di atas adalah

```
Hasil penjumlahannya : 5050
```

■

Studi Kasus 8.7

Buatlah program untuk menampilkan semua bilangan yang habis dibagi 3 dan 5 antara 1 sampai 100, kemudian hitung banyaknya bilangan, serta hitung pula jumlahan semua bilangan tersebut.

Jawab:

Ide penyelesaian dari kasus yang diberikan bisa menggunakan perulangan FOR, dimulai dari `i` = 0 sampai dengan 99 (100 bilangan). Sehingga bilangan 1 s/d 100 dapat diperoleh dengan `bil = i + 1`. Di setiap nilai `bil` tersebut, nanti dilakukan pengecekan apakah `bil` tersebut habis dibagi 3 dan 5 atau tidak. Jika ya, maka masuk hitungan dan dijumlahkan, serta ditampilkan. Sedangkan jika tidak, maka diabaikan. Untuk menghitung banyaknya bilangan dengan menggunakan cara menambahkan 1 dari hasil hitungan sebelumnya (*increment*) setiap kali ditemukan bilangan yang habis dibagi 3 dan 5.

Sehingga kode program untuk solusi kasus tersebut adalah sebagai berikut:

```
hitung=0
sum=0
for i in range(100):
    bil = i+1
    if bil%3==0 and bil%5==0:
        hitung = hitung + 1
        sum = sum + bil
        print(bil)

print('Banyak bilangan   : ' + str(hitung))
print('Jumlahan bilangan : ' + str(sum))
```

Adapun output dari kode program di atas adalah

```
15
30
45
60
75
90
Banyak bilangan   : 6
Jumlahan bilangan : 315
```

■

Ekuivalensi Perulangan WHILE dan FOR

Seperti yang pernah dibahas sebelumnya bahwa statement WHILE dapat digunakan untuk menyatakan perulangan yang diketahui banyaknya perulangan. Demikian juga halnya dengan statement FOR. Oleh karena itu apabila dijumpai kasus di mana terdapat perulangan yang demikian, maka bisa menggunakan salah satu dari keduanya.

Perhatikan kedua kode program berikut ini untuk mencetak 'Hello World' sebanyak 5 kali, di mana program pertama menggunakan FOR, sedangkan yang ke dua menggunakan WHILE.

```
for i in range(5):
    print('i=' + str(i) + ', Hello world')
```

```
i = 0
while i<5:
    print('i=' + str(i) + ', Hello world')
    i = i + 1
```

Jika kedua program di atas dijalankan, maka akan menghasilkan output yang sama yaitu

```
i=0, Hello world
i=1, Hello world
i=2, Hello world
i=3, Hello world
i=4, Hello world
```

Meskipun keduanya menghasilkan output yang sama, namun jika dilihat dari efisiensi penulisan kode program tampak bahwa penulisan FOR lebih simpel dibandingkan WHILE. Apabila menggunakan WHILE untuk menyatakan perulangan yang diketahui banyaknya perulangan butuh deklarasi nilai awal (dalam hal ini `i = 0`), dan ekspresi increment (`i = i + 1`). Sedangkan di dalam FOR kedua hal ini tidak perlu dituliskan secara eksplisit karena keduanya sudah terkandung secara implisit di dalam sintaks FOR itu sendiri. Oleh karena itu, statement FOR lebih disarankan digunakan untuk menyatakan perulangan yang diketahui banyaknya perulangan.