3. Struktur Perulangan

Perulangan dalam dunia pemrograman adalah baris kode atau instruksi yang dieksekusi oleh komputer secara berulang-ulang sampai suatu kondisi tertentu terpenuhi. Konsep perulangan ini didukung di semua bahasa pemrograman modern, termasuk di antaranya adalah python.

Dengan perulangan, suatu kode program bisa dieksekusi berkali-kali dengan jumlah tertentu, atau selama sebuah kondisi tertentu terpenuhi.

Pada python, perulangan bisa dilakukan dengan beberapa cara, di antaranya:

1. Perulangan for
2. Perulangan while

Perulangan For Pada Python

Perulangan `for` pada python adalah perintah yang digunakan untuk **melakukan iterasi dari sebuah nilai sequence** atau data koleksi pada python seperti List, Tuple, String dan lain-lain.

For pada python memiliki **perilaku yang berbeda** dengan for pada kebanyakan bahasa pemrograman yang lain, karena pada python ia sangat berkaitan dengan data *sequence* atau data kolektif. Mungkin kalau dibandingkan dengan bahasa lain, for pada python lebih dikenal sebagai foreach.

Syntax For

Berikut ini adalah struktur sintaks metode for:

```
for nilai in sequence:
  # blok kode for
```

Jadi, ada 3 bagian penting.

1. `sequence`: adalah sebuah nilai yang bersifat *iterable* alias bisa diulang-ulang.

   Di antara tipe data yang bersifat *sequence* atau *iterable* adalah:

   - list
   - tuple
   - string
   - dan lain sebagainya
2. `nilai`: adalah setiap item yang diekstrak dari `sequence`
3. **Blok kode**: yaitu statemen-statemen atau perintah-perintah tertentu yang akan dieksekusi secara berulang.

For dengan list

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

for kota in listKota:
  print(kota)
```

Jika dieksekusi, program di atas akan menghasilkan output:

```
Jakarta
Surabaya
Depok
Bekasi
Solo
Jogjakarta
Semarang
Makassar
```

Mengetahui urutan iterasi for dengan list

Untuk mengetahui urutan iterasi for dengan list, bisa menggunakan fungsi `enumerate`.

Fungsi tersebut akan mengekstrak 2 buah nilai:

1. yang pertama adalah `index`: yaitu urutan iterasi yang ke berapa
2. dan `item` yang mana itu adalah nilai dari list itu sendiri.

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

for i, kota in enumerate(listKota):
  print(i, kota)
```

Kode program di atas sama saja seperti sebelumnya, hanya menambahkan fungsi `enumerate()` dan mem-*passing* variabel `listKota` sebagai parameter. Juga mengekstrak dua buah nilai yang diberi nama `i` dan `kota`.

Jila dijalankan, berikut adalah output yang didapat:

```
0 Jakarta
1 Surabaya
2 Depok
```

```
3 Bekasi
4 Solo
5 Jogjakarta
6 Semarang
7 Makassar
```

Perhatikan, urutannya dimulai dari **0**, bukan dari angka **1**.

For dengan fungsi range()

Selain dengan list, juga bisa menggunakan for dengan fungsi range().

Perhatikan contoh berikut:

```
## 0 sampai 4
for i in range(5):
  print("Perulangan ke -", i)
```

Output:

```
Perulangan ke - 0
Perulangan ke - 1
Perulangan ke - 2
Perulangan ke - 3
Perulangan ke - 4
```

Dengan fungsi range, perulangan bisa dilakukan dari 0, sampai **kurang dari** nilai range yang didefinisikan (yaitu 5 dalam contoh di atas). Sehingga hasil perulangan yang didapatkan adalah 0 sampai 4.

Bisa memulai range dari selain 0

Perhatikan contoh berikut:

```
## 10 sampai 15
for i in range(10, 16):
  print('i =', i)
```

Perulangan di atas akan menghasilkan output:

```
i = 10
i = 11
i = 12
i = 13
i = 14
i = 15
```

Juga bisa mendefinisikan kelipatannya:

Perhatikan contoh berikut:

```
## Bilangan genap kelipatan 2
```

```
for i in range(2, 12, 2):
  print('i =', i)
```

Pada contoh di atas, sistem akan melakukan perulangan dimulai dari angka 2, hingga kurang dari 12 dengan interval/kelipatan sebanyak 2.

Hasilnya:

```
i = 2
i = 4
i = 6
i = 8
i = 10
```

Untuk bilangan ganjil, mulai saja dari angka 1:

```
## Bilangan ganjil kelipatan 2
for bilangan_ganjil in range(1, 12, 2):
  print(bilangan_ganjil)
```

Output:

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

Nama variabel-nya bebas tidak harus `i`.

For dengan tuple

Tuple adalah di antara tipe data yang bersifat *iterable*, sehingga juga bisa memperlakukannya sebagai objek perulangan mengguankan `for`.

Perhatikan contoh di bawah:

```
tupleBuah = ('Mangga', 'Jeruk', 'Apel', 'Pepaya')

for buah in tupleBuah:
  print(buah)
```

Output:

```
Mangga
Jeruk
Apel
Pepaya
```

For dengan string

String pun demikian, bersifat **iterable**, sehingga bisa dijadikan objek perulangan.

Perhatikan contoh berikut:

```
for karakter in "Indonesia":
  print(karakter)
```

Jika dijalankan, output-nya:

```
I
n
d
o
n
e
s
i
a
```

Break dan continue

Pada python, bisa **menginterupsi** dan juga **men-skip** suatu iterasi pada perulangan.

Terdapat 2 perintah yang bisa digunakan, yaitu:

- `break` untuk interupsi (memberhentikan paksa) sebuah perulangan
- `continue` untuk menskip ke iterasi selanjutnya

Perhatikan contoh berikut:

```
for i in range(10, 20):
  # skip jika i == 15
  if (i == 15):
    continue

  print(i)
```

Output:

```
10
11
12
13
14
16  <-- Habis 14 langsung 16
17
18
19
```

Perhatikan output di atas, pada saat `i == 15`, perintah `print(i)`tidak dieksekusi dan justru di-skip ke iterasi berikutnya.

Atau...

Justru bisa memberhentikan paksa suatu perulangan sekalipun belum sampai ke iterasi yang terakhir.

```
for i in range(10, 20):
  # hentikan jika i == 15
  if (i == 15):
    break

  print(i)
```

Jika dijalankan:

```
10
11
12
13
14 <-- print terakhir sebelum terjadi break pada i == 15
```

Sistem akan memberhentikan perulangan ketika `i == 15` dan belum sempat melakukan perintah `print()`.

For ... else?

Hampir mirip dengan `if ... else`.

Tapi tugasnya berbeda.

Perulangan `for` jika ditambahkan blok `else`, maka perintah yang ada pada blok `else` hanya akan dieksekusi ketika perulangan selesai **secara natural** *–tanpa interupsi*.

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

for kota in listKota:
  print(kota)
else:
  print('Tidak ada lagi item yang tersisa')
```

Jika dijalankan, program di atas akan menghasilkan output seperti ini:

```
Jakarta
Surabaya
Depok
Bekasi
Solo
```

```
Jogjakarta
Semarang
Makassar
Tidak ada lagi item yang tersisa
```

For ... Else + Break

Jika digabungkan `for ... else` dengan `break`, maka blok `else` **hanya akan** dieksekusi jika perintah `break` tidak dieksekusi.

Bisa memanfaatkan `for ... else + break` untuk pencarian sebuah item pada list.

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

kotaYangDicari = input('Ketik nama kota yang kamu cari: ')

for i, kota in enumerate(listKota):
  # ubah katanya ke lowercase agar
  # menjadi case insensitive
  if kota.lower() == kotaYangDicari.lower():
    print('Kota yang anda cari berada pada indeks', i)
    break
else:
  print('Maaf, kota yang anda cari tidak ada')
```

Program di atas akan meminta user untuk menginputkan nama kota yang ingin dicari. Jika kotanya maka akan diberikan info indeks-nya berapa (dalam `listKota`), dan jika tidak ada maka perintah `print()` yang ada di blok `else` akan dieksekusi.

Coba jalankan. Kemudian input kata `solo`, ini hasilnya:

```
Ketik nama kota yang kamu cari: solo
Kota yang anda cari berada pada indeks 4
```

Jika cari kota yang tidak ada di dalam list, begini hasilnya:

```
Ketik nama kota yang kamu cari: pontianak
Maaf, kota yang anda cari tidak ada
```

Nah, harusnya sekarang sudah lebih jelas bagaimana cara `for ... else` bekerja, dan kapan blok kode `else` akan dieksekusi. Dia hanya akan dieksekusi ketika perulangan mencapai titik akhirnya (alias sudah tidak ada iterasi lagi yang tersisa).

Ada pun jika sebuah perulangan `for` dihentikan paksa dengan perintah `break`, maka perintah yang ada pada blok `else` tidak akan dieksekusi.

Perulangan While Pada Python

Perulangan while pada python adalah proses pengulangan suatu blok kode program **selama sebuah kondisi terpenuhi**.

Singkatnya, perulangan while adalah **perulangan yang bersifat *indefinite* alias tidak pasti**, atau bahkan tidak terbatas .

Sebuah blok kode akan dilakukan terus-menerus selama suatu kondisi terpenuhi. Jika suatu kondisi **ternyata tidak terpenuhi** pada iterasi ke 10, maka **perulangan akan berhenti**. Jika kondisi yang sama pada saat yang berbeda ternyata berhenti pada iterasi ke 100, maka perulangan akan berhenti pada jumlah tersebut.

Penulisan Sintaks While

Sintaks while bisa ditulis dengan cara berikut:

```
while <kondisi>:
  # blok kode yang akan diulang-ulang
```

Terdapat 3 komponen utama:

1. Yang pertama adalah keyword `while`, ini harus diisi.
2. Yang kedua adalah `<kondisi>`: ini bisa berupa variabel boolean atau ekspresi logika.
3. Dan yang terakhir adalah blok (atau kumpulan baris) kode yang akan diulang-ulang **kondisi terpenuhi**.

Langsung pada contohnya saja.

Perulangan Tanpa Batas

Perulangan while **sangat berkaitan** dengan variabel boolean, atau *logical statement*. Karena penentuan **kapan suatu blok kode akan diulang-ulang** ditinjau dari `True` or `False` dari suatu pernyataan logika.

Sehingga jika suatu kondisi itu selalu benar, maka perulangannya pun akan selalu di eksekusi.

Perhatikan contoh berikut:

```
while (1 + 2 == 3):
  print('Halo dunia!')
```

Jika dieksekusi, sistem akan mencetak tulisan “Halo dunia!” berkali-kali tanpa henti.

```
...
```

```
Halo dunia!
Halo dunia!
Halo dunia!
Halo dunia!
Halo dunia!
Halo dunia!
Halo dunia!
Halo dunia!
...
```

Kita bisa **memaksanya berhenti dengan menekan tombol** `Ctr + C` jika menggunakan CLI, atau dengan cara menekan tombol stop jika menggunakan IDE atau sejenisnya.

Kenapa perulangan di atas dieksekusi terus menerus?

Karena komputer diperintahkan untuk menulis "Hello World" **selama satu ditambah dua sama dengan tiga**.

Pertanyaannya: apakah satu ditambah dua sama dengan tiga terus-menerus atau tidak?

Jawabannya iya! Oleh karena itu sistem melakukan iterasi tak terbatas.

Contoh perulangan while seperti for + range

Nah, timbul pertanyaan.

Lalu bagaimana caranya agar **bisa memberhentikan** perulangan while?

Caranya buat kondisinya bersifat dinamis (alias bisa berubah-ubah).

Di dalam contoh berikut, akan menampilkan angka 1 sampai dengan angka 5 menggunakan perulangan while.

```
i = 1

while i <= 5:
  print(i)
  i += 1
```

Kode program di atas akan menghasilkan output seperti berikut:

```
1
2
3
4
5
```

Penjelasan

Pada kode program di atas, sistem diinstruksikan untuk:

1. Melakukan perulangan selama variabel `i` kurang dari atau sama dengan `5`.
2. Setiap kali iterasi, sistem akan menampilkan nilai dari `i`.
3. Dan yang terakhir, pada setiap iterasi, sistem akan menambahkan nilai `i` dengan angka `1`.

Bisa memodifikasi kode program di atas misalnya untuk:

- menampilkan bilangan prima dari 1 sampai 100
- menampilkan angka kelipatan 4 dari 1 sampai 100
- menampilkan angka ganjil dari 1 sampai 27
- dan sebagainya

Contoh perulangan while untuk list

Untuk menampilkan semua item pada list, cara yang paling clean adalah dengan menggunakan metode for seperti yang telah dibahas sebelumnya.

Meskipun begitu, tetap bisa menggunakan perulangan while untuk bermain-main dengan list.

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

# bermain index
i = 0
while i < len(listKota):
  print(listKota[i])
  i += 1
```

Jika dijalankan, hasilnya akan terlihat seperti ini:

```
Jakarta
Surabaya
Depok
Bekasi
Solo
Jogjakarta
Semarang
Makassar
```

Juga bisa menggunakan fungsi `list.pop()`. Perhatikan kode program berikut:

```
# bermain pop
while listKota:
  print(listKota.pop(0))
```

Kode program di atas juga akan menghasilkan output yang sama seperti yang dilakukan dengan pendekatan indeks.

Contoh perulangan while dengan inputan

Juga bisa menggunakan while dengan inputan.

Perhatikan contoh di bawah. Pada contoh ini akan meminta user untuk memasukkan angka ganjil lebih dari 50. Jika user justru memasukkan nilai genap atau nilai yang kurang dari 50, maka sistem akan meminta user untuk menginputkan kembali.

```
a = int(input('Masukkan bilangan ganjil lebih dari 50: '))

while a % 2 != 1 or a <= 50:
  a = int(input('Salah, masukkan lagi: '))

print('Benar')
```

Contoh Output:

```
Masukkan bilangan ganjil lebih dari 50: 1
Salah, masukkan lagi: 2
Salah, masukkan lagi: 3
Salah, masukkan lagi: 10
Salah, masukkan lagi: 50
Salah, masukkan lagi: 52
Salah, masukkan lagi: 54
Salah, masukkan lagi: 55
Benar
```

Contoh perulangan while dengan continue

Sama dengan perulangan `for`, juga bisa menggunakan perintah `continue` pada perulangan `while`.

Apa itu perintah `continue`?

Perintah `continue` berfungsi **untuk men-skip** iterasi sekarang ke iterasi selanjutnya.

Contoh:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

# skip jika i adalah bilangan genap
# dan i lebih dari 0
i = -1
while i < len(listKota):
  i += 1
  if i % 2 == 0 and i > 0:
    print('skip')
```

```
    continue

  # tidak dieksekusi ketika continue dipanggil
  print(listKota[i])
```

Output:

```
Jakarta
Surabaya
skip          <-- i sama dengan 2
Bekasi
skip          <-- i sama dengan 4
Jogjakarta
skip          <-- i sama dengan 6
Makassar
skip          <-- i sama dengan 8
```

Pada output di atas, ketika `i`-nya adalah bilangan genap yang lebih dari satu, perintah `print(listKota[i])` tidak dieksekusi dan justru di-skip.

Contoh perulangan while dengan break

Kita juga bisa menggunakan perintah `break` pada perulangan `while`.

Perintah `break` itu sebenarnya mirip dengan perintah `continue`.

Bedanya:

Ketika perintah `break` dipanggil, maka perulangan akan **dihentikan secara paksa**.

Perhatikan contoh berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

kotaYangDicari = input('Masukkan nama kota yang dicari: ')

i = 0
while i < len(listKota):
  if listKota[i].lower() == kotaYangDicari.lower():
    print('Ketemu di index', i)
    break

  print('Bukan', listKota[i])
  i += 1
```

Contoh output:

```
Masukkan nama kota yang dicari: bekasi
Bukan Jakarta
Bukan Surabaya
```

```
Bukan Depok
Ketemu di index 3
```

while ... else

Sama seperti for, kita juga bisa menggunakan blok kode `else` pada perulangan `while`.

Tugasnya pun sama: yaitu untuk mendefinisikan suatu tugas yang akan dieksekusi ketika perulangan telah **selesai secara natural** tanpa dihentikan secara paksa.

Coba ubah program pencarian kota di atas dengan menambahkan blok kode `else` seperti berikut:

```
listKota = [
  'Jakarta', 'Surabaya', 'Depok', 'Bekasi', 'Solo',
  'Jogjakarta', 'Semarang', 'Makassar'
]

kotaYangDicari = input('Masukkan nama kota yang dicari: ')

i = 0
while i < len(listKota):
  if listKota[i].lower() == kotaYangDicari.lower():
    print('Ketemu di index', i)
    break

  print('Bukan ', listKota[i])
  i += 1
else:
  print('Maaf, kota yang anda cari tidak ditemukan.')
```

Coba eksekusi lalu **masukkan kota yang tidak ada** pada variabel `listKota`.

Berikut ini contoh output yang didapatkan:

```
Masukkan nama kota yang dicari: sidoarjo
Bukan Jakarta
Bukan Surabaya
Bukan Depok
Bukan Bekasi
Bukan Solo
Bukan Jogjakarta
Bukan Semarang
Bukan Makassar
Maaf, kota yang anda cari tidak ditemukan.
```

Berbeda jika misal kota yang dicari adalah kota “Depok”:

```
Masukkan nama kota yang dicari: depok
Bukan Jakarta
Bukan Surabaya
Ketemu di index 2
```

Di sini perintah yang ada di blok kode `else` tidak dieksekusi oleh sistem. Kenapa? Karena perulangannya diberhentikan secara paksa dengan perintah `break`, bukan karena berhenti secara natural.

Kapan harus menggunakan for, dan kapan harus menggunakan while?

Sekarang, mungkin masih ada satu pertanyaan lagi yang belum terjawab, yaitu:

Kapan seharusnya menggunakan for? Dan kapan seharusnya menggunakan while?

Sebenarnya tidak ada acuan yang sangat baku, karena banyak sekali kasus-kasus yang bisa diselesaikan dengan menggunakan keduanya.

Tapi, kalau memang ingin sebuah jawaban:

- Kalian bisa **menggunakan for** untuk kasus-kasus **yang berkaitan dengan data *sequence* pada python, atau untuk kasus yang sudah jelas jumlah perulangannya berapa**.
- Dan kalian bisa **menggunakan while** jika memang **perulangannya tidak jelas** akan dilakukan berapa banyak.

Latihan

- Membuat Deret Fibonacci
- Menentukan Bilangan Prima
- Menghitung Jumlah Huruf Vokal
- Menghitung Pangkat Secara Manual
- Prigrim Mingibih Hirif Vikil

Referensi

[1] https://techterms.com/definition/loop – disakses tanggal 17 Mei 2021

[2] https://www.w3schools.com/python/python_for_loops.asp – diakses tanggal 17 Mei 2021

[3] https://www.programiz.com/python-programming/while-loop – diakses tanggal 17 Mei 2021

[4] https://pythonbasics.org/while-loop/ – diakses tanggal 17 Mei 2021