Kontrol Perulangan

Break

Pernyataan break menghentikan perulangan kemudian keluar, dilanjutkan dengan mengeksekusi pernyataan (statement) setelah blok perulangan. Salah satu penggunaannya yang paling sering adalah sebuah kondisi eksternal yang membutuhkan program untuk keluar dari perulangan. Jika Anda memiliki perulangan bertingkat, break akan menghentikan perulangan sesuai dengan tingkatan atau di perulangan mana ia berada. Namun jika ia diletakkan di perulangan dengan kedalaman kedua misalnya, hanya perulangan itu saja yang berhenti, tidak dengan perulangan utama.

Contoh 1:

```
1. for huruf in 'Dico ding':
2.     if huruf == ' ':
3.         break
4.     print('Huruf saat ini: {}'.format(huruf))
```

Output contoh 1:

```
Huruf saat ini: D
Huruf saat ini: i
Huruf saat ini: c
Huruf saat ini: o
```

Contoh 2, cetak bintang dengan memanfaatkan fungsi break

```
1. for i in range (0,10):
2.     for j in range (0,10):
3.         if j>i:
4.             print()
5.             break
6.         else:
7.             print("*",end="")
8.
```

Output contoh 2:

```
**
***
****
*****
******
*******
********
*********
**********
```

Continue

Pernyataan continue akan membuat iterasi saat ini berhenti, kemudian melanjutkan ke iterasi berikutnya, **mengabaikan** pernyataan (statement) yang berada antara continue hingga akhir blok perulangan.

Contoh 1:

```
1. for huruf in 'Dico ding':
2.     if huruf == ' ':
3.         continue
4.     print('Huruf saat ini: {}'.format(huruf))
```

Output contoh 1:

```
Huruf saat ini: D
Huruf saat ini: i
Huruf saat ini: c
Huruf saat ini: o
Huruf saat ini: d ## perhatikan spasi dilewati
Huruf saat ini: i
Huruf saat ini: n
Huruf saat ini: g
```

Contoh 2, cetak bintang yang sama dengan contoh 2 pada pembahasan fungsi break, dengan 1 loop dan 1 if (tanpa else):

```
baris = 0
bintang = 0
while baris < jumlah_baris:
    if (bintang) >= (baris+1):
        print()
        baris = baris+1
        bintang = 0
        continue      # Saat masuk ke if, maka bagian print * diluar if tidak akan dijalankan, langsung ulang ke w
    print("*", end="")
    bintang = bintang+1
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Else setelah For

Pada Python juga dikenal fungsi else setelah for. Fungsinya diutamakan pada perulangan yang bersifat pencarian - untuk memberikan jalan keluar program saat pencarian tidak ditemukan.

Struktur umumnya adalah sebagai berikut:

```
for item in items:
    if cari(item):
        #ditemukan!
        proses_item()
        break
else:
    #Item tidak ditemukan
    not_found_in_container()
```

Saat sebuah perulangan dijalankan, fungsi if akan dievaluasi. Saat ia *pernah sekali saja benar*, maka else tidak akan dieksekusi. Dengan kata lain, if yang berada dalam perulangan *harus selalu salah* untuk memicu blok statemen else dijalankan. Lebih singkat lagi, struktur pseudocode yang diikuti dalam membuat else pada perulangan adalah seperti berikut:

```
if any(something_about(thing) for each thing in container):
    do_something(that_thing)
else:
    no_such_thing()
```

Contoh penggunaan dari for-else dapat dilihat pada potongan kode berikut (jalankan di konsol sebelum melanjutkan ke bagian selanjutnya):

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n/x)
            break
```

Potongan kode di atas melakukan pencarian faktor dari setiap bilangan antara 2 s/d 9. Namun demikian, pada saat bilangan n bernilai 2, 3, 5, 7, program tidak akan mencetak apapun. Ini karena bilangan-bilangan tersebut merupakan bilangan prima. Hal ini bisa diatasi dengan menambahkan keterangan else dan mencetak bahwa bilangan tersebut adalah bilangan prima:

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print( n, 'equals', x, '*', n/x)
            break
    else:
        # loop fell through without finding a factor
        print(n, ' adalah bilangan prima')
```

Else setelah While

Berbeda dengan Else setelah For, pada statement while, blok statement else *akan selalu dieksekusi saat kondisi pada while menjadi salah*. Contoh mudahnya adalah sebagai berikut:

```
while n > 0:
    n = n - 1
    if n == 7:
        break
    print(n)
else:
    print("Loop selesai")
```

Output:

```
9
8
```

Pada contoh diatas, **loop akan di break saat nilai n == 7**, saat keluar dari perulangan, maka python tidak akan memunculkan tulisan Loop selesai, namun jika tidak dilakukan break (perulangan berakhir dengan normal):

```
n = 10
while n > 0:
    n = n - 1
    print(n)
else:
    print("Loop selesai")
```

Output:

```
9
8
7
6
5
4
3
2
1
0
Loop selesai
```

Pass

belum melakukan implementasi (atau menyiapkan tempat untuk implementasi), serta membiarkan program tetap berjalan saat misalnya Anda mengalami kegagalan atau exception.

Pass statement adalah operasi bersifat Null (kosong), tidak ada yang terjadi saat ia dipanggil. Contohnya:

```
1. def sebuahfungsi():
2.     pass
```

Output:

```
#tidak ada
```

Jika Anda mendeklarasi sebuah fungsi tanpa kode apapun, justru akan terjadi error:

```
1. def sebuahfungsi():
```

Output:

```
File "<test.py>", line 2
^
IndentationError: expected an indented block
```

Contoh pass untuk mengantisipasi exception/kegagalan fungsi, perhatikan contoh kode yang belum ditambahkan pass berikut:

```
1. var1=""
2. while(var1!="exit"):
3.     var1=input("Please enter an integer (type exit to exit): ")
4.     print(int(var1))
```

Output:

```
Please enter an integer (type exit to exit): a
Traceback (most recent call last):
File "testp3.py", line 7, in <module>
print(int(var1))
ValueError: invalid literal for int() with base 10: 'a'
```

Kita dapat membaca bahwa program gagal saat mencoba proses konversi variabel var1. Saat program mengalami kegagalan atau exception, ia akan langsung keluar. Bandingkan dengan pendekatan berikut (mengenai exception, import/library sys, dan try/except akan dibahas pada modul-modul pembelajaran berikutnya:

```python
import sys
data=''
while(data!='exit'):
    try:
        data=input('Please enter an integer (type exit to exit): ')
        print('got integer: {}'.format(int(data)))
    except:
        if data == 'exit':
            pass  # exit gracefully without prompt any error
        else:
            print('error: {}'.format(sys.exc_info()[0]))
```

Outputnya saat dijalankan:

```
Please enter an integer (type exit to exit): 1
got integer: 1
Please enter an integer (type exit to exit): a
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): b
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): c
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): exit
```

melakukan antisipasi apapun sebelum menulis pass (misalnya memasukkan informasi terkait error ke log, memberi notifikasi ke admin, dan sebagainya).

List Comprehension (membuat list dengan inline loop dan if)

Ada kalanya Anda perlu membuat sebuah list baru dari operasi list sebelumnya. Misalnya membuat nilai kuadrat dari semua item dalam list:

```
1. #Cara 1
2. angka = [1, 2, 3, 4]
3. pangkat = []
4. for n in angka:
5.   pangkat.append(n**2)
6. print(pangkat)
```

Cara yang umum digunakan adalah cara di atas, yakni melakukan perulangan sejumlah item pada list angka kemudian membuat list baru (pangkat) dan menambahkan hasil operasinya dalam list baru (pangkat). Bandingkan dengan cara berikut:

```
1. #Cara 2 List Comprehension
2. angka = [1, 2, 3, 4]
3. pangkat = [n**2 for n in angka]
4. print(pangkat)
```

Cobalah kedua contoh di atas. Hasilnya sama bukan?

Output:

```
[1, 4, 9, 16]
```

List comprehension adalah salah satu cara untuk menghasilkan list baru berdasarkan list atau iterables yang telah ada sebelumnya. Sintaksis dasarnya adalah sebagai berikut:

```
1. new_list = [expression for_loop_one_or_more conditions]
```

Contoh kedua di atas dapat diartikan sebagai: untuk setiap anggota angka buatlah nilai kuadratnya comprehension dapat digunakan lebih lanjut, misalnya:

```
list1 = ['d', 'i', 'c', 'o']
list2 = ['d', 'i', 'n', 'g']
duplikat = []
for a in list1:
  for b in list2:
    if a == b:
      duplikat.append(a)

print(duplikat)  # Output ['d','i']
```

Bandingkan dengan:

```
#Contoh4 Implementasi dengan list comprehension
list1 = ['d', 'i', 'c', 'o']
list2 = ['d', 'i', 'n', 'g']
duplikat = [a for a in list1 for b in list2 if a == b]
print(duplikat) # Output: ['d','i']
```

Sangat memudahkan bukan? Dari 5 baris cukup disingkat menjadi 1 baris saja.

Contoh penggunaan yang lain dapat dilihat pada contoh 5 berikut:

```
# Contoh 5 kecilkan semua huruf
list_a = ["Hello", "World", "In", "Python"]
small_list_a = [_.lower() for _ in list_a]
print(small_list_a)
```

Output:

```
['hello', 'world', 'in', 'python']
```

Anda tidak perlu bingung saat melihat kode di Internet yang menuliskan seperti contoh di atas, karena garis bawah (underscore) **termasuk penamaan variabel yang valid**. **Secara umum "_" biasa digunakan sebagai throwaway variable (variabel tidak penting)**.

Banyak hal yang mungkin dilakukan dengan list comprehension. Coba perkirakan apa yang akan dimunculkan oleh potongan kode berikut dan pilih jawaban yang tepat pada kuis di bagian berikutnya:

```
2.  x = [[a**2, a**3] for a in list_a]
3.  print(x)
```

Sebelumnya

Selanjutnya