Modul 2
Operator dan Ekspresi

Hampir semua statemen (baris logis) yang Anda tulis akan mengandung *ekspresi*. Contoh sederhana dari ekspresi adalah `2+3`. Sebuah ekspresi dapat diturunkan menjadi operator dan operand.

Operator adalah fungsi yang menjalankan sesuatu dan direpresentasikan oleh simbol, seperti `+` atau kata kunci khusus. Operator membutuhkan data untuk dioperasikan dan data ini disebut *operand*. Dalam kasus ini `2` dan `3` adalah operand.

Operator

Kita akan melihat operator secara singkat dan bagaimana penggunaannya:

Operator	Keterangan
+	Menambahkan dua obyek
-	Mengurangi obyek dengan obyek yang lain
*	Perkalian
**	Pangkat
/	Pembagian
//	Pembagian bulat ke bawah
%	Sisa hasil bagi (modulus)
<<	(geser kiri) Menggeser bit ke sebelah kiri sesuai dengan jumlah bit yang ditentukan. `2 << 2` menghasilkan `8`. `2` direpresentasikan `10` dalam bit (binary digit). Menggeser 2 bit kekiri akan menghasilkan `1000` yang merupakan representasi dari desimal `8`.
>>	(geser kanan) Menggeser bit ke sebelah kanan sesuai dengan jumlah bit yang ditentukan. `11 > 1` menghasilkan `5`. `11` direpresentasikan oleh bit dengan `1011` kemudian digeser kekanan 1 bit menghasilkan `101` yang merupakan desimal angka `5`.
&	(bit-wise AND) Operasi bit-wise AND dari angka (bit-wise adalah operasi angka berbasis bit yakni dengan `0` dan `1`). `5 & 3` menghasilkan `1`.
|	(bit-wise OR) Operasi bit-wise OR dari angka. `5 | 3` menghasilkan `7`.
^	(bit-wise XOR) Operasi bit-wise XOR (ekslusif OR). `5 ^ 3` menghasilkan `6`.
~	(bit-wise invert) Operasi membalikkan angka bitwise dari `x`, menghasilkan `-x - 1`. `~5` akan menghasilkan `-6`. lihat two's complement.
<	(kurang dari)

Operator	Keterangan
	Mengembalikan apakah x kurang dari y. Semua operator pembanding mengembalikan `True` atau `False`. `5 < 3` mengembalikan `False`, `3 < 5` mengembalikan `True` dan `2 < 5 < 7` mengembalikan `True`.
>	(lebih dari) Mengembalikan apakah x lebih dari y. `5 > 3` mengembalikan `True`.
<=	(kurang dari atau sama dengan) Mengembalikan apakah x kurang dari atau sama dengan y. `5 <= 5` mengembalikan `True`.
>=	(lebih dari atau sama dengan) Mengembalikan apakah x lebih dari atau sama dengan y. `5 >= 5` mengembalikan `True`.
==	(sama dengan) Membandingkan apakah kedua obyek sama. `2 == 2` mengembalikan `True`, `'nama' == 'Nama'` mengembalikan `False`, `'nama' == 'nama'` mengembalikan `True`.
!=	(tidak sama dengan) Membandingkan apakah kedua obyek berbeda. `2 != 3` mengembalikan `True`.
not	(boolean NOT) Jika x bernilai `True` akan mengembalikan `False`. Jika x bernilai `False` akan mengembalikan `True`. `x = True; not x` mengembalikan `False`.
and	(boolean AND) `x and y` mengembalikan `False` jika x bernilai `False`, selain itu akan mengembalikan nilai `y`. `x = False; y = True; x and y` akan mengembalikan `False` karena `x` bernilai `False`. Pada kasus ini Python tidak akan mengevaluasi `y` karena nilai `x`. Hal ini disebut *short-circuit* evaluasi.
or	(boolean OR) Jika `x` bernilai True, `x or y` akan mengembalikan `True`, selain itu akan mengembalikan nilai `y`. `x = True; y = False; x or y` mengembalikan `True`. *short-circuit* evaluasi berlaku juga disini.

```
# lat6.py
# Operator dan ekspresi

bilangan1 = 5
bilangan2 = 3

print ('bil1 = ', bilangan1)
print ('bil2 = ', bilangan2)
```

```
print ('bil1 + bil2 = ', bilangan1 + bilangan2)
print ('bil1 - bil2 = %s' % (bilangan1 - bilangan2))
print ('bil1 * bil2 = {0}'.format(bilangan1 * bilangan2))
print ('bil1 ** bil2 = ', bilangan1 ** bilangan2)

bilangan1 = 5.0
print ('bil1 = ', bilangan1)
print ('bil2 = ', bilangan2)

print ('bil1 / bil2 = ', bilangan1 / bilangan2)
print ('bil1 // bil2 = ', bilangan1 // bilangan2)
print ('bil1 % bil2 = ', bilangan1 % bilangan2)

print ('-' * 80)

bilangan1 = 5
print ('bil1 = ', bilangan1)
print ('bil2 = ', bilangan2)

print ('bil1 << bil2 = ', bilangan1 << bilangan2)
print ('bil1 >> bil2 = ', bilangan1 >> bilangan2)
print ('bil1 & bil2 = ', bilangan1 & bilangan2)
print ('bil1 | bil2 = ', bilangan1 | bilangan2)
print ('bil1 ^ bil2 = ', bilangan1 ^ bilangan2)
print ('~bil1 = ', ~bilangan1

print ('-' * 80)

print ('bil1 < bil2 = ', bilangan1 < bilangan2)
print ('bil1 > bil2 = ', bilangan1 > bilangan2)
print ('bil1 <= bil2 = ', bilangan1 <= bilangan2)
print ('bil1 >= bil2 = ', bilangan1 >= bilangan2)
print ('bil1 == bil2 = ', bilangan1 == bilangan2)
print ('bil1 != bil2 = ', bilangan1 != bilangan2)

print ('-' * 80)

print ('not True = ', not True)
print ('True and False = ', True and False)
print ('True or False = ', True or False)
```

Cara lain operasi matematika dan pengisian variabel

Ketika melakukan operasi matematika, kita sering setelah dilakukan operasi hasil tersebut kita simpan dalam variabel. Di python ada jalan pintas untuk melakukan operasi dan melakukan *assignment*.

Anda bisa menulis:

```
a = 2
a = a * 3
```

sebagai:

```
a = 2
a *= 3
```

Berikut latihan 7 untuk menghitung uang kembalian.

```
# lat7.py

total_uang = 10000
harga_barang = 5000
diskon = 0.10

# harga barang setelah diskon
harga_barang *= (1 - diskon)

total_uang -= harga_barang

print ('total uang = %s' % total_uang)
```

Urutan Evaluasi

Jika ada rantaian ekspresi seperti `2 + 3 * 4`, apakah penambahan dilakukan terlebih dahulu atau perkalian? Saat pelajaran matematika kita diajari bahwa perkalian harus dikerjakan terlebih dahulu. Hal ini menandakan perkalian mempunyai urutan lebih tinggi daripada penambahan.

Berikut tabel urutan evaluasi ekspresi dalam Python, dari terrendah sampai tertinggi.

Operator	**Keterangan**
lamda	Ekspresi `lambda`
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in	Tes Keanggotaan
is, is not	Tes Identitas
<, <=, >, >=, !=, ==	Perbandingan
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shift
+, -	Penambahan dan Pengurangan
*, /, //, %	Perkalian, Pembagian, Pembagian ke bawah, mod
+x, -x	Positif, Negatif
~x	Bitwise NOT / inverse
**	Pangkat
x.attribute	Referensi atribut
x[index]	Akses item
x[index1:index2]	Slicing
f(argument ...)	Pemanggilan fungsi
(ekspresi, ...)	literal tuple
[ekspresi, ...]	literal list
{key:value, ...}	literal dictionary

Mengubah Urutan Evaluasi

Untuk membuat ekspresi lebih mudah dibaca, kita dapat menggunakan tanda kurung. Sebagai contoh, `2 + (3 * 4)` lebih mudah dipahami daripada `2 + 3 * 4` dimana pembaca harus mengetahui urutan evaluasi operator. Namun pemakaian tanda kurung jangan terlalu berlebihan seperti `(2 + (3 * 4))`.

Selain itu, tanda kurung dapat mengubah urutan evaluasi operator. Sebagai contoh `(2 + 3) * 4`, operasi penambahan akan dievaluasi terlebih dahulu.

```
# lat8.py

hasil = 2 + 3 * 4
print ('2 + 3 * 4 = %s' % hasil)

hasil = (2 + 3) * 4
print ('(2 + 3) * 4 = %s' % hasil)

hasil = 2 / 3 * 4
print ('2 / 3 * 4 = %s' % hasil)

hasil = 2.0 / 3 * 4
print ('2.0 / 3 * 4 = %s' % hasil)
```

Sifat Asosiatif

Operator dengan level urutan evaluasi yang sama akan dievaluasi dari kiri ke kanan. Sebagai contoh `2 + 3 + 4` akan dievaluasi sebagai `(2 + 3) + 4`. Beberapa operator seperti pengisian nilai (assignment) mempunyai sifat asosiatif dari kanan ke kiri, contoh: `a = b = c` akan dievaluasi `a = (b = c)`.

Menerima input dari user

Terkadang program kita membutuhkan input dari user secara langsung. Input dari user tersebut kemudian kita proses dalam program yang kita buat untuk keperluan tertentu.

Dalam bahasa pemrograman python, cara untuk menerima input user dari terminal cukup sederhana. Kita cukup menggunakan fungsi input() dengan syntax seperti dibawah ini.

```
var_input = input('Masukkan input anda kemudian tekan ENTER ')
```

Dengan syntax di atas, program yang kita buat akan menunggu user mengetikkan sesuatu dan menekan tombol ENTER. Setelah user menekan tombol ENTER, apa yang diketikkan akan disimpan sebagai teks pada variabel yang kita siapkan (pada contoh variabelnya bernama var_input)

Coba jalankan program di bawah ini untuk lebih jelasnya:

```
nama = input('Namanya siapa? ')
usia = input('Umurnya berapa tahun? ')
```

```
print('Halooo ',nama, ' yang umurnya sudah ', usia, 'tahun')
```

Hasil dari perintah input adalah variabel bertipe data teks

Pada python 3, apapun yang dimasukkan user, baik berupa teks, angka (numerik), maupun yang lain, akan selalu dianggap sebagai teks. Jika kita menginginkan tipe data yang lain, maka kita harus melakukan konversi terlebih dahulu. Jalankan contoh di bawah untuk lebih jelasnya.

```
angka1 = input('Masukkan angka pertama ')
angka2 = input('Masukkan angka kedua ')

jumlah = angka1 + angka2 # penjumlahan sebelum konversi
print('Hasil penjumlahan sebelum konversi: ', jumlah)

angka1 = int(angka1) # konversi nilai variabel angka1 ke tipe integer
angka2 = int(angka2) # konversi nilai variabel angka1 ke tipe integer

jumlah = angka1 + angka2 # penjumlahan setelah konversi
print('Hasil penjumlahan setelah konversi: ', jumlah)
```

Latihan:

1. Membuat program hitung luas Segitiga:

```
# hitung_segitiga_input.py
# program hitung luas segitiga

alas = 10
tinggi = 20

#hitung luas
luas = (alas * tinggi) / 2

# menampilkan hasil
print('Luas segitiga adalah %0.2f' %luas)
```

2. Membuat program hitung luas Segitiga dengan input:

```
# hitung_segitiga.py
# program hitung luas segitiga
alas = float(input('Tuliskan Alas Segitiga: '))
tinggi = float(input('Tuliskan Tinggi Segitiga: '))

# hitung luas

luas = (alas * tinggi) / 2

# menampilkan hasil
print('Luas segitiga adalah %0.2f' %luas)
```

3. Program Menghitung Volume balok:

```
# Menginput Panjang, lebar dan Tinggi Balok
p = int(input('masukan panjang balok: '))
l = int(input('masukan lebar balok: '))
t = int(input('masukan tinggi balok: '))

# Hitung Luas Permukaan
L = 2 * ( (p*l) + (p*t) + (l*t) )
# Hitung Volume Balok
V = p * l * t
print('Jadi balok dengan ukuran panjang:{}, lebar:{},
tinggi:{} Mempunyai luas:{} dan volume:{} '.format(p,l,t,
L, V))
```

Tugas :

1. Buatlah program untuk menyelesaikan persamaan kuadrat **ax2 + bx + c = 0**.
2. Buatlah program kalkulator sederhana untuk melakukan operasi penjumlahan, pengurangan, pembagian dan perkalian dengan inputan 2 buah bilangan.