BAB 02

CASAR
PEMROGRAMAN PYll-lON
(BAGIAN PERTAMA)

Pokok bahasan pada bab ini mencakup hal-hal berikut:

- interpreter Python;
- skrip python;
- komentar pada skrip Python;
- penamaan pengenal;
- literal, konstanta, dan variabel;
- ekspresi;
- penulisan satu perintah ke lebih dari satu baris;
- penulisan lebih dari satu perintah dalam satu baris;
- pelibatan modul eksternal;
- objek, metode, dan fungsi;
- pembacaan data dari papan ketik.

2.1 Interpreter Python

Python merupakan bahasa pemrograman yang diproses melalui interpreter Python. Sebagai interpreter, Python dapat menerima perintah secara interaktif dan langsung mengeksekusinya. Setelah Python diinstal, interpreter Python dapat dipanggil dengan memberikan perintah python di Command Prompt. Hasilnya diperlihatkan pada Gambar 2.1.

Gambar g.J InteI]Jreter Python

Tanda »> dinamakan *prompt.* Tanda ini mengisyaratkan bahwa interpreter Python siap menerima perintah secara interaktif. Dalam hal ini, setiap kali suatu perintah yang diakhiri dengan penekanan tombol Enter akan membuat interpreter Python mengeksekusinya.

Sebagai contoh, tulisan "Selamat belajar Python" dapat ditampilkan dengan cara menuliskan perintah seperti berikut:

```
>>> print('Selamat belajar Python')
Selamat belajar Python
```

Tampak bahwa setelah tombol Enter ditekan, argumen 'Selamat belajar Python' yang berada pada fungsi print() ditampilkan sebagai akibat pernyataan print ('Selamat belajar Python') telah dieksekusi.

Tulisan 'Selamat belajar Python' dikenal dengan istilah string. String berarti sederet karakter. Dalam praktik, string dapat tidak mengandung karakter sama sekali dan dinamakan string kosong, tetapi juga bisa mengandung satu karakter atau sejumlah karakter. Secara lebih khusus, 'Selamat belajar Python' disebut literal string. Adapun yang dimaksud

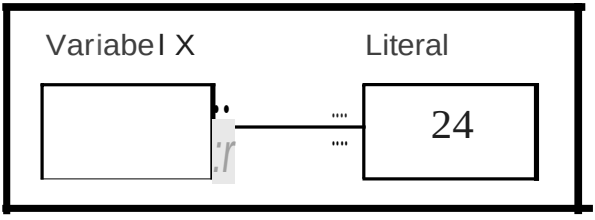
dengan karakter dapat berupa satu huruf, satu digit (0..9), atau satu simbol seperti + dan *.

Sebagaimana telah disinggung pada Bab 1, pernyataan adalah istilah yang menyatakan perintah. Kode `print ('Selamat belajar Python')` merupakan contoh perintah. Secara prinsip, setiap kode yang dapat dieksekusi oleh Interpreter Python dinamakan pernyataan.

Contoh lain pernyataan berupa:

```
x = 24
```

Dalam hal ini, pernyataan tersebut dimaksudkan untuk menugaskan nilai 24 agar dirujuk oleh variabel x. Gambar 2.2 memperjelas efek pernyataan ini. Dalam hal ini, baik variabel x maupun konstanta bilangan 24 diletakkan pada memori tersendiri di komputer.



Gambar 2.1 Variabel x mewakili konstanta

Hal yang menarik, x yang diikuti dengan penekanan tombol Enter juga akan berkedudukan sebagai pernyataan karena interpreter Python dapat mengeksekusinya. Contoh:

```
>>> x = 24
>>> x
24
>>>
```

Tampak bahwa pemanggilan x menampilkan nilai yang diwakili oleh variabel x.

2.2 Skrip Python

Dalam praktik, sejumlah pernyataan ditulis dalam suatu berkas. Berkas skrip Python mempunyai ekstensi .py. Sebagai contoh, pada Bab 1 telah diperkenalkan satu skrip dengan nama tes. py.Berkas semacam ini dapat ditulis menggunakan sembarang editor teks. Namun, sebaiknya editor yang menyediakan penomoran baris yang digunakan karena ketika interpreter Python menemukan kesalahan sewaktu mengeksekusi pernyataan, nomor baris yang menjadi penyebab kesalahan akan dicantumkan. Sebagai contoh, notepad++ bisa dipakai. Editor ini dapat diunduh secara gratis di https://notepad-plus-plus. org.

Berikut adalah contoh skrip Python yang digunakan untuk menampilkan gambar robot:

11!1 Berkas : robot.py

```
print('*********')
print (' *         *')
print ('*   O   O  *')
print ('*          *')
print ('*          *')
print ('*          *')
print('**********')
print ('  | |     | |')
print ('  | |     | |')
t (' ==    ==')
```

Akhir berkas

Skrip ini disimpan dengan nama robot.pydan disimpan pada folder C: \LatOpenCV. Pemanggilan dapat dilakukan di Command Prompt dengan memberikan perintah:

```
python robot.py
```

Hasil pemanggilan melalui Command Prompt adalah seperti berikut:

```
C:\LatOpenCV>python robot.py
*********
*       *
* 0   0 *
*       *
*       *
*       *
*********
 ||   ||
 ||   ||


C:\LatOpenCV>
```

2.3 Komentar pada Skrip Python

Komentar biasa disertakan pada skrip Python dengan tujuan untuk memberikan penjelasan kepada pembuat skrip ataupun pembaca skrip dan tidak ditujukan kepada komputer sama sekali. Komentar ditulis dengan awalan # dan bisa diletakkan di mana saja. Dengan sendirinya, sejak tanda tersebut hingga akhir baris akan diabaikan oleh interpreter Python karena tidak berfungsi sebagai perintah. Conteh:

```
# Pengolahan gambar untuk meningkatkan kecerahan
print('\n') # Pindah baris
# print('\n')
```

Pada contoh pertama, komentar mencakup satu baris. Pada contoh kedua, komentar berada setelah pernyataan `print('\n')`. Pada contoh ketiga, `print('\n')` bukan suatu pernyataan, melainkan bagian dari komentar. Cara ketiga ini seringkali digunakan untuk mematikan perintah secara pertama sewaktu pengujian skrip dilakukan.

Di lingkungan Linux, seringkali terdapat baris yang menyerupai komentar, yang berisi semacam berikut:

```
#!/usr/local/bin/python
```

Hal ini digunakan untuk mengisyaratkan lokasi interpreter Python (yang bernama python) berada di /usr/local/bin. Hal ini memungkinkan pemanggilan nama skrip secara langsung di *shell prompt.*

2.4 Penamaan Pengenal

Pengenal *(identifier)* digunakan pada berbagai elemen di Python seperti untuk memberi nama variabel ataupun fungsi. Oleh karena itu, pemahaman terhadap pemberian nama pengenal sangat penting.

Secara prinsip, aturan yang berlaku dalam memberikan nama pengenal mudah untuk dipahami. Aturan-aturannya seperti berikut:

1) huruf (A-Z, a-z), digit (0-9), dan garis-bawah (_) dapat digunakan untuk membentuk nama pengenal, akan tetapi nama pengenal tidak boleh berawalan digit;
2) huruf kecil dan huruf kapital dibedakan sehingga pengenal seperti harga dan Harga menyatakan pengenal yang berbeda;
3) kata-kunci *(keyword)* seperti for dan True yang sudah digunakan secara khusus oleh Python tidak diperkenankan untuk digunakan sebagai nama pengenal.

Tabel 2.1 mencantumkan contoh pengenal yang absah dan tidak absah.

label 2.1 Contoh pengenal yang absah dan tidak absah

Pengenal Absah	Pengenal Tidak Absah
_register	nama-barang (Tanda - tidak diperkenankan).
A	class (Merupakan kata-tercadang. Tidak boleh digunakan).
A (berbeda dengan a)	kuartal 2 (Spasi tidak boleh digunakan).
hargaBarang	2mobil (Angka tidak boleh berada di depan).
harga_barang	harga$ (Tanda $ tidak diperkenankan).
kuartal2	a+b (Tanda + tidak diperkenankan).

2.5 Literal, Konstanta, dan Variabel

Subbab ini memberikan dasar-dasar mengenai literal, konstanta, dan variabel.

2.5.1 Literal

Literal menyatakan nilai mentah yang biasa diberikan ke konstanta atau variabel. Literal dasar yang dibahas di sini mencakup bilangan, *Boolean,* dan string. Selain itu, literal khusus bernama None juga diperkenalkan.

Secara garis besar literal bilangan dapat dibedakan menjadi tiga jenis, yaitu literal bilangan bulat, bilangan real, dan bilangan kompleks.

Literal bilangan bulat dapat ditulis dalam berbagai sistem bilangan. Sistem bilangan yang lazim adalah sistem desimal atau yang dikenal dengan nama sistem bilangan berbasis 10, yang menggunakan digit 0 hingga 9. Tabel mencantumkan aturan penulisan konstanta untuk berbagai sistem bilangan.

Tabel 2.2 Berbagai konstanta bilangan bulat

Sistem Bilangan	Keterangan
Desimal	Pada sistem bilangan yang berbasis 10 ini, digit yang berlaku adalah dari O hingga 9.
Oktal	Pada sistem bilangan yang berbasis 8 ini, digit yang berlaku adalah dari O hingga 7. Penulisan konstanta diawali dengan angka O diikuti dengan huruf O kecil maupun kapital.
Heksadesimal	Pada sistem bilangan yang berbasis 16 ini, digit yang berlaku adalah dari O hingga 9 serta huruf A hingga F atau a hingga f. Penulisan konstanta diawali dengan angka Ox atau OX.
Biner	Pada sistem bilangan yang berbasis 2 ini, digit yang berlaku adalah dari O dan 1. Penulisan konstanta diawali dengan angka OB atau Ob .

Contoh berikut memperlihatkan berbagai cara untuk menampilkan bilangan 10:

```
>>> print(10)  4)
10
>>> print(0xA)  4)
10
>>> print(0xa)  4)
10
>>> print(0o12)  4)
10
>>> print(0O12)  4)
10
>>> print(0b1010)  4)
10
>>> print(0B1010)  4)
10
>>>
```

Pada contoh ini, 0xA dan 0xa adalah literal dalam sistem heksadesimal untuk bilangan 10, 0o12 dan 0012 adalah literal dalam sistem oktal untuk bilangan 10, dan 0bl0l0 dan 0B1010 adalah literal dalam sistem biner untuk bilangan 10. Literal bilangan real ditulis dengan menggunakan tanda titik untuk menyatakan bagian pecahan. Sebagai contoh, nilai n ditulis menjadi:

3.14

Literal 1.2E+2 berarti $1{,}2 \times 10^{2}$, sedangkan literal 1.2E-2 berarti $1{,}2 \times 10^{-2}$. Literal dengan notasi E atau e ini disebut literal eksponensial atau literal dengan notasi sains.

Contoh pengujian di Python:

```
>>> print(3.14) {}
3.14
>>> print(l.2E+2) {}
120.0
>>> print(l.2E-2) {}
0.012
>>>
```

Literal bilangan kompleks ditulis dengan notasi:

a+ bj

Dalam hal ini, bagian *a* disebut bilangan real dan *b* dinamakan bilangan imajiner. Contoh:

```
>>> print(2 - Sj) {}
(2-5j)
>>> print(Sj) {}
8j
>>>
```

Literal *Boolean* menyatakan literal dengan nilai logika benar atau salah. Python menyediakan dua literal *Boolean* berupa True (menyatakan nilai benar) dan False (menyatakan nilai salah). Contoh:

```
>>> print(True) {}
True
>>> print(Fa1se) {}
False
>>>
```

Literal string ditulis dengan awalan dan akhiran " (petik ganda) atau ' (petik tunggal). Contoh:

```
>>> print("Tes..tes.123") {}
Tes ..tes .123
>>> print('Tes..tes.123') {}
Tes ..tes .123
>>>
```

String kosong adalah string yang tidak mengandung karakter sama sekali. Konstanta string kosong ditulis dengan awalan ' dan langsung diakhiri dengan 'atau diawali " dan langsung diakhiri dengan ". Contoh:

```
''
""
```

Apabila string mengandung karakter " atau ', perlu digunakan awalan dan akhiran yang berbeda dengan karakter tersebut. Contoh:

```
>>> print("'Que sera sera', ia berkata")  (/}
'Que sera sera', ia berkata
>>> print('"Que sera sera", ia berkata')  (/}
"Que sera sera", ia berkata
>>>
```

Alternatif lain, notasi \" atau \' bisa dipakai tanpa memerhatikan awalan dan akhiran yang digunakan. Contoh:

```
>>> print("\"Que sera sera\", ia berkata") (/}
"Que sera sera", ia berkata
>>> print('\'Que sera sera\', ia berkata') (/}
'Que sera sera', ia berkata
>>>
```

Karakter yang ditulis dengan awalan \ biasa dinamakan karakter *escape.* Tabel 2.3 mencantumkan karakter-karakter *escape.*

Tabel 2.3 Daftar karakter *escape*

Karakter *Escape*	Keterangan
\\	*Backslash*
\'	Petik tunggal
\"	Petik ganda
\b	*Backspace*
\e	*Escape*
\0	*Null*

Karakter *Escape*	Keterangan
\n	*Linefeed*
\v	Tab vertikal
\t	Tab horizontal
\r	*Carriage return*
\f	*Form feed*
\ooo	Karakter dengan nilai oktalnya adalah ooo
\xhh	Karakter dengan nilai heksadesimalnya adalah hh

Contoh penggunaan karakter escape "\n" ditunjukkan berikut ini:

```
>>> print("Rajawali\nBangau\nMerpati")
Rajawali
Bangau
Merpati
>>>
```

Pada contoh ini, karakter "\n" membuat setiap nama burung diletakkan pada baris tersendiri. Itulah sebabnya, "\n" sering disebut karakter pindah baris mengingat efeknya adalah memindahkan string yang mengikutinya ke baris berikutnya.

Literal bernama `None` merupakan konstanta spesial yang berarti belum ada. Contoh:

```
>>> mobil = None
>>> print(mobil)
None
>>>
```

2.5.2 Konstanta

Konstanta adalah jenis variabel dengan nilai yang diwakilinya tidak dimaksudkan untuk diubah setelah ditugasi dengan suatu nilai. Hal seperti ini sering dilakukan sekiranya literal yang sama digunakan di

beberapa bagian dalam suatu skrip sehingga perlu diwakili dengan konstanta.

Secara konvensional, konstanta yang dinyatakan dengan nama ditulis dengan menggunakan pengenal yang semua hurufnya berupa huruf kapital. Misalnya, GRAVITASI adalah suatu konstanta. Hal ini dipakai untuk memberikan isyarat kepada pemrogram atau pembaca skrip bahwa GRAVITASI (karena ditulis dengan huruf kapital seluruhnya) adalah nama konstanta, bukan variabel. Secara teknis, tidak ada perbedaan antara pembuatan konstanta dan variabel.

Contoh konstanta:

```
>>> GRAVITAS! = 9.8
>>> print(GRAVITASI)
9.8
>>>
```

2.5.3 Tipe Data

Sebelum ini, Anda telah mengenal string, bilangan, dan bahkan *Boolean.* Sebenarnya, istilah-istilah tersebut menyatakan tipe data. Adapun nama spesifik tipe data, antara lain:

- int (bilangan bulat);
- float (bilangan real);
- complex (bilangan kompleks)
- str (string);
- bool *(Boolean).*

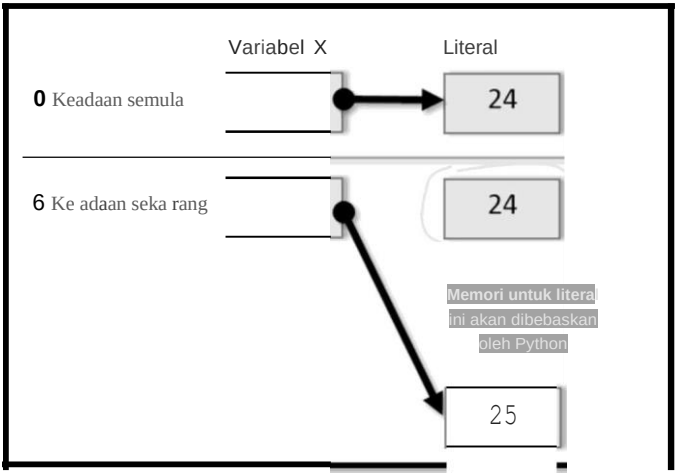
Contoh untuk mendapatkan tipe data sejumlah literal dapat dilihat berikut ini:

```
>>> type (2)
<class 'int'>
>>> type(Ob101)
<class 'int'>
>>> type(Oo123)
<class 'int'>
>>> type (OxA2)
<class 'int'>
>>> type (1. 2)
<class 'float'>
>>> type(l + 2j)
<class 'complex'>
>>> type ( "Ba1o")
<class 'str'>
>>> type (True)
<class 'bool'>
>>>
```

Pemanggilan `type()` pada berbagai konstanta menunjukkan bahwa semua konstanta pun diperlakukan sebagai objek di Python. Objek adalah instan kelas. Kelas dapat dibayangkan seperti suatu cetakan. Dengan cetakan ini, banyak objek yang bisa dibentuk.

2.5.4 Variabel dan Penugasan Nilai

Variabel merupakan suatu objek yang memungkinkan untuk mewakili suatu nilai dan nilai yang diwakilinya bisa berubah setiap saat. Pada saat nilai yang diwakilinya berubah, nilai semula yang dirujuknya sebenarnya tidak berubah. Gambar 2.3 memperjelas hal ini.



Gambar £.S Perubahan nilai yang diwakili variabel

Tampak bahwa semula variabel x mewakili nilai 24. Setelah itu, variabel ini mewakili angka 25. Perhatikan bahwa literal 24 tetap berada di memori pada saat terjadi perubahan pada variabel. Namun, kelak angka 24 tersebut akan dibebaskan dengan sendirinya oleh interpreter Python mengingat tidak ada lagi yang merujuknya melalui mekanisme yang dinamakan pengumpulan sampah.

Secara prinsip, nama variabel bersifat bebas sepanjang memenuhi syarat penamaan pengenal. Namun, nama variabel sebaiknya menyiratkan data yang diwakilinya. Selain itu, jika nama mengandung lebih dari satu kata, model punuk unta bisa digunakan. Pada model punuk unta, semua huruf dinyatakan dengan huruf kecil kecuali untuk awal kata kedua, ketiga, dan seterusnya. Contoh nama variabel yang mengikuti model punuk unta:

- `namaBarang`
- `hargaBarangPerUnit`
- `jumlahPenjualanKuartal2`

Penugasan suatu nilai ke suatu variabel dilakukan melalui operator penugasan dengan bentuk seperti berikut:

variabel = nilai

Jadi, harap diperhatikan bahwa tanda sama dengan pada posisi tersebut bukanlah untuk melakukan pembandingan seperti pada notasi matematika.

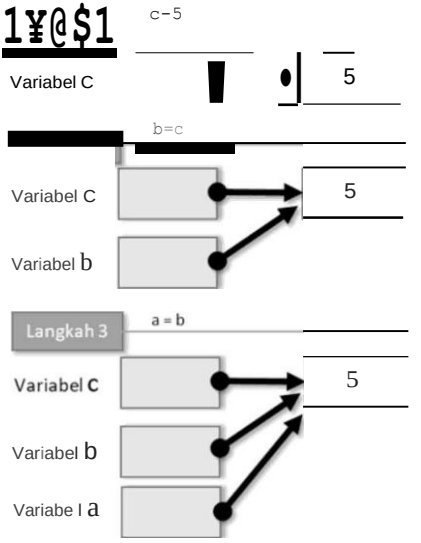
Contoh penugasan pada variabel:

```
>>> usia = 21  4)
>>> print(usia)  4)
21
>>> usia = 22  4)
>>> print(usia)  4)
22
>>>
```

Penugasan dengan bentuk seperti berikut juga dimungkinkan:

```
>>> a = b = c = 5  4)
>>> print(a)  4)
5
>>> print(b)  4)
5
>>> print(c)  4)
5
>>>
```

Dengan cara seperti itu, terlihat bahwa baik a, b, maupun c merujuk ke nilai yang sama yaitu 5. Namun, mekanisme yang terjadi sebenarnya adalah seperti yang dilukiskan pada Gambar 2.4. Pertama-tama, nilai 5 ditugaskan ke c. Kemudian, nilai pada c ditugaskan ke b. Terakhir, nilai pada b ditugaskan ke a. Jadi, pada saat b = c diproses, b akan merujuk ke nilai yang dirujuk oleh c. Dengan demikian, b dan c merujuk ke angka yang sama. Hal yang sama berlaku untuk a = b.



Gambar** £.4 **Penugasan a = b = c = 5

Bentuk lain penugasan yang juga bersifat unik di Python adalah semacam berikut ini:

```
x, y, z = 1, 2, 3
```

Pada bentuk seperti ini, nilai 1 ditugaskan ke x, 2 ke y, dan 3 ke z. Hal ini diperlihatkan pada contoh berikut:

```
>>> x, y, z = 1, 2, 3
>>> print (x)
1
>>> print(y)
2
>>> print(z)
3
>>>
```

Berdasarkan hal tersebut, dimungkinkan untuk memberikan perintah semacam berikut:

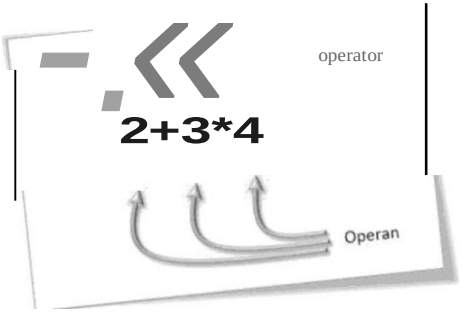
```
x, y = y, x
```

Dengan cara seperti itu, nilai yang diwakili oleh x dan y dipertukarkan. Hal ini diperlihatkan pada contoh berikut:

```
>>> x = 1
>>> y = 2
>>> print("x semula: " x)
x semula:  1
>>> print("y semula: " y)
y semula:  2
>>> x, y = y, x
>>> print("x sekarang: " x)
x sekarang:  2
>>> print("y sekarang: " y)
y sekarang:  1
>>>
```

2.6 Ekspresi

Ekspresi merupakan suatu cara untuk mendapatkan nilai dengan melibatkan operan dan operator. Gambar 2.5 memperlihatkan contoh suatu ekspresi untuk menghitung hasil penjumlahan bilangan 5 dan 6. Pada contoh ini, + adalah operator dan 5 serta 6 berkedudukan sebagai operan. Operator biasanya berupa simbol seperti + dan - tetapi juga bisa berupa kata seperti in. Operan dapat berupa konstanta, variabel, pemanggilan fungsi, atau bahkan berupa ekspresi.



Gambar .5 Ekspresi, operator, dan operan

Dalam praktik, ekspresi tidak selalu berupa operasi aritmetika seperti pada contoh di depan. Namun, suatu ekspresi dapat berupa operasi pembandingan, pengaksesan pada suatu string, atau berbagai operasi lain pada struktur data, seperti senarai dan tupel.

2.6.1 Ekspresi Aritmetika

Ekspresi aritmetika adalah ekspresi yang digunakan untuk melakukan operasi aritmetika seperti penjumlahan, perkalian, dan pengurangan. Terkait dengan ekspresi aritmetika, tersedia sejumlah operator aritmetika yang tercantum pada Tabel 2.4.

Tabel 2.4 Daftar operator aritmetika

Operator	Keterangan	Contoh
**	Perpangkatan	2 ** 3 8 25 ** 0.5 5.0
-	Negasi	-5
*	Perkalian	2 * 3 6 2.5 * 3 7.5
/	Pembagian pecahan	3 / 2 1.5
//	Pembagian bulat (Tipe hasil menyesuaikan tipe operan)	31/2 1 8ll 3 2
%	Sisa pembagian	3 % 2 1
+	Penjumlahan	8 % 3 2
-	Pengurangan	8-2 6

Pengujian ekspresi aritmetika dapat dilakukan secara langsung tanpa perlu menggunakan `print ()`.Conteh:

```
>>> 6 + 7</i
13
>>> 2 ** 3</i
8
>>> 5 - 1</i
4
>>> 7 / 3</i
2
>>> 7 %3</i
1
>>> 7 / 2</i
3.5
>>>
```

Saat melakukan operasi aritmetika, Python mengonversi tipe data dengan ketentuan sebagai berikut.

- Jika terdapat operan yang berupa bilangan kompleks, yang lain akan dikonversi secara sementara ke bilangan kompleks.
- Jika terdapat operan yang berupa bilangan real, yang lain akan dikonversi secara sementara ke bilangan real.
- Kalau semua operan berupa bilangan bulat, konversi tipe data tidak dilakukan.

Oleh karena itu, dua operasi pembagian berikut memberikan tipe hasil yang berbeda:

```
8 // 2
8 // 2.0
```

Hal ini diperlihatkan pada contoh berikut:

```
>>> 8 // 2
4
>>> 8 //  2.0</i
4.0
>>>
```

Hasil yang pertama berupa 4 (bilangan bulat) dan hasil kedua berupa 4.0 (bilangan real). Untuk memastikan tipe data masing-masing, Anda bisa menuliskan perintah seperti berikut:

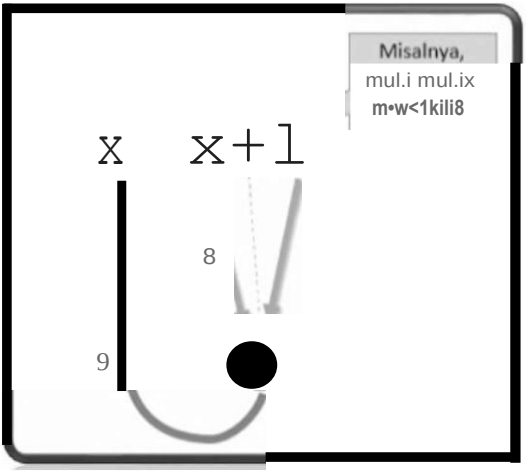
```
>>> type(S // 2)
<class 'int'>
>>> type(S // 2.0)
<class 'float'>
>>>
```

2.6.2 Ekspresi x = x + 1

Kadangkala, dijumpai ekspresi semacam berikut:

```
x = x + 1
```

Sekali lagi, operator = pada Python berfungsi sebagai operator penugasan, bukan pembandingan. Pada ekspresi seperti itu, nilai x dan 1 dijumlahkan terlebih dahulu. Kemudian, hasilnya ditugaskan ke variabel x. Sebagai contoh, mula-mula x mewakili nilai 8. Maka, x + 1 menghasilkan 9 (Gambar 2.6). Nilai 9 inilah yang kemudian ditugaskan ke x. Dengan demikian, kini x mewakili nilai 9.



Gambar f!.6 Ekspresi x = x + 1

Dengan kata lain,

```
x = x + 1
```

adalah perintah untuk menaikkan nilai yang diwakili x sekarang sebesar 1 terhadap nilai yang diwakili sebelumnya. Contoh:

```
>>> x = 8
>>> x = x + 1
>>> print(x)
9
>>> x = x + 1
>>> print(x)
10
>>>
```

2.6.3 Operator Penugasan Melekat

Untuk menyederhanakan penulisan semacam x x + 1, Python menyediakan ekspresi berupa:

```
x += 1
```

Tentu saja, tidak hanya + yang disediakan, melainkan juga yang lain-lain. Tabel 2.5 mencantumkan keseluruhan operator penugasan melekat.

Tabel 2.5 Daftar operator penugasan melekat

Operator	Bentuk Pemakaian	Contoh
**=	var_a **= n identik dengan: var_a = var_a ** n	x = 5 x **= 2 Kini x bernilai 25 (5 ** 2)
*=	var_a *= n identik dengan: var_a = var_a * n	x = 5 x *= 2 Kini x bernilai 10 (5 * 2)
/=	var_a /= n identik dengan: var_a = var_a / n	x = 5 x /= 2 Kini x bernilai 2.5 (5 / 2)
//=	var_a //= n	x = 5

Operator	Bentuk Pemakaian	Contoh
	identik dengan: var_a = var_a // n	x // = 2 Kini x bernilai 25 (5 // 2)
%=	var_a %= n identik dengan: var_a = var_a % n	x = 5 x %= 2 Kini x bernilai 1 (5 % 2)
+=	var_a += n identik dengan: var_a = var_a + n	x = 5 x += 2 Kini x bernilai 7 (5 + 2)
-=	var_a -= n identik dengan: var_a = var_a - n	x = 5 x -= 2 Kini x bernilai 3 (5 - 2)

2.6.4 Ekspresi pada String

Salah satu ekspresi pada string adalah untuk mendapatkan satu karakter yang berada dalam string. Ekspresinya berupa:

variabe/String[indeks]

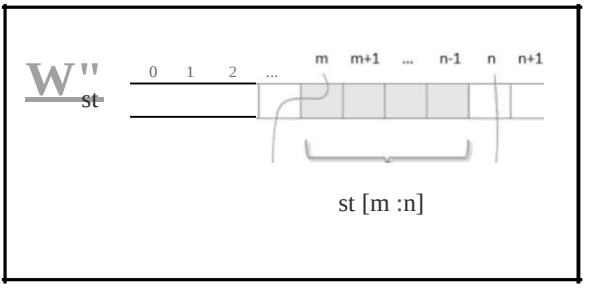
Dalam hal ini, *indeks* dimulai dari O yang menyatakan posisi karakter pertama. Contoh diperlihatkan berikut ini:

```
>>> buah = "Matoa"<!J
>>> print(buah[O])  <JJ
M
>>> print(buah[l])  <JJ
a
>>> print(buah[4])  <JJ
a
>>>
```

Indeks yang digunakan pada [] dapat berbentuk seperti berikut:

variabe/String[m:n]

Dalam hal ini, hasilnya adalah string dimulai dari indeks *m* hingga indeks *n-1*. Penjelasan visual dapat dilihat pada Gambar 2.7.



Gambar f!. 7 Ekspresi variabelString[m:n]

Conteh:

```
>>> buah = "Matoa"()J
>>> print(buah[0:1]) ()J
M
>>> print(buah[0:2]) 7H
Ma
>>> print(buah[0:3]) ()J
Mat
>>> print(buah[l:4]) ()J
ato
>>>
```

Operator+ dan * juga dapat dikenakan pada string. Operator+ berguna untuk melakukan operasi konkatenasi atau penggabungan string, sedangkan operator * berguna untuk membentuk string yang merupakan perulangan dari suatu string beberapa kali. Conteh:

```
>>> print("AB" +"  CD")0
AB CD
>>> print("12" + "34") 0
1234
>>>print("+-"* 5) ()J
+-+-+-+-+-
>>>
```

Operator in tersedia untuk, dengan bentuk pemakaian seperti berikut:

substring in *string*

Ekspresi ini menghasilkan True sekiranya substring berada di string atau False untuk keadaan sebaliknya. Contoh:

```
>>> print("ab" in "Krabi") (/)
True
>>> print("AB" in "Krabi") (/)
False
>>>
```

Hasil yang didapat menyatakan bahwa "ab" terdapat pada "Krabi", tetapi "AB" tidak terdapat pada "Krabi".

Operator not in digunakan pada string dengan bentuk seperti berikut:

substring not in *string*

Ekspresi ini menghasilkan True sekiranya *substring* tidak berada di *string* atau False kalau *substring* justru terdapat pada *string.* Contoh:

```
>>> print("ab" not in "Krabi")
False
>>> print("AB" not in "Krabi") (/)
True
>>>
```

Operator % digunakan sebagai tempat untuk memformat data. Penggunaannya, pertama-tama % diikuti dengan huruf pemformat. Kedua, % digunakan tepat sebelum data yang diformat. Format untuk % dapat dilihat pada Tabel 2.6.

Tabel 2.6 Kode pemformat data untuk membentuk string

Format	Keterangan
%u	Bilangan bulat tak bertanda
%0	Bilangan oktal
%x	Bilangan heksadesimal dengan huruf kecil

Format	Keterangan
%X	Bilangan heksadesimal dengan huruf kapital
%e	Format eksponensial (notasi sain) dengan huruf e kecil
%E	Format eksponensial (notasi sains) dengan huruf E kapital
%f	Bilangan real
%g	Bisa berlaku seperti %e atau %f tergantung nilai data yang diformat
%G	Bisa berlaku seperti %E atau %f tergantung nilai data yang diformat

Conteh berikut menunjukkan penggunaan %d yang diterapkan pada bilangan bulat:

```
>>> harga = 123400
>>> st = "Barga %d per unit" % ha.rga
>>> print(st)
Harga 123400 per unit
>>>
```

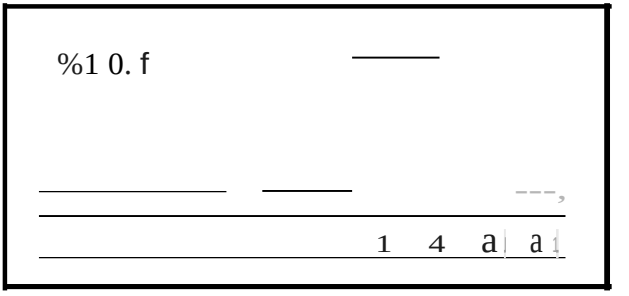
Perhatikan bahwa %ct merupakan wadah untuk data harga. Dalam hal ini, %ct digunakan mengingat harga berisi data bilangan bulat. Selain itu, pemformatan data dilakukan dengan pola seperti berikut:

stringPemformat%*dataYangDiformat*

Conteh berikut menunjukkan penggunaan %f yang diterapkan pada bilangan real:

```
>>> ni1aiPhi = 3.14
>>> print("Phi =%.4f" % ni1aiPhi)
Phi= 3.1400
>>>
>>> print("Phi = %10.4 f¹¹ % ni1aiPhi)
Phi       3.1400
>>>
```

Pada contoh ini, "%.4f" menyatakan bahwa bilangan real diatur agar mengandung empat digit pada bagian pecahannya. Adapun "%10.4f" menyatakan bahwa bilangan real ditampilkan dalam ruang selebar 10 karakter dengan digit pecahan sebanyak 4. Gambar 2.8 memperjelas format ini.



Gambar f!.8 Pengaturan dengan %10.41"

Conteh berikut menunjukkan penggunaan %f yang diterapkan pada data string dan bilangan bulat:

```
>>> namaBarang = "Spidel"
>>> harga = 4000
>>> print("Nama: %s, Harg·a: %d" % (namaBarang, harga))
Nama: Spidol, Harga: 4000
>>>
```

Hal yang perlu diperhatikan di sini adalah penggunaan tanda () yang dipakai untuk menyebutkan data yang hendak diformat, mengingat jumlah data lebih dari satu. Dalam hal ini, tanda koma diperlukan untuk memisah antardata.

Conteh berikut menunjukkan penggunaan tanda minus pada pemformatan string:

```
>>> print("|%20s|" %"Spidol")
|              Spidol|
>>> print("|%-20s|" %"Spidol")
|Spidol              |
>>>
```

Conteh ini menunjukkan bahwa jika tanda minus tidak digunakan pada pemformatan string yang mengatur lebar ruang untuk string, string akan diatur rata kanan. Sebaliknya, penambahan tanda minus membuat string diatur rata kiri.

Berikut menunjukkan cara menyajikan data alam bentuk tabel:

Berkas : daftarbarang.py

```
namaBarang1 = "TV Sakura 21\""
hargaBarang1 = 1200000

namaBarang2 = "TV Poligon 32\""
hargaBarang2 = 256000  0

namaBarang3 = "Kamera  Karunia TX32"
hargaBarang3 = 750000

NKALI = 47

print("-"* NKALI)
print("I %-30s I %10s  I" % ("Nama Barang", "Harga"))
print("-"* NKALI)
print("I %-30s I %10d  I" % (namaBarang1, hargaBarang1))
print("I %-30s I %10d  I" % (namaBarang2, hargaBarang2))
print("I %-30s I %10d  I" % (namaBarang3, hargaBarang3))
print("-"* NKALI)
```

Akhir berkas

Berikut adalah hasil pemanggilan skrip ini:

```
C:\LatOpenCV>python databarang.py

| Nama Barang                          Harga |

  TV Sakura 21"                      1200000 |
| TV Poligon 32"                     2560000 |
| Kamera Karunia TX32                 750000 |


C:\LatOpenCV>
```

2.6.5 Konversi Data

Untuk mendukung konversi data secara eksplisit, Python menyediakan sejumlah fungsi. Beberapa fungsi dasar untuk keperluan ini dicantumkan pada Tabel 2.7.

label 2.7 Fungsi-fungsi untuk melakukan konversi data

Fungs	Keterangan
int(x, b)	Fungsi ini digunakan mengonversi string x yang mengandung bilangan pada sistem berbasis *b* ke bilangan bulat
int(x)	Fungsi ini mengonversi x ke bilangan bulat
float(x)	Fungsi ini mengonversi x ke bilangan real
ord(x)	Fungsi ini mengonversi karakter x ke bilangan bulat yang menyatakan nilai ASCII argumen x
chr(x)	Fungsi ini menghasilkan karakter yang nilai ASCll-n ya berupa x
hex(x)	Fungsi **ini** mengonversi bilangan bulat x ke string heksadesimal
oct(x)	Fungsi ini mengonversi bilangan bulat x ke string oktal
bin(x)	Fungsi ini mengonversi bilangan bulat x ke string biner
str(x)	Fungsi ini mengonversi bilangan x ke string
complex(x, y)	Fungsi ini mengonversi bilangan yang dinyatakan oleh x dan y ke bilangan kompleks: x + yj

Conteh penggunaan fungsi-fungsi yang tertera pada Tabel 2.7 dapat dilihat berikut ini:

```
>>> int("AB", 16) {}
171
>>> int ("17", 8) {}
15
>>> int ("1101",2) {}
13
>>> int (76. 89) {}
76
>>> hex(171) {}
'Oxab'
>>> oct (15) {}
'0o17'
>>> bin (13) {}
'0b1101 '
```

```
>>> chr (65) {}
'A'
>>> ord ( "A" ) {}
65
>>> f l o a t ( "1 2 3 . 5 ") {}
123.5
>>> str(123.5) {}
'123.5'
>>> complex(2, 3) {}
(2+3j)
>>>
```

2.6.6 Prioritas Operator

Prioritas operator perlu dipahami dengan baik terutama kalau ekspresi yang perlu ditulis melibatkan beberapa operator. Sebagai contoh, terdapat ekspresi seperti berikut:

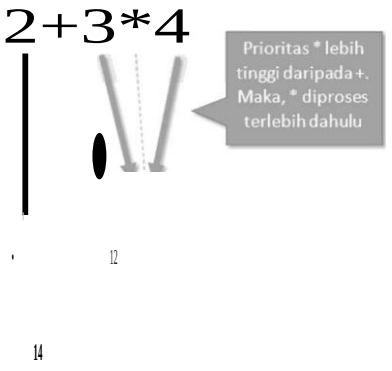
```
2+3*4
```

Berapa hasilnya? Tanpa memahami prioritas operator + dan * hasil ekspresi tersebut tidak dapat dijawab dengan tepat.

Sekarang, Anda bisa melihat Tabel 2.8. Berdasarkan tabel tersebut, terlihat bahwa prioritas * lebih tinggi daripada +. Dengan demikian, 3 * 4 akan diproses terlebih dahulu. Kemudian, hasilnya dijumlahkan dengan 2. Dengan demikian, hasilnya adalah 14 (Gambar 2.9).

label 2.8 Prioritas operator

Prioritas	Operator
1	() Tuple [] List {} Dictionary ' obj ek' (konversi string)
2	objek[i] Indeks objek[i :j] Irisan x.atribut Referensi atribut fungsi() Pemanggilan fungsi
3	**
4	+x (Positif) -x (Negatif) ~x (Negasi bit)
5	*,/,//,%
6	+, - (sebagai penjumlahan dan pengurangan)
7	<<,>>
8	&
9	A
10	|
11	<, <=, >, >=, !=, <>, is, is not, in, not in
12	Not
13	And
14	Or
15	Ekspresi berkondisi
16	Lambda



*Gambar f!.9 Pengeijaan pada ekspresi f! + s * 4*

Kadangkala, urutan pengerjaan dalam suatu ekspresi perlu kita atur sendiri. Hal ini dapat dilakukan dengan menggunakan tanda kurung. Conteh:

(2 + 3) * 4

memberikan hasil 20. Pada contoh ini, tanda kurung menyebabkan 2 + 3 dikerjakan terlebih dulu. Setelah itu, baru mengerjakan pengalian hasil 2 + 3 dengan 4.

2.6.7 Operator Pemhanding

Operator pembanding atau operator relasional adalah operator yang berfungsi untuk melakukan pembandingan. Hasilnya berupa nilai benar atau salah. Daftar operator pembanding dicantumkan pada Tabel 2.9.

Tabel 2.9 Daftar operator pembanding

Operator	Keterangan	Contoh
>	Lebih dari	4 > 6 False
<	Kurang dari	4 < 6 True
=	Sama dengan	4 = 6 False
!=	Tidak sama dengan	4 != 6 True

Operator	Keterangan	Contoh
>=	Lebih dari atau sama den gan	4 >= 6 False 6 >= 6 True
<=	Kurang dari atau sama dengan	8 <= 6 False 4 <= 6 True

Pembandingan tidak hanya untuk bilangan, tetapi juga untuk string. Conteh:

```
>>> "A" > "B"
False
>>> "C" > "A"
True
>>> "Yogyakarta" > "YOGYAKARTA"
True
>>> "Yogyakarta" == " YOGYAKARTA" ¢
False
>>>
```

Mungkin menjadi pertanyaan, "Mengapa ekspresi Yogyakarta > YOGYAKARTA menghasilkan nilai benar?" Selusi atas pertanyaan ini mudah dipahami sekiranya Anda mengenal tabel ASCII *(American Standard Code for Information Interchange).* ASCII merupakan standar yang digunakan untuk menyatakan karakter dalam bentuk nilai bilangan. Tabel 2.10 memperlihatkan sebagian nilai ASCII untuk sejumlah karakter.

Tabel .10 ***Nilai ASCII*** *sejumlah karakter*

Karakter	Nilai ASCI	Karakter	Nilai ASCI
A	65	A	97
B	66	B	98
C	67	C	99
D	68	D	100
Z	90	Z	122

Karakter	Nilai ASCI	Karakter	Nilai ASCI
0	48	@	64
1	49	*	42
9	57	+	43

Berdasarkan informasi pada Tabel 2.10, terlihat bahwa nilai ASCII huruf a (huruf kecil) lebih besar daripada huruf A (huruf kapital). Dengan demikian, ekspresi "a" > "A" menghasilkan True. Itulah sebabnya, "Yogyakarta" > "YOGYAKARTA" menghasilkan True.

2.6.8 Operator Logika

Operator logika atau juga disebut operator *Boolean* biasa digunakan untuk membentuk suatu keadaan logika (benar atau salah) berdasarkan satu atau dua operan yang bernilai True atau False. Tabel 2.11 mencantumkan tiga operator logika pada Python.

Tabel 2.11 Daftar operator logika

Operator	Keterangan	Bentuk Pemakaian
And	Operasi "dan "	*operanl* and *operan2*
Or	Operasi " at au "	*operanl* or *operan2*
Not	Operasi " bukan "	not *operan*

Tabel 2.12 memperlihatkan tabel kebenaran operator logika and dan or. Pada and, hasil True hanya kalau kedua operan bernilai True. Pada or, hasil True kalau ada operan bernilai True.

Tabel 2.12 Tabel kebenaran and dan or

X	y	x and y	xory
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Contoh berikut menunjukkan pengujian nilai kar berupa huruf kapital atau tidak menggunakan operator and:

```
>>> kar = "m"
>>> kar >= "A" and kar <= "Z"
False
>>>
```

```
>>> kar = "M"
>>> kar >= "A" and kar <= "Z"
True
>>>
```

Tampak bahwa pengujian dengan kar >= "A" and kar <= "Z" untuk kar berupa "m" menghasilkan False, sedangkan hasilnya berupa True untuk kar bernilai **"M".**

Contoh berikut menunjukkan pengujian nilai pilihan memenuhi daftar pilihan atau tidak:

```
>>> pil.ihan = "Q"
>>> pil.ihan - "Q" or pil.ihan = "q"
True
>>> pil.ihan = "q"
>>> pil.ihan - "Q" or pil.ihan - "q"
True
>>> pil.ihan = "x"
>>> pil.ihan - "Q" or pil.ihan - "q"
False
>>>
```

Tampak bahwa pengujian dengan pilihan == "Q" and pilihan "q" menghasilkan True untuk pilihan bernilai Q maupun q. Namun, hasilnya berupa False untuk selain q dan Q.

Adapun operator not hanya melibatkan satu operan. Bentuk pemakaiannya seperti berikut:

notoperan

Hasilnya berupa kebalikan nilai operan. Jadi,

- True kalau *operan* bernilai False;
- False kalau *operan* bernilai True.

Hal ini ditunjukkan pada contoh berikut:

```
>>> not True
False
>>> not False
True
>>>
```

Nah, bagian setelah not dapat diisi dengan ekspresi apa saja yang menghasilkan nilai True atau False.

2.6.9 Operator Berhasis Bit

Untuk kepentingan operasi yang berbasis bit (O dan 1), Python menyediakan enam operator berikut:

- & (dan untuk biner),
- | (atau untuk biner),
- ^ (atau eksklusif),
- ~ (inversi untuk biner),
- « (geser kiri), dan
- » (geser kanan).

Operator & digunakan untuk melakukan operasi "dan" pada tataran bit. Tabel 2.13 memperlihatkan sifat operasi dengan operator ini.

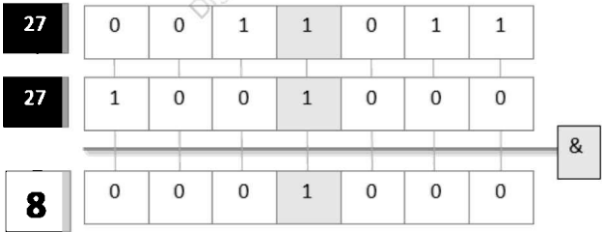
Tabel 2.13 Operasi dengan operator &

Bit	Bit2	Bitl & Bit2
0	0	0
0	1	0
1	0	0
1	1	1

Tampak bahwa bit hasil berupa 1 hanya kalau kedua bit yang dikenai operator & bernilai 1. Contoh:

```
>>> x = 27(/)
>>> y = 72(/)
>>> z= x & y(/}
>>> print(bin(x), bin(y), bin(z), z)  (/}
Ob11011 Ob1001000 Ob1OOO 8
>>>
```

Pada contoh ini, nilai x, y, dan z ditampikan dalam bentuk biner. Nah, hasil pada z adalah 8 dan bentuk binernya adalah 1000. Penjelasannya diperlihatkan pada Gambar 2.10.



***Gamhar 12.10 Operasi** !27 & 712*

Operator | digunakan untuk melakukan operasi "or" pada tataran bit. Tabel 2.14 memperlihatkan sifat operasi dengan operator ini.

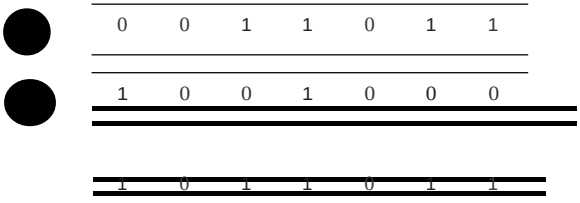
Tabel 2.14 Operasi dengan operator |

Bit	Bit2	Bit | Bit2
0	0	0
0	1	1
1	0	1
1	1	1

Tampak bahwa bit hasil berupa 1 kalau terdapat bit yang dikenai operator | bernilai 1. Conteh:

```
>>> x = 27(/)
>>> y = 72(/)
>>> z = x | y(/}
>>> print(bin(x), bin(y), bin(z), z)  (/}
Ob11011 Ob1001000 Ob1011011 91
>>>
```

Pada contoh ini, nilai x, y, dan z I dalam bentuk biner. Nah, hasil pada z adalah 91 dan bentuk binernya adalah 1011011. Penjelasannya diperlihatkan pada Gambar 2.11.



Gamhar f!.11 Operasi *27 / 71!*

Operator /1 digunakan untuk melakukan operasi "atau eksklusif" pada tataran bit. Tabel 2.15 memperlihatkan sifat operasi dengan operator ini.

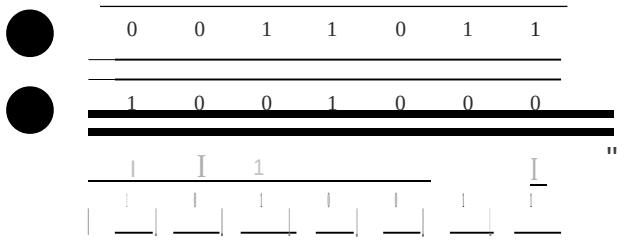
Tabel 2.15 Operasi dengan operator,..

Bit	Bit2	Bitl & Bit2
0	0	0
0	1	1
1	0	1
1	1	0

Tampak bahwa bit hasil berupa 1 hanya kalau hanya salah satu bit yang dikenai operator,.. bernilai 1. Contoh:

```
>>> x = 27(/)
>>> y = 72(/)
>>> z = x " y(/}
>>> print(bin(x), bin(y), bin(z), z) (/}
Ob11011 Ob1001000 Ob1010011 83
>>>
```

Pada contoh ini, nilai x, y, dan z ditampikan dalam bentuk biner. Nah, hasil pada z adalah 83 dan bentuk binernya adalah 1010011. Penjelasannya diperlihatkan pada Gambar 2.12.



***Gamhar** !2.1!2 **Operasi** !27 A 712*

Operator ~ dikenakan pada satu operan. Dalam hal ini, hasil yang diperoleh mempunyai kebalikan bit operan. Contoh:

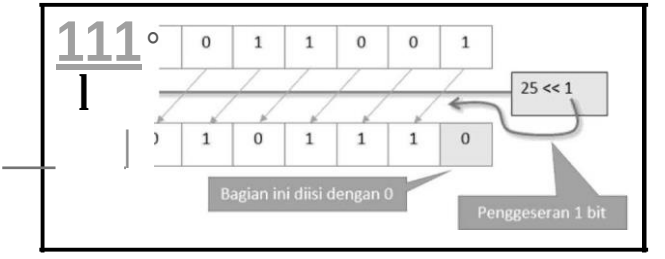
```
>>> x = 27(/)
>>> z = -x (/}
>>> print(bin(x), bin(z), z) (/}
Ob11011 -Ob11100 -28
>>>
```

Pada contoh ini, nilai x dan z ditampikan dalam bentuk biner. Nah, hasil pada z adalah -28. Nilai ini merupakan bentuk komplemen kedua nilai 27. Secara umum, untuk n positif, ~n berupa - (n+l).

Operator« (geser kiri) berguna untuk menggeser bit-bit ke kiri. Jumlah penggeseran ditentukan oleh operan yang berada di kanan operator ini. Contoh:

```
>>> x = 25(/)
>>> z = x<< 1(/)
>>> print(bin(x), bin(z), z) (/}
Ob11001 Ob110010 50
>>>
```

Pada contoh ini, nilai x dan z ditampikan dalam bentuk biner. Nah, hasil pada z adalah 50 dan bentuk binernya adalah 110010. Penjelasannya diperlihatkan pada Gambar 2.13.

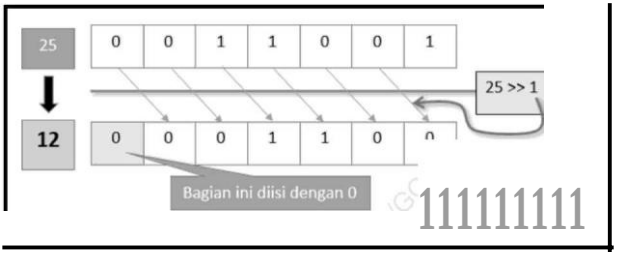


Gambar 2.1:J Operasi dengan <<

Operator » (geser kanan) berguna untuk menggeser bit-bit ke kanan. Jumlah penggeseran ditentukan oleh operan yang berada di kanan operator ini. Contoh:

```
>>> x = 25
>>> y = 72
>>> z = x >> 1
>>> print(bin(x), bin(z), z)
0b11001 0b1100 12
>>>
```

Pada contoh ini, nilai x dan z ditampilkan dalam bentuk biner. Nah, hasil pada z adalah 12 dan bentuk binernya adalah 1100. Penjelasannya diperlihatkan pada Gambar 2.14.



Gambar fJ,14 Operasi dengan >>

2.7 Penulisan Satu Perintah Ke Lebih dari Satu Baris

Satu perintah bisa ditulis dengan mencakup lebih satu baris. Hal seperti ini perlu dilakukan agar isi suatu baris tidak terlalu panjang sehingga mudah dibaca oleh orang hanya dengan satu kali pandang.

Kadangkala, tanda \ perlu diletakkan di akhir baris untuk menyatakan bahwa masih ada kelanjutan perintah pada baris ini di baris berikutnya. Sebagai contoh, perintah

```
1+2+3+4+5+6
```

dapat ditulis menjadi:

```
1 + 2 + 3+\
4+5+6
```

Contoh pengujiannya seperti berikut:

```
>>> 1 + 2 + 3 +\<P
      4 + 5 + 6<P
 21
>>>
```

Pada contoh ini, setelah tanda \ diberikan dan tombol Enter ditekan, Python akan menampilkan tanda "...", yang menyatakan bahwa interpreter Python menunggu kelanjutan perintah yang belum berakhir. Begitu 4 + 5 + 6 diberikan dan tombol Enter ditekan, Python mengetahui bahwa perintah telah berakhir dan perintah yang diberikan segera dieksekusi.

Tidak semua perintah yang ditulis lebih satu baris perlu menggunakan tanda \ untuk menyatakan bahwa perintah masih berlanjut. Ciri-ciri perintah yang masuk kategori ini adalah seperti berikut.

1. Terdapat ekspresi dalam tanda kurung, kurung siku, atau kurung kurawal. Pemenggalan pada bagian yang terletak di dalam tanda kurung tidak memerlukan tanda \ sebagai penanda bahwa perintah masih berlanjut. Sebagai contoh, pernyataan berikut

   ```
   print("Harga barang per unit", 2000)
   ```

   dapat ditulis menjadi:

   ```
   print("Harga barang per unit",
         2000)
   ```

2. Karakter pindah baris *(newline)* terdapat pada string yang ditulis dalam tiga tanda petik. Contoh dapat dilihat berikut ini.

```
>>> teks = ''' Nama-nama kota:
        Semarang
        Yogyakarta
        Medan
        Padang
        Ambon
        Merauke'''
>>> print(teks)
Nama-nama kota:
Semarang
Yogyakarta
Medan
Padang
Ambon
Merauke
>>>
```

Pada contoh ini, teks diisi dengan string yang mengandung sejumlah karakter *newline* atau karakter pindah baris ("\n") tetapi tidak ditulis secara eksplisit. Hal ini bisa dilakukan dengan menggunakan awalan dan akhiran ''' (tiga tanda petik-tunggal}. Pada keadaan seperti ini, setiap kali tombol Enter ditekan, interpreter Python mengetahui bahwa perintah belum berakhir dan akan menambahkan karakter pindah baris pada string. Kemunculan tanda ... menyatakan bahwa kelanjutan string masih ditunggu.

2.8 Penulisan Lebih dari Satu Perintah dalam Satu Baris

Dua perintah atau lebih bisa ditulis dalam satu baris. Dalam hal ini, tanda titik-koma (;) perlu digunakan sebagai pemisah antarpernyataan. Sebagai contoh, tiga pernyataan berikut

```
a = 1
b = 2
c = 3
```

bisa ditulis menjadi:

```
a= 1; b =2; c= 3
```

Conteh lain ditunjukkan berikut ini:

```
>>> st = "ABCD"; print (st)
ABCD
>>>
```

2.9 Pelibatan Modul Eksternal

Setiap skrip Python dianggap sebagai modul. Modul adalah kode yang disimpan dalam suatu berkas. Kode yang terdapat pada modul bisa digunakan dengan terlebih dulu "mengimpor" modul tersebut.

Cara mengimpor suatu modul dilaksanakan dengan menggunakan pernyataan import. Conteh:

```
>>> import math
>>> math.sqrt(25)
5.0
>>> math.exp(2)
7.38905609893065
>>>
```

Pada contoh di atas, modul yang diimpor yaitu math. Modul ini memungkinkan sejumlah metode yang berhubungan dengan operasi matematika bisa digunakan. Sebagai contoh, math. sqrt () berguna untuk mendapatkan akar kuadrat suatu bilangan dan ma th. exp () bermanfaat untuk menghitung ex.

2.10 Objek, Metode, dan Fungsi

Semua data di Python diperlakukan sebagai objek. Setiap objek memiliki tipe dan dapat mengandung properti dan metode. Setiap objek mempunyai kelas, yang menjadi bahan cetakan untuk objek. Conteh

berikut menunjukkan penugasan nilai ke variabel bilangan dan tipe data untuk variabel tersebut:

```
>>> bilangan = 5 + 7j
>>> type(bilangan)
<class 'complex'>
>>> bilangan
(5+7j)
>>>
```

Conteh ini menunjukkan bahwa bilangan adalah objek berkelas complex (bilangan kompleks) dan nilainya adalah 5+7j.

Terkadang objek memiliki properti (kadang disebut atribut atau *field).* Sebagai contoh, objek berkelas complex mempunyai properti data bernama real yang berisi bagian real bilangan kompleks dan imag yang berisi bagian imajiner bilangan kompleks. Conteh:

```
>>> bilangan = 5 + 7j
>>> bilangan.real
5.0
>>> bilangan.imag
7.0
>>>
```

Di Python, yang disebut fungsi adalah suatu nama yang mewakili sejumlah kode dan jika nama tersebut dipanggil akan diperoleh suatu nilai balik. Sebagai contoh, len () adalah fungsi yang tersedia pada Python yang berguna untuk menghitung jumlah karakter dalam string. Conteh penggunaan fungsi ini adalah seperti berikut:

```
>>> len("Python")
6
>>>
```

Pada contoh ini, len adalah nama fungsi, "Python" merupakan argumen fungsi, dan 6 adalah nilai balik fungsi (Gambar 2.15)

Gambar fi!.1.5 Fungsi, argumen, dan nilai balik

Suatu fungsi yang melekat pada suatu objek dinamakan metode. Bentuk pemanggilan metode adalah seperti berikut:

nama_objek.nama_metode()

nama_objek.nama_metode(argumen,...)

Bentuk pertama menyatakan pemanggilan metode yang tidak memiliki argumen, sedangkan yang kedua ditujukan untuk yang memiliki argumen. Sebagai contoh, string memiliki banyak metode untuk memanipulasi string. Salah satu metodenya berupa upper (), yang berguna untuk mengonversi setiap huruf kecil menjadi huruf kapital. Contoh penggunaannya seperti berikut:

```
>>> st = "Python"
>>> st.upper()
'PYTHON'
>>>
```

Contoh berikut menunjukkan penggunaan metode pada string yang melibatkan argumen:

```
>>> st = "Python"
>>> st.find("h")
3
>>>
```

Metode find() pada contoh ini digunakan untuk mendapatkan huruf "h" pada st. Nilai balik 3 menyatakan bahwa "h" terdapat pada indeks 3 di "Python". Sebagaimana diketahui, indeks pada string dimulai dari 0.

Catatan

Python menyediakan fungsi dir () yang berguna untuk mendapatkan nama-nama properti dan metode yang terdapat pada suatu objek. Sebagai contoh, perintah berikut akan menampilkan nama-nama properti dan metode pada string:

```
dir("a")
```

2.11 Pembacaan Data dari Papan Ketik

Pemasukan data dari papan ketik ditangani dengan menggunakan fungsi input (). Bentuk pemakaiannya:

```
input([prompt])
```

Argumen *prompt* (yang bersifat opsional) menyatakan keterangan yang ditampilkan tepat sebelum komentar menunggu pemakai memasukkan data dari papan ketik. Nilai balik fungsi berupa string yang dimasukkan pemakai. Contoh:

```
>>> nilai = input("Masukkan apa saja: ")  4)
Masukkan apa saja: tes .. tes .. 1234)
>>> print(nil.ai)  4)
tes .. tes ..... 123
>>>
```

Apabila dikehendaki data yang bukan berupa string, melainkan bilangan, hasil input () perlu dikonversi dengan menggunakan fungsi seperti int() atau float().