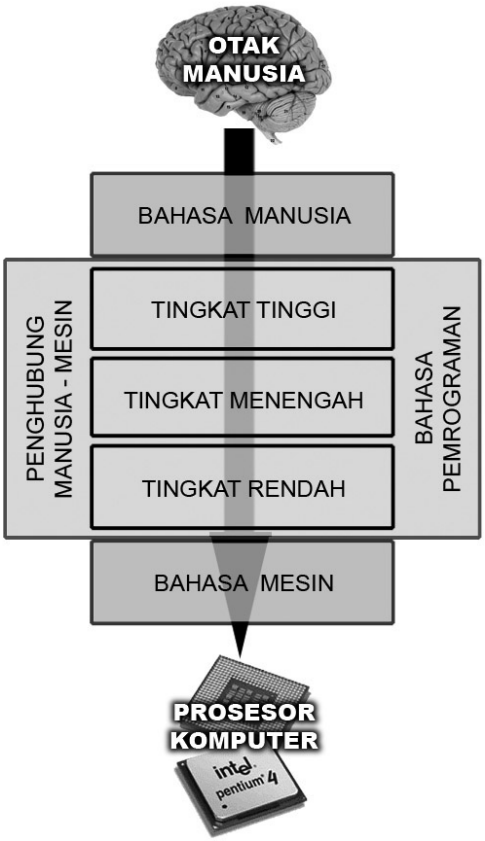
DASAR – DASAR PYTHON

2.1 Bahasa Pemrograman Secara Umum

Tingkat suatu bahasa pemrograman ditentukan oleh kedekatan bahasa tersebut dengan bahasa manusia pada umumnya, semakin dekat dengan bahasa manusia berarti tingkatnya semakin tinggi. Pada dasarnya, bahasa pemrograman dapat digolongkan menjadi 3 tingkat, yaitu tingkat tinggi, menengah, atau rendah. Sebuah bahasa pemrograman tingkat menengah dan tinggi pada umumnya menggunakan library (yang dapat berupa fungsi, sub-rutin, prosedur, atau class) yang sudah tersedia dan biasanya berupa kode dalam bahasa yang lebih rendah. User atau programer hanya perlu melakukan tindakan yang minimal untuk mendapatkan atau melakukan sesuatu yang tergolong kompleks dan rumit, sehingga kode program yang harus dibuat tidak terlihat rumit dan lebih mudah untuk melakukan *debug*. Dengan kata lain, dalam membangun program, kepingan-kepingan kode yang dipergunakan hanya sedikit, berarti memerlukan waktu yang lebih singkat dalam pembuatannya. Sedangkan jika menggunakan bahasa pemrograman tingkat rendah, programer harus membangun programnya dengan kepingan-kepingan kode yang lebih banyak dan kecil, untuk menghasilkan sebuah program dengan kegunaan dan hasil yang sama, tetapi dengan resiko melakukan kesalahan yang lebih besar.

Berikut ini adalah diagram yang menggambarkan tingkat suatu bahasa pemrograman dalam menghubungkan manusia dan mesin.



Gambar 2. Diagram tingkat bahasa pemrograman dalam menghubungkan manusia dan mesin

2.2 Berkenalan dengan Python

Python termasuk bahasa pemrograman terstruktur tingkat tinggi dan berorientasi objek. Itu berarti segala sesuatu di dalam Python adalah suatu objek, yang dapat memiliki elemen-elemen yang mendukung berjalannya fungsi objek tersebut.
Anda dapat mendownload manual Python di situs python.org.

2.3 Menggunakan Python

Sebagai permulaan, anda sebaiknya mencoba mengenal Python melalui konsol Python, dengan menjalankan python.exe.

Anda akan mendapatkan tampilan seperti berikut :

```
Python 2.4.2 (#67, Sep 28 2005, 12:41:11) [MSC v.1310 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Tanda >>> adalah tanda prompt Python, yang menunjukkan bahwa Python siap menerima perintah. Setiap baris perintah yang Anda ketik harus diakhiri dengan menekan Enter untuk menjalankannya. Jika Anda tidak menggunakan IDLE, Anda bisa menutup konsol Python dengan Ctrl-Z dan menekan Enter. Dalam bab ini, semua kode program langsung dijalankan pada prompt Python.

2.4 Tipe-tipe Objek Dasar dalam Python

Dalam pemrograman, hal yang paling mendasar yang sering dipergunakan adalah variabel. Variabel adalah pengingat atau tempat yang digunakan untuk menampung sebuah nilai yang — sesuai namanya — dapat berubah-ubah. Variabel dalam Python sedikit berbeda dengan variabel dalam kebanyakan bahasa pemrograman lain, yaitu pada proses definisinya. Dalam bahasa pemrograman lain, setiap variabel yang akan digunakan harus memiliki tipe yang jelas dan tidak dapat berubah selama program berjalan, dan harus didefinisikan secara eksplisit. Dalam Python, variabel tidak didefinisikan secara eksplisit, tetapi secara implisit dan otomatis, yaitu pada saat pertama kali variabel tersebut dipergunakan untuk menyimpan suatu nilai, dan tidak harus bertipe sama selama program berjalan. Karena Python adalah bahasa pemrograman berorientasi objek, variabel pun adalah suatu objek.

Setiap objek memiliki nama, yang hanya boleh tersusun dari :

- huruf : a, b, c, ..., z, A, B, C, ..., Z
- angka : 0, 1, 2, 3, ..., 9
  Angka tidak boleh mengawali nama sebuah objek.
- tanda garis bawah (*underscore*) : _

Contoh :

```
>>> var1 = 1                    (1)
>>> var2 = 1.5                  (2)
>>> var3 = 'abc'                (3)
>>> print var1, var2, var3      (4)
```

(1) kita membuat objek bernama `var1` yang kita beri nilai 1, yang secara otomatis membuat objek ini bertipe integer.

(2) kita membuat objek bernama `var2` yang kita beri nilai 1.5 (satu setengah), yang secara otomatis membuat objek ini bertipe float.

(3) kita membuat objek bernama `var3` yang kita beri nilai ‘abc’, yang secara otomatis membuat objek ini bertipe string.

(4) `print` digunakan untuk menampilkan output yang kita inginkan, yaitu isi `var1, var2, var3`.

hasil :

```
1 1.5 abc
```

Dalam contoh di atas, Anda telah mengenal beberapa tipe objek dalam Python. Berikut ini adalah penjelasan beserta contoh dari beberapa tipe objek dalam Python yang akan sering Anda pergunakan.

2.4.1 Tipe Bilangan

Tipe bilangan yang dibahas di sini hanya 2, yaitu integer dan float. Tipe-tipe bilangan yang lain sangat jarang dipergunakan.

2.4.1.1 Integer

Integer adalah bilangan bulat, yang mencakup bilangan negatif, nol, dan bilangan positif.

2.4.1.2 Float

Float adalah bilangan nyata/utuh, tidak dibulatkan, misalnya 1.5, 2.0, 453.87457

2.4.1.3 Operasi Tipe Bilangan

Antara 2 objek tipe bilangan dapat dilakukan beberapa operasi sebagai berikut :

Tabel 2. Tabel Operasi Bilangan

OPERASI	OPERATOR	SINTAKS	CONTOH
penjumlahan	+	A + B	1 + 1.5 = 2.5
pengurangan	-	A - B	2 + 5 = 7
perkalian	*	A * B	3 * 5.5 = 16.5
pembagian	/	A / B	10 / 4 = 2 10.0 / 4 = 2.5 ①
sisa hasil bagi	%	A % B	16 % 5 = 1
pangkat	**	A ** B	2 ** 3 = 8

① hal penting yang harus Anda ingat :

- integer / integer = integer
- float / integer = float
- integer / float = float

Operasi Relatif

Jika Anda memiliki objek bilangan dan ingin melakukan operasi yang relatif terhadap nilai objek tersebut, Python memiliki sintaks yang lebih sederhana daripada

```
<objek1> = <objek1> <OPERATOR> <objek2>
```

yaitu :

```
<objek1> <OPERATOR> = <objek2>
```

Contoh :

```
>>> a =  2
>>> a **= 3
>>> a
8
```

Fungsi Bilangan Standar

Beberapa fungsi standar untuk bilangan yang sering dipergunakan :

Tabel 2. Tabel Fungsi Bilangan Standar

OPERASI	FUNGSI	SINTAKS	CONTOH
nilai absolut	`abs()`	`abs(A)`	`abs(-10) = 10`
konversi menjadi integer	`int()`	`int(A)`	`int(5.2) = 5` `int(5.8) = 5`
konversi menjadi float	`float()`	`Float(A)`	`float(5) = 5.0`
pembulatan	`round()`	`round(A)` `round(A,<digit>)`	`round(9.2) = 9.0` `round(9.8) = 10.0`
pangkat	`pow()`	`pow(A,B)`	`pow(2,3) = 8`

2.4.2 Tipe Boolean

Boolean adalah tipe objek yang dipergunakan sebagai nilai benar/salah. Hanya ada 2 objek bertipe boolean, yaitu `True` (benar) dan `False` (salah). Boolean dipergunakan dalam pengecekan kondisi.

2.4.3 Tipe Sekuen (rangkaian/urutan)

Tipe sekuen yang dibahas di sini hanya 3, yaitu string, list, dan tuple. Tipe-tipe sekuen yang lain sangat jarang dipergunakan.

2.4.3.1 String

String adalah untaian karakter yang mencakup semua karakter dalam ASCII (*American Standard Code for Information Interchange*). ASCII adalah kode informasi yang memiliki indeks 0 hingga 255. ASCII mencakup seluruh karakter normal yang dapat Anda lihat pada keyboard Anda, ditambah dengan karakter-karakter perluasan yang tidak terdapat pada keyboard Anda.

Jika anda belum tahu sama sekali tentang ASCII dan ingin mengetahui karakter apa saja yang tersedia, Anda dapat menggunakan fungsi `chr` dalam Python.

Kode :

```
>>> for c in range(256):
...     print c, chr(c)
... <tekan Enter>
```

Output kode di atas berupa nomor indeks (dalam format desimal) dan diikuti oleh karakter ASCII dengan indeks tersebut.
Atau Anda dapat melihat selengkapnya di www.lookuptables.com

String didefinisikan dengan diapit tanda kutip. Anda bisa menggunakan tanda kutip tunggal maupun ganda, asalkan konsisten.
Contoh :

```
>>> print 'ini adalah string'
ini adalah string
>>> print "ini juga string"
ini juga string
```

Jika Anda menggunakan kedua tanda kutip yang berbeda di awal dan akhir, Python akan memberikan peringatan bahwa telah terjadi kesalahan sintaks (syntax error) dan proses program akan berhenti.

Contoh :

```
>>> print 'pasti error"
  File "<stdin>", line 1
    print 'pasti error"
                       ^
SyntaxError: EOL while scanning single-quoted string
```

Namun Anda dapat menggunakan tipe tanda kutip yang berbeda di dalam string yang diapit oleh tipe tanda kutip yang lain.
Contoh :

```
>>> print "ini tipe 'string'"
ini tipe 'string'
```

Konversi Objek Lain Menjadi String
Anda dapat mengubah objek selain string menjadi string dengan fungsi `str()`.
Contoh :

```
>>> print type( str(123) )
<type 'str'>
```

Format String
Jika Anda perlu membuat suatu string yang merupakan penggabungan dari beberapa objek string maupun bilangan, seperti berikut :

```
>>> m = 20
>>> print 'kemajuan bulan ini : ' + str(m) + ' persen'
kemajuan bulan ini : 20 persen
```

format string Anda tidak terlihat jelas, dan mungkin akan terlihat rumit dalam penggabungan yang melibatkan lebih banyak objek.

Untuk membuat format string Anda terlihat lebih jelas, Python menyediakan operator % yang memungkinkan substitusi nilai ke dalam string.

Sintaks :

```
<format string> %<nilai>                                    (1)
<format string> %( <nilai>, <nilai>, <nilai>,... )          (2)
```

(1) dipergunakan jika nilai yang disubstitusikan hanya satu
(2) jika nilai yang disubstitusikan lebih dari satu, harus disimpan dalam sebuah tuple

Nilai yang diberikan akan menggantikan `%<operator tipe hasil konversi>` yang dipergunakan dalam `<format string>` secara berurutan. Dalam proses substitusi nilai ke dalam string, Anda juga dapat melakukan konversi objek nilai ke beberapa tipe objek lain.
Berikut ini adalah beberapa operator tipe hasil konversi yang sering dipergunakan :

Tabel 2. Tabel Operator Tipe Hasil Konversi Format String

Operator Tipe Hasil Konversi	Tipe Hasil Konversi
s	string
c	satu karakter
i	Integer
f	float
e	Float (eksponensial)
%	karakter %

Contoh :

```
>>> m = 20
>>> print 'kemajuan bulan ini : %s persen' %m                (1)
kemajuan bulan ini : 20 persen

>>> print 'X: %s,  Y: %i,  Z: %i' %(10.5, 15.75, 20.22)      (2)
X: 10.5,  Y: 15,  Z: 20
```

(1) nilai integer m langsung disubstitusikan sebagai string.
(2) ketiga objek float dalam tuple disubstitusikan secara berurutan :
- 10.5 dikonversi langsung menjadi string
- 15.75 dan 20.22 dikonversi menjadi integer terlebih dulu sebelum menjadi string

Beberapa pilihan tambahan dalam menentukan format akhir dari hasil konversi dapat Anda baca dalam manual Python.

2.4.3.2 List

List — sesuai namanya — adalah daftar yang dapat menampung segala jenis objek Python. List didefinisikan dengan tanda kurung siku [ ].

Contoh :

```
>>> list1 = []                                          (1)
>>> list2 = [ 0, 1, 2, 3, 'empat']                     (2)
>>> list3 = [ 'nol', 'satu', 'dua', 'tiga' ]           (3)
```

(1) hasilnya adalah list kosong
(2) hasilnya adalah list yang berisi 4 integer, yaitu 0, 1, 2, 3, dan 1 string : 'empat'
(3) hasilnya adalah list yang berisi 4 string, yaitu 'nol', 'satu', 'dua', 'tiga'

Mendapatkan Anggota List

Anda dapat memperoleh anggota sebuah list dengan memberikan nomor indeksnya yang diapit tanda kurung siku [ ]. Indeks terkecil adalah 0.

Contoh :

```
>>> print list3[2]
dua
```

Mengubah Isi List

Anda juga dapat mengubah isi list dengan memberikan nomor indeks yang Anda perlukan.

Contoh :

```
>>> list3[2] =  'bukan dua lagi'
>>> print list3[2]
bukan dua lagi
```

Menambah Isi List

Anda dapat menambah isi sebuah list dengan menggunakan fungsi `append` dan memberikan objek yang perlu ditambahkan sebagai parameternya.

Contoh :

```
>>> list4 = [0,1,2]
>>> list4.append( 500 )
>>> list4.append( 'tambahan' )
>>> print list4
[0, 1, 2, 500, 'tambahan']
```

Menggabungkan Beberapa List

Anda dapat menggabungkan beberapa list dengan menggunakan operator +.

Contoh :

```
>>> print [0,1,2] + [5,6,7]
[0, 1, 2, 5, 6, 7]
```

2.4.3.3 Tuple

Tuple hampir sama dengan list, perbedaannya hanyalah tuple adalah objek yang tidak dapat diubah. Tuple didefinisikan dengan tanda kurung biasa ().

Contoh :

```
>>> tuple1 = ()                                          (1)
>>> tuple2 = ( 0, 1, 2, 'tiga' )                         (2)
>>> tuple3 = ( 'nol', 'satu', 'dua', 'tiga' )            (3)
```

(1) hasilnya adalah tuple kosong
(2) hasilnya adalah tuple yang berisi 3 integer, yaitu 0, 1, 2, dan 1 string : 'tiga'
(3) hasilnya adalah tuple yang berisi 4 string, yaitu 'nol', 'satu', 'dua', 'tiga'

Tuple yang hanya berisi satu anggota pada saat didefinisikan harus diakhiri dengan tanda koma. Jika tidak, objek yang Anda buat bukanlah sebuah tuple.

Contoh :

```
>>> tuple4 = (100)
>>> print tuple4
100
>>> print type( tuple4 )    (1)
<type 'int'>
```

(1) `type` digunakan untuk mendapatkan tipe dari objek yang diberikan

Dapat Anda lihat, ternyata hasilnya bukanlah sebuah tuple, melainkan sebuah integer. Jika Anda mendefinisikan sebuah tuple dengan satu anggota, seharusnya Anda menambahkan tanda koma :

Contoh :

```
>>> tuple4 = (100,)
>>> print tuple4
(100,)
>>> print type( tuple4 )
<type 'tuple'>
```

Mendapatkan Anggota Tuple

Anda dapat memperoleh anggota sebuah tuple dengan memberikan nomor indeksnya yang diapit tanda kurung siku [ ]. Indeks terkecil adalah 0.

Contoh :

```
>>> print tuple3[1]
satu
```

Mengubah Isi Tuple

Tuple tidak dapat diubah, dan jika Anda berusaha mengubahnya, Python akan memberikan peringatan bahwa Anda telah berusaha memberikan nilai terhadap item dari suatu objek yang tidak dapat berubah, dan proses program akan berhenti.

Contoh :

```
>>> tuple3[2] = 'bukan dua lagi'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
```

Tuple didesain tidak dapat berubah, yang menjadikan proses sebuah tuple lebih cepat daripada sebuah list. Jika data yang perlu Anda simpan tidak akan berubah selama program berjalan, lebih baik Anda menggunakan tuple daripada list.

Menggabungkan Beberapa Tuple

Anda tidak dapat mengganti isi sebuah tuple, tetapi Anda bisa menambah isinya dengan menggabungkannya dengan tuple lain, menggunakan operator +.

Contoh :

```
>>> print (0,1,2) + (500,600)
(0, 1, 2, 500, 600)
```

2.4.3.4 Dictionary

Dictionary adalah daftar untuk menampung pasangan data (kunci dan nilai). Kunci dan nilai dapat berupa segala tipe objek Python. Dictionary didefinisikan dengan tanda kurung kurawal {}. Antara kunci dan nilai dipisahkan dengan tanda titik dua.

Contoh :

```
>>> dict1 = {}                                                    (1)
>>> jam = {'senin':'07.00', 'selasa':'08.40', 'rabu':'10.30'}     (2)
```

(1) hasilnya berupa dictionary kosong

(2) hasilnya berupa dictionary yang berisi 3 pasang hari dan jam, 'senin', 'selasa', 'rabu' adalah kunci, yang diikuti oleh nilainya masing-masing.

Mendapatkan Isi Dictionary

Hampir sama seperti list dan tuple, Anda dapat memperoleh anggota sebuah dictionary, tetapi dengan memberikan sebuah kunci yang diapit tanda kurung siku [ ].

Contoh :

```
>>> print jam['selasa']
08.40
```

Mengubah Isi Dictionary

Anda dapat mengubah isi dictionary dengan memberikan kunci yang Anda perlukan.
Contoh :

```
>>> jam['selasa'] = '13.30'
>>> print jam['selasa']
13.30
```

Menambah Isi Dictionary

Anda dapat menambah isi dictionary dengan memberikan kunci yang Anda perlukan.
Contoh :

```
>>> jam['minggu'] = 'libur'
>>> print jam['minggu']
libur
```

2.5 Fungsi

Fungsi juga merupakan suatu objek dalam Python, namun memiliki keistimewaan, yaitu dapat dipanggil/dijalankan. Fungsi merupakan kode-kode yang dikumpulkan menjadi satu dan dikemas dalam suatu bentuk yang sewaktu-waktu dapat dijalankan secara berulang-ulang tanpa harus mendefinisikan ulang setiap kode yang diperlukan. Dalam bahasa pemrograman lain, fungsi sering disebut juga prosedur atau sub-rutin.
Beberapa hal yang dapat dilakukan dengan fungsi :

- fungsi dapat menerima parameter/argumen yang dapat diproses dalam fungsi
- fungsi dapat mengembalikan suatu objek sebagai hasil kerjanya

Fungsi tanpa parameter

Sintaks :

```
def <nama fungsi>():
    <proses>
```

Contoh :

```
>>> def fungsi1():                    (1)
...     print 'aku adalah fungsi'     (2)
... <tekan Enter>                     (3)
```

(1) fungsi didefinisikan dengan awalan `def`, yang berarti *define*, diikuti oleh nama fungsi, yang diikuti dengan parameter/argumen (yang diberikan pada fungsi ini) yang diapit tanda kurung biasa, dan diakhiri dengan tanda titik dua. Dalam fungsi ini, kita tidak menggunakan parameter apapun, sehingga tanda kurung fungsi ini kosong, tetapi tetap harus ada.
(2) kode program dalam suatu fungsi harus merupakan blok kode. Blok kode dalam Python tidak perlu dimulai dan diakhiri dengan awalan dan akhiran apapun seperti dalam bahasa pemrograman lain, tetapi cukup menggunakan indentasi (menjorok ke kanan).

Anda dapat mengindent dengan menggunakan Tab atau space. Dalam fungsi ini, kita hanya menampilkan tulisan 'aku adalah fungsi'.
③ tanda ... hanya akan Anda jumpai jika Anda menggunakan konsol / command line, yang menandakan bahwa anda sedang dalam blok kode. Untuk menandai bahwa Anda sudah selesai bekerja dalam suatu blok kode, tekan Enter.

Setelah Anda membuat fungsi di atas, Anda dapat menggunakannya kapan saja dengan memanggilnya seperti berikut :
Contoh :

```
>>> fungsi1()                          ①
aku adalah fungsi
>>> fungsi1                            ②
<function fungsi1 at 0x00DA9EF0>
```

① untuk menjalankan fungsi, Anda hanya perlu mengetik nama fungsi dan diikuti dengan parameter yang diapit tanda kurung biasa. Tanda kurung dipergunakan untuk menjalankan fungsi tersebut. Hasilnya adalah fungsi ini menampilan string yang telah kita definisikan tadi.
② jika Anda lupa memberikan tanda kurung, hasilnya adalah objek fungsi itu sendiri. `0x00DA9EF0` adalah posisi fungsi ini dalam memori komputer.

Fungsi dengan parameter
Sintaks :

```
def <nama fungsi>( <parameter> ):
    <proses>
```

Contoh :

```
>>> def apakahAku( aku ):              ①
...     print 'nilai :', aku           ②
...     print ' tipe :', type(aku)     ③
... <tekan Enter>
```

Sekarang cobalah memanggil fungsi ini :
Contoh :

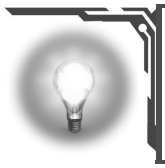
```
>>> apakahAku( 'tulisan' )                                    ①
nilai : tulisan
 tipe : <type 'str'>                                         ②

>>> apakahAku( 12345 )                                        ③
nilai : 12345
 tipe : <type 'int'>                                         ④

>>> apakahAku()                                               ⑤
Traceback (most recent call last):
```

```
  File "<pyshell#12>", line 1, in -toplevel-
    apakahAku()
TypeError: apakahAku() takes exactly 1 argument (0 given)    6
```

1 kita panggil fungsi ini dengan memberikan parameter berupa string 'tulisan'.
2 hasilnya adalah tampilan nilai dari objek string ini dan tipenya yang menunjukkan tipe string.
3 kita panggil fungsi ini dengan memberikan parameter berupa integer 12345.
4 hasilnya adalah tampilan nilai dari objek integer ini dan tipenya yang menunjukkan tipe integer.
5 kita panggil fungsi ini tanpa parameter apapun.
6 hasilnya adalah error :
`apakahAku() takes exactly 1 argument (0 given)` yang berarti fungsi ini memerlukan 1 parameter/argumen, tetapi tidak mendapatkannya.



Untuk memastikan fungsi Anda tidak mengalami hal seperti ini, Anda dapat menggunakan parameter bernilai baku, yang akan dipergunakan jika parameter tersebut tidak diberikan pada fungsi.

Fungsi dengan parameter bernilai baku

Parameter baku adalah parameter yang nilai bakunya sudah ditentukan pada definisi fungsi. Jika pada saat fungsi dipergunakan dan parameter tersebut tidak didapatkan, maka nilai bakunya yang akan dipergunakan.
Menggunakan parameter baku akan mempermudah Anda dalam situasi berikut :

- fungsi Anda memerlukan banyak parameter, tetapi tidak semua parameter diperlukan pada satu waktu tertentu

Sintaks :

```
def <nama fungsi>( <parameter>=<nilai baku> ):
    <proses>
```

Contoh :

```
>>> def data( nama, alamat='-', noTelp='-' ):      1
...     print 'Nama    :', nama
...     print 'Alamat  :', alamat
...     print 'No.Telp :', noTelp
... <tekan Enter>
```

1 kita buat fungsi dengan 1 parameter mutlak, dan 2 parameter bernilai baku yang nilainya ditentukan pada saat definisi fungsi.

Sekarang cobalah memanggil fungsi ini :
Contoh :

```
>>> data(nama='Andi', alamat='Jl. Lurus', noTelp='01234')      1
```

```
Nama    : Andi
Alamat  : Jl. Lurus
No.Telp : 01234

>>> data(nama='Bayu', alamat='Jl. Turun')                    (2)
Nama    : Bayu
Alamat  : Jl. Turun
No.Telp : -

>>> data(nama='Dani')                                        (3)
Nama    : Dani
Alamat  : -
No.Telp : -

>>> data(noTelp='010203', nama='Egi', alamat='Jl. Kembar')   (4)
Nama    : Egi
Alamat  : Jl. Kembar
No.Telp : 010203

>>> data(noTelp='000', nama='Error', 'Jl. Salah')            (5)
SyntaxError: non-keyword arg after keyword arg

>>> data('02468', 'Error', 'Jl. Salah')                      (6)
Nama    : 02468
Alamat  : Error
No.Telp : Jl. Salah
```

(1) ketiga parameter diberikan secara berurutan
(2) hanya 2 parameter nama dan alamat saja yang diberikan
(3) hanya parameter nama saja yang diberikan
(4) ketiga parameter dapat diberikan secara tidak berurutan dan tidak akan mengubah hasilnya, asalkan Anda menggunakan kata kunci (*keyword*) untuk tiap parameter.
(5) kata kunci (*keyword*) alamat tidak diberikan, hanya nilainya yang diberikan langsung. Ini tidak diperbolehkan dalam fungsi Python. Jika Anda telah menggunakan kata kunci untuk satu parameter, maka parameter sesudahnya harus mengunakan kata kunci juga.
(6) Jika Anda tidak ingin menggunakan kata kunci, Anda harus memberikan parameter-parameter dalam urutan yang sama seperti pada definisi fungsi, atau nilainya akan tertukar.

Fungsi yang Mengembalikan Objek

Fungsi dapat juga mengembalikan objek sebagai hasil dari proses yang dilakukannya, sehingga hasil proses dapat digunakan lebih lanjut oleh kode yang memanggil fungsi tersebut. Objek yang dikembalikan akan menggantikan fungsi yang dipanggil.

Contoh :

```
>>> def jadi3( obj ):
...     return obj * 3      (1)
... <tekan Enter>
```

(1) `return` berfungsi untuk menghentikan eksekusi kode dalam fungsi dan mengembalikan objek yang diperlukan. Fungsi ini mengembalikan tiga kali lipat dari objek yang diberikan padanya.

Sekarang cobalah memanggil fungsi ini :
Contoh :

```
>>> jadi3( 5 )                          (1)
15

>>> jadi3( 1.5 )                        (2)
4.5

>>> print 'ini ' + jadi3('NagaBonar ')  (3)
ini NagaBonar NagaBonar NagaBonar

>>> jadi3( [0,1,2] )                    (4)
[0, 1, 2, 0, 1, 2, 0, 1, 2]
```

(1) kita berikan integer 5, hasilnya adalah 5 kali 3, yaitu 15
(2) kita berikan float 1.5, hasilnya adalah 1.5 kali 3, yaitu 4.5
(3) kita berikan string ' NagaBonar ', hasilnya adalah penggabungan 3 kali string itu.
(4) kita berikan list [0,1,2], hasilnya adalah penggabungan 3 kali list itu.

Transfer parameter antar fungsi

Anda dapat melakukan transfer parameter dari satu fungsi ke fungsi yang lain, tanpa harus memberikan parameter tersebut satu per satu.
Sintaks :

```
def <nama fungsi>( *<tuple>, **<dictionary> ):
    <nama fungsi lain>( *<tuple>, **<dictionary> )
```

`<tuple>` : tuple yang digunakan Python untuk menyimpan semua parameter yang diberikan pada fungsi.

`<dictionary>` : dictionary yang digunakan Python untuk menyimpan semua parameter bernilai baku yang diberikan pada fungsi.

Pada saat melakukan transfer, Anda harus memberikan keduanya pada fungsi lain yang Anda perlukan, karena semua parameter diperlukan oleh fungsi tersebut.

Contoh :

```
>>> def fu(a,b,c=None):                                    ①
...     print a,b,c
... <tekan Enter>

>>> def fu2(*prm, **kw):
...     fu(*prm, **kw)                                      ②
...     print 'telah ditransfer semua :, prm,kw
... <tekan Enter>
>>> fu2(0,1,c=2)
0 1 2
telah ditransfer semua : (0, 1) {'c': 2}

>>> def fu3(*prm, **kw):
...     fu(*prm)                                            ③
...     print 'telah ditransfer tanpa keyword :', kw
... <tekan Enter>
>>> fu3(111,222,c=333)                                     ④
111 222 None
telah ditransfer tanpa keyword : {'c': 333}
```

① parameter c dalam fungsi ini bernilai baku None.
② semua parameter ditransfer ke fungsi `fu`.
③ hanya parameter tanpa keyword yang ditransfer ke fungsi `fu`.
④ karena parameter c diberikan dengan menggunakan keyword — yang nantinya akan disimpan dalam `kw` dan tidak diberikan ke fungsi `fu` — maka nilai baku None yang digunakan oleh fungsi `fu`.

2.6 Pengecekan Kondisi

Ini adalah bagian penting dari pemrograman, yang memungkinkan adanya percabangan alur program, tergantung hasil perbandingan antara kondisi yang diinginkan dengan kondisi yang sedang terjadi. Perbandingan kedua kondisi tersebut akan menghasilkan suatu nilai benar (True) atau nilai salah (False), yang bertipe boolean.

2.6.1 Nilai Salah (False)

Semua objek Python dapat dikonversi menjadi suatu tipe boolean untuk dipergunakan dalam perbandingan kondisi.
Berikut ini adalah objek yang dapat menghasilkan nilai salah (False) :

- False
- None
- nilai nol (numerik), contoh : 0 (tipe : integer), 0.0 (tipe : float)

- sekuen kosong, contoh : '' (tipe : string), [ ] (tipe : list), ( ) (tipe : tuple)
- dictionary kosong : {}

Selebihnya akan menghasilkan nilai benar (True).
Konversi objek menjadi tipe boolean dapat dilakukan secara otomatis oleh Python, tetapi Anda juga dapat melakukannya sendiri dengan menggunakan fungsi `bool()`.
Sintaks :

```
bool(<objek>)
```

Contoh :

```
>>> print bool(None)
False
>>> print bool(-1), bool(0), bool(1)
True False True
>>> print bool(''), bool('abc')
False True
>>> print bool(()), bool((1,))
False True
>>> print bool([]), bool([1])
False True
>>> print bool({}), bool({123:'abc'})
False True
```

2.6.2 Operator *and* dan *or*

Operator *and* (dan) dan *or* (atau) dipergunakan untuk mendapatkan nilai gabungan antara 2 objek boolean.

Tabel 2. Tabel Hasil Gabungan Objek Boolean

OPERATOR	Operator AND		Operator OR	
NILAI	True	False	True	False
True	True	False	True	True
False	False	False	True	False

Contoh :

```
>>> A=[]          (1)
>>> B=1           (2)
>>> C=None        (3)

>>> bool( A and B )          (4)
False
>>> bool( A or B )           (5)
True
>>> bool( A and B and C)     (6)
False
```

```
>>> bool( A or B or C)          7
True
>>> bool( A and B or C)         8
False
>>> bool( A or B and C)         9
False
```

1. nilai booleannya adalah False
2. nilai booleannya adalah True
3. nilai booleannya adalah False
4. False and True = False
5. False or True = True

Proses penggabungan berjalan dari kiri ke kanan :

6. (False and True) = (False) and False = False
7. (False or True) = (True) or False = True
8. (False and True) = (False) or False = False
9. (False or True) = (True) and False = False

2.6.3 Operator Perbandingan

Operator perbandingan dipergunakan untuk memperbandingkan objek, dan hasilnya adalah nilai boolean True atau False.
Berikut ini adalah beberapa operator perbandingan beserta artinya :

Tabel 2. Tabel Operator Perbandingan

Operator	Arti
==	sama dengan
!=	tidak sama dengan
<	lebih kecil dari
<=	lebih kecil dari atau sama dengan
>	lebih besar dari
>=	lebih besar dari atau sama dengan

Perbandingan 2 objek

Sintaks :

```
<objek1> <OPERATOR> <objek2>
```

Contoh :

```
>>> a=2; b=2; c=4
>>> a==b
True
>>> a==c
False
```

```
>>> b==c
False
>>> a!=c
True
>>> a<b
False
>>> c>b
True
```

Perbandingan menerus

Sintaks :

```
<objek1> <OPERATOR1> <objek2> <OPERATOR2> <objek3>
```

artinya sama dengan :

```
<objek1> <OPERATOR1> <objek2> and <objek2> <OPERATOR2> <objek3>
```

Contoh :

```
>>> a=5; b=2; c=10
>>> a>b<c
True
>>> a<c>b
True
>>> a<b<c
False
>>> a*b<c>b          (1)
False
```

(1) artinya sama dengan 5*2<10 and 10>2 = 10<10 and 10>2 = False and ...

Proses perbandingan berjalan dari kiri ke kanan, jika salah satu perbandingan menghasilkan False, perbandingan selanjutnya tidak akan diproses.

2.6. Pengecekan Kondisi dengan *if <kondisi>*

Sintaks :

```
if <kondisi>:
   <proses1>
```

Artinya jika nilai boolean `<kondisi>` adalah True, maka `proses1` akan dilakukan.

Contoh :

```
>>> a=5; b=2; c=10
>>> if a>b<c:
      print 'ini dia'

ini dia
```

2.6. Pengecekan Kondisi dengan *if <kondisi> else*

Sintaks :

```
if <kondisi>:
   <proses1>
else:
   <proses2>
```

Artinya jika nilai boolean `<kondisi>` adalah :

- True : `<proses1>` akan dilakukan
- False : `<proses2>` akan dilakukan.

Contoh :

```
>>> a=5; b=2; c=10
>>> if a>b==c:
      print 'ini dia'
else:
      print 'gagal'

gagal
```

2.6. Pengecekan Kondisi dengan *if <kondisi1> elif <kondisi2> else*

Sintaks :

```
if <kondisi1>:
   <proses1>
elif <kondisi2>:
   <proses2>
elif <kondisi3>:
   <proses3>
else:
   <prosesLain>
```

Artinya jika nilai boolean `<kondisi1>` adalah :

- True : `<proses1>` akan dilakukan
- False : jika nilai boolean `<kondisi2>` adalah :
    - True : `<proses2>` akan dilakukan
    - False : jika nilai boolean `<kondisi3>` adalah :
        - True : `<proses3>` akan dilakukan
        - False : `<prosesLain>` akan dilakukan

Contoh :

```
>>> a=5; b=2; c=10
>>> if a<b:
      print 'benar1'
elif a<c:
```

```
      print 'benar2'
else:
      print 'salah1'

benar2
```

2.7 Range

Range adalah fungsi standar dalam Python yang bertugas menghasilkan integer-integer bertahap yang tercakup dalam batas awal dan batas akhir, yang kemudian dikemas dalam sebuah list. List hasil fungsi ini sering dipergunakan dalam perulangan yang akan Anda pelajari pada sub-bab selanjutnya.

Sintaks fungsi ini ada beberapa, tergantung dari banyak parameter yang dipergunakan.

Range dengan 3 parameter

Sintaks :

```
range( <awal>, <akhir>, <langkah> )
```

- `<akhir>` tidak termasuk dalam hasil
- langkah baku adalah 1

Contoh :

```
>>> print range(0,10,2)
[0, 2, 4, 6, 8]
>>> print range(-5,5,2)
[-5, -3, -1, 1, 3]
>>> print range(5,-5,-2)
[5, 3, 1, -1, -3]
```

Range dengan 2 parameter

Sintaks :

```
range( <awal>, <akhir> )
```

- `<akhir>` tidak termasuk dalam hasil
- langkah baku adalah 1

Contoh :

```
>>> print range(0,10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(-5,5)
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>> print range(5,-5)
[]
```

Range dengan 1 parameter

Sintaks :

```
range( <akhir> )
```

- `<akhir>` tidak termasuk dalam hasil
- awal baku adalah 0
- langkah baku adalah 1

Contoh :

```
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(5)
[0, 1, 2, 3, 4]
>>> print range(-5)
[]
```

2.8 Perulangan (loop)

Ada kalanya kita perlu melakukan hal yang sama terhadap banyak objek, yang terlalu menyakitkan dan membosankan jika dilakukan melalui cara biasa.

Contoh yang membosankan :

```
>>> def kali( a,b ):
...     print a, 'x', b, '=', a * b
... <tekan Enter>

>>> kali( 2,4 )
2 x 4 = 8
>>> kali( 4,4 )
4 x 4 = 16
>>> kali( 7,9 )
7 x 9 = 63
>>> kali( 9,12 )
9 x 12 = 108
>>> kali( 15,3 )
15 x 3 = 45
>>> kali( 20,3 )
20 x 3 = 60
>>> kali( 30,2.5 )
30 x 2.5 = 75.0
>>> kali( 17,8 )
17 x 8 = 136
```

2.8.1 Perulangan *for <elemen> in <sekuen>*

Sintaks loop ini adalah :

```
for <elemen> in <sekuen>:
    <proses1>
```

Artinya untuk setiap `<elemen>` dalam `<sekuen>` akan dilakukan `<proses1>`.

Contoh :

```
>>> for c in 'abcde':
...     print '-',c
... <tekan Enter>

- a
- b
- c
- d
- e

>>> for i in range(5):
...     print i * 2
... <tekan Enter>

0
2
4
6
8
```

Eksekusi fungsi kali() pada contoh sebelumnya dapat disederhanakan dengan menggunakan loop ini.

Pertama, kita harus menyimpan bilangan-bilangan yang harus dikalikan ke dalam suatu sekuen. Karena data bilangan ini tidak akan berubah, kita pergunakan tuple untuk menyimpannya.

Contoh :

```
>>> data = (
      (2,4),
      (4,4),
      (7,9),
      (9,12),
      (15,3),
      (20,3),
      (30,2.5),
      (17,8)
      )
```

Kedua, saatnya melakukan loop.
Contoh :

```
>>> for el in data:                    (1)
...     kali( el[0],el[1] )             (2)
... <tekan Enter>

2 x 4 = 8
4 x 4 = 16
7 x 9 = 63
9 x 12 = 108
15 x 3 = 45
20 x 3 = 60
30 x 2.5 = 75.0
17 x 8 = 136
```

(1) `el` adalah elemen yang kita proses, berupa tuple yang berisi 2 item, salah satu contohnya adalah (2,4).
(2) kita panggil fungsi kali() dengan parameter item pertama dan kedua dari `el` yang akan dikalikan oleh fungsi kali().

Karena sekuen yang kita pakai adalah tuple yang bisa diekstrak, kita juga bisa membuat loop seperti berikut, dengan hasil yang sama :
Contoh :

```
>>> for el in data:
...     kali( *el )                     (1)
... <tekan Enter>
```

(1) `el` diekstrak terlebih dulu sebelum diberikan pada fungsi kali().

Atau seperti ini :
Contoh :

```
>>> for el1, el2 in data:              (1)
...     kali( el1,el2 )                 (2)
... <tekan Enter>
```

(1) isi tuple (contoh : (2,4)) dari `data` dipisahkan dan dijadikan objek tersendiri, yaitu `el1` dan `el2`.
(2) `el1` dan `el2` diberikan pada fungsi kali().

2.8.2 Perulangan *while <kondisi>*

Sintaks loop ini adalah :

```
while <kondisi>:
    <proses1>
```

Artinya adalah selama nilai boolean `<kondisi>` adalah True, maka `<proses1>` akan dilakukan.

Contoh :

```
>>> a=5
>>> while a < 10:
        print a
        a += 1

5
6
7
8
9
```

2.9 Modul

Modul adalah paket kode eksternal yang dapat dipergunakan dalam kode program yang sedang Anda bangun. Modul dapat dipergunakan setelah diimpor oleh kode program Anda.

Sintaks :

```
import <modul>                                                    (1)
from <direktori>.<sub-dir>.<sub-dir> import <modul>               (2)
from <modul> import <objek1>, <objek2>, <objek3>, .....           (3)
from <modul> import *                                             (4)
import <modul> as <namaModulBaru>                                 (5)
from <modul> import <objek> as <namaObjekBaru>                    (6)
```

(1) bentuk paling sederhana untuk mengimpor modul. Dipergunakan jika modul yang diimpor berada pada direktori yang sama dengan file kode program Anda, atau berada pada lokasi instalasi Python, atau merupakan modul built-in Python.
(2) jika modul berada pada direktori lain, Anda harus memberikan path yang lengkap dan antar sub-direktori dipisahkan oleh tanda titik.
(3) hanya mengimpor beberapa objek dari modul tersebut
(4) mengimpor semua objek dalam modul tersebut
(5) mengimpor modul dan kemudian mengganti namanya
(6) mengimpor objek dari modul dan kemudian mengganti namanya

Contoh :

```
>>> import math          (1)
>>> help(math)           (2)
Help on built-in module math:

NAME
    math

FILE
    (built-in)

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    atan(...)
        atan(x)

        Return the arc tangent (measured in radians) of x.

    atan2(...)
        atan2(y, x)

        Return the arc tangent (measured in radians) of y/x.
        Unlike atan(y/x), the signs of both x and y are considered.

    ceil(...)
        ceil(x)

        Return the ceiling of x as a float.
        This is the smallest integral value >= x.

    cos(...)
```

```
    cos(x)

    Return the cosine of x (measured in radians).

cosh(...)
    cosh(x)

    Return the hyperbolic cosine of x.

degrees(...)
    degrees(x) -> converts angle x from radians to degrees

exp(...)
    exp(x)

    Return e raised to the power of x.

fabs(...)
    fabs(x)

    Return the absolute value of the float x.

floor(...)
    floor(x)

    Return the floor of x as a float.
    This is the largest integral value <= x.

fmod(...)
    fmod(x,y)

    Return fmod(x, y), according to platform C.  x % y may differ.

frexp(...)
    frexp(x)

    Return the mantissa and exponent of x, as pair (m, e).
    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

hypot(...)
    hypot(x,y)

    Return the Euclidean distance, sqrt(x*x + y*y).
```

```
    ldexp(...)
        ldexp(x, i) -> x * (2**i)

    log(...)
        log(x[, base]) -> the logarithm of x to the given base.
        If the base not specified, returns the natural logarithm (base e)
of x.

    log10(...)
        log10(x) -> the base 10 logarithm of x.

    modf(...)
        modf(x)

        Return the fractional and integer parts of x.  Both results carry
the sign
        of x.  The integer part is returned as a real.

    pow(...)
        pow(x,y)

        Return x**y (x to the power of y).

    radians(...)
        radians(x) -> converts angle x from degrees to radians

    sin(...)
        sin(x)

        Return the sine of x (measured in radians).

    sinh(...)
        sinh(x)

        Return the hyperbolic sine of x.

    sqrt(...)
        sqrt(x)

        Return the square root of x.

    tan(...)
        tan(x)
```

```
        Return the tangent of x (measured in radians).

    tanh(...)
        tanh(x)

        Return the hyperbolic tangent of x.

DATA
    e = 2.7182818284590451
    pi = 3.1415926535897931

>>> help(math.cos)
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
```

1. modul math merupakan modul built-in Python, jadi dapat diimpor secara sederhana
2. Anda bisa mendapatkan penjelasan tentang modul maupun objek yang Anda impor dengan menggunakan fungsi `help()`.

Contoh :

```
>>> import math                                  1
>>> math.ceil(2.2)                               2
3.0
>>> ceil(2.2)                                    3

Traceback (most recent call last):
  File "<pyshell#38>", line 1, in -toplevel-
    ceil(2.2)
NameError: name 'ceil' is not defined

>>> from math import ceil                        4
>>> ceil(2.2)                                    5
3.0

>>> import calendar as KLD                       6
>>> help(calendar.week)                          7
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in -toplevel-
    help(calendar.week)
NameError: name 'calendar' is not defined

>>> help(KLD.week)                                    (8)
Help on function week in module calendar:

week(theweek, width)
    Returns a single week in a string (no newline).
```

(1) impor math secara keseluruhan sebagai modul
(2) kita gunakan fungsi `ceil()` dalam modul math dengan mencantumkan modul beserta fungsinya karena yang kita impor tadi adalah modul math secara keseluruhan
(3) jika Anda mencoba menggunakan fungsi `ceil()` secara langsung, Python tidak bisa menemukannya karena fungsi `ceil()` berada di dalam modul math
(4) impor fungsi `ceil()` dari modul math
(5) sekarang Anda bisa menggunakan fungsi `ceil()` secara langsung
(6) impor modul `calendar` sebagai `KLD`, sehingga sekarang bernama `KLD`
(7) Python tidak bisa menemukan modul `calendar` karena sekarang bernama `KLD`
(8) modul `calendar` yang berganti nama menjadi `KLD` sekarang bisa dipergunakan