Cepat Mahir

Python

Hendri

c4c3m_shadow@yahoo.com

Serial Cepat Mahir

IlmuKomputer.Com

Cepat Mahir Python

Hendri
c4c3m_shadow@yahoo.com

Editor:
Romi Satria Wahono
romi@romisatriawahono.net
http://romisatriawahono.net

Biografi Penulis

Hendri. Saat ini sedang melanjutkan kuliah jenjang S1 di Universitas Bina Nusantara, Jakarta - Indonesia, Jurusan Tehnik Informatika. Sambil bekerja sebagai Application developer di PT Aspri Indonesia (menangani administrasi aplikasi Backoffice, dan membangun beberapa aplikasi dengan PHP, VB6 serta menggunakan DBMS Oracle dan Sun sollaris sebagai Platformnya). Pernah sebagai Developer Python PostgreSQL di PT Rab Indonesia, ikut serta membangun Interface Installer dalam Proyek BPPT kerjasama dengan PT Trustix. Saat ini juga sebagai penulis lepas dan Research and Development untuk Indolinux Magazine, Instruktur Freelance Korporat di beberapa perusahaan swasta. Di samping itu juga pernah mendevelop Aplikasi Pemerintahan untuk Kepegawaian (SIMPEG, Sistem Informasi Manajemen Kepegawaian) di Kabupaten Tulungagung. Aktif dalam organisasi IT diantaranya adalah INTIP(Information for Innovative Project)

Informasi lebih lanjut tentang penulis ini bisa didapat melalui:
Email: c4c3m@oke.net.id

Daftar Isi

Daftar Tabel

Daftar Gambar

Kata Pengantar

Modul ini disediakan bagi pemula yang belum mengenal dan ingin mengetahui seluk-beluk pemograman di Linux. Ditujukan untuk pembaca tingkat awal sampai menengah. Penulis menunjuk bahasa pemograman Python sebagai interface bahasa pemograman yang akan di gunakan.

Python selain dapat dengan mudah di pelajari, juga dapat digunakan untuk mengembangkan perangkat lunak secara cepat (RAD) Rapid Application Development.Aplikasi. Aplikasi yang menggunakan Python juga terjaga integritasnya, seperti pembuatan WebServer stand-alone [1] atau Embedded WebServer Zope (http://www.zope.org)), installer distribusi Linux RedHat, yang dikenal dengan Anaconda Installer (www.redhat.com) dan digunakan juga pada distribusi Linux Pemerintah Indonesia yang bernama WinBI (http://www.software-ri.or.id/winbi). Perusahaan swasta di Indonesia juga telah mengembangkan aplikasi kasir dan rumah sakit di Indonesia (http://www.rab.co.id) dengan menggunakan python ini.

Karena Python di open source khan, maka pendukung programmer Python banyak tersebar di dunia dan membuat modul - modul library tambahan sesuai dengan kebutuhan, juga tersedia modul - modul interface ke Relational Database Management System, seperti MySQL, PostgreSQL, Berkeley-DB dan lainnnya.Sintaks - sintaks yang mudah dipelajari, pemogramannya berorientasi objek dan mudah dibaca pengembang aplikasi, membuat Python banyak digunakan sebagai Aplikasi

[1] Yang berarti dapat berjalan dengan WebServer Lainnya, misalnya Apache

Database, interface konsep CGI(Common Gateway Interface)[2] untuk menggantikan bahasa pemograman Perl.

Dengan keunggulan - keunggulan diatas, penulis menunjuk bahasa pemograman Python sebagai Interface bahasa pemograman yang paling mudah untuk dipelajari untuk pemula dalam pemograman sampai tingkat menengah.Akhir kata semua meteri yang terkandung dalam buku ini jauh dari sempurna, maka dari itu diminta partisipasi pembaca atas komentar, saran dan permintaan penyediaan modul yang belom ada di buku ini, agar bisa di lengkapi lebih lanjut dan untuk kemajuan kita bersama..

Terima kasih atas dukungannya,

Penulis,
Hendri

[2]Aplikasi yang berjalan di sisi Server

Bab 1 Pengantar Python

Dalam bab ini akan dijelaskan konsep - konsep bahasa pemograman pada umumnya, untuk mengetahui dasar - dasar dari pemograman sebelum melangkah ke bahasa pemograman tingkat tinggi Python. Menjelaskan bagaimana sebuah program dapat ditulisi atau dieksekusi dan bagaimana proses yang berjalan di dalam program tersebut.

Untuk menulisi suatu kode program dikenal dengan 2 jenis aplikasi, yaitu Interpreter dan Compiler. Dalam bab ini akan dijelaskan maksud dari aplikasi tersebut termasuk keuntungan dan kerugiannya.

Gambar: Python

Gambar: Uler kekek

Bahasa Pemograman yang akan Anda pelajari adalah Python, Python termasuk dari jajaran bahasa pemograman tingkat tinggi, lainnya Anda mungkin mengenal bahasa pemograman C, C++, Java, Perl dan Pascal.

Bilamana terdapat bahasa pemograman tingkat tinggi, juga dikenal bahasa pemograman tingkat rendah, yang dikenal sebagai bahasa mesin yaitu bahasa pemograman Assembly, Kenyataannya Komputer hanya dapat mengeksekusi bahasa tingkat rendah, jadi bahasa pemograman tingkat tinggi harus melewati beberapa proses untuk diubah ke bahasa pemograman tingkat rendah, hal tersebut merupakan kelemahan yang tidak berarti bagi bahasa pemograman tingkat tinggi.

Tetapi kekurangan tersebut tidak sebanding dengan kelebihannya. Pertama, lebih mudah memprogram sebuah aplikasi dengan bahasa tingkat tinggi. Lebih cepat, lebih mudah dimengerti menulis program komputer dengan bahasa tingkat tinggi, dan juga kesalahan dalam penulisan program cenderung tidak mengalami kesalahan yang berarti. Kedua bahasa pemograman tingkat tinggi lebih portable dalam arti bisa digunakan untuk menulis di berbagai jenis arsitektur komputer (seperti Intel 386, 486, 586, SPARC, RISC/6000) yang berlainan dengan sedikit modifikasi ataupun tidak memerlukan modifikasi sama sekali. Bahasa pemograman tingkat rendah hanya dapat berjalan di satu jenis arsitektur komputer dan harus ditulis ulang untuk menjalankannya di lain mesin, hal ini dikarenakan karena perbedaan urutan register dan services - servicesnya.

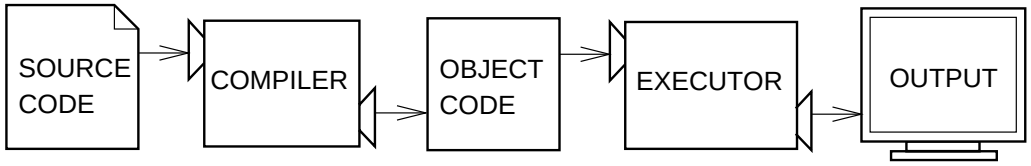
Dengan keuntungan keuntungan tersebut, kebanyakan aplikasi - aplikasi komputer di tulis dengan bahasa pemograman tingkat tinggi. Penggunaan bahasa pemograman tingkat rendah hanya digunakan di aplikasi - aplikasi tertentu.

Terdapat 2 jenis aplikasi untuk memproses bahasa tingkat tinggi ke bahasa tingkat rendah, yaitu : compiler dan interpreter. Sebuah interpreter membaca sebuah program yang ditulis dengan bahasa tingkat tinggi dan langsung menjalankannya per baris, memakan waktu sedikit.

Gambar: Interpreter

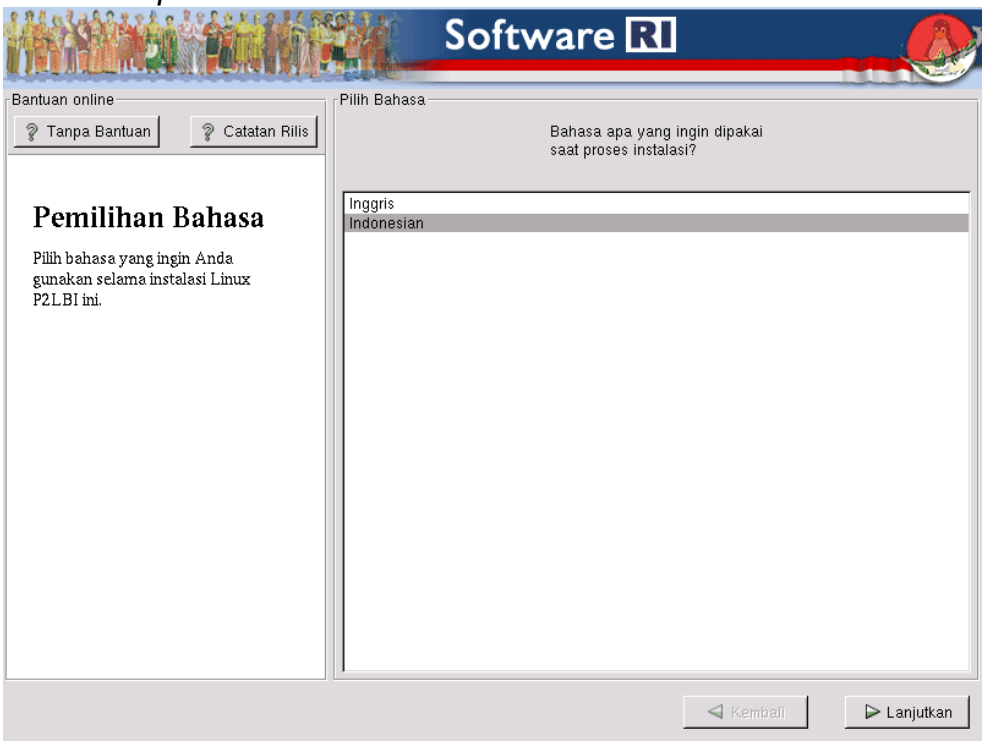
Sebuah kompiler menerjemahkan keseluruhan kode program sebelum menjalankan program tersebut. Dalam kasus ini kode tersebut disebut sebagai source code dan program yang diterjemahkan disebut dengan object code atau executable. Sekali program tersebut di kompilasikan, Anda dapat mengeksekusinya berulang kali tanpa menerjemahkannya lagi kedalam object code.

Gambar: Kompiler

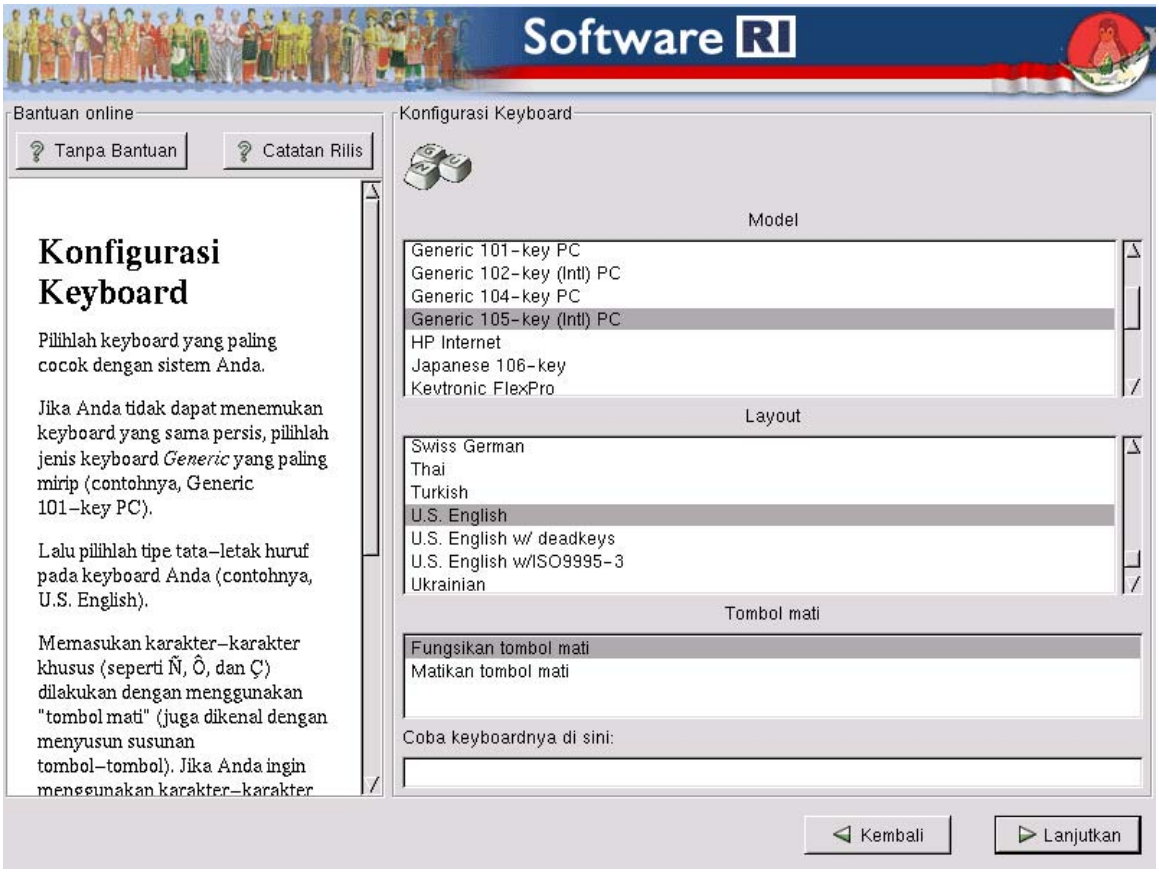


Bekerja pada modus baris perintah sangat baik untuk membuat program dan untuk mencoba - coba algoritma, karena Anda dapat langsung menjalankan perintah tersebut dan melihat hasilnya. Tetapi pada saat Anda ingin membuat program atau aplikasi yang real, Anda seharusnya menyimpan ke dalam bentuk script, jadi dapat Anda jalankan dan dimodifikasi untuk pengembangan program selanjutnya. Contoh aplikasi Python yang dapat kita temui adalah konfigurator - konfigurator dalam suatu distribusi, seperti Xconfigurator dan Installer Distribusi RedHat (Anaconda). Selain untuk melakukan konfigurasi - konfigurasi untuk sistem Linux, Python juga dapat membuat WebServer yang cukup terkenal yaitu Zope(Dapat dilihat di [http://www.zope.org]) dan pada aplikasi database suatu perusahaan swasta di Indonesia telah mengimplementasikan Python dengan PostgreSQL (Sistem Inventori, Rumah Sakit dan Kasir), Billing Warnet dalam bentuk OSCA juga dibuat dengan bahasa pemograman Python([http://www.jakarta.linux.or.id]), Updater RPM Version (swuf) yang dikeluarkan oleh Trustix dan XAlsaKonfigurator yang terdapat pada distribusi buatan Pemerintah Indonesia (WinBi).

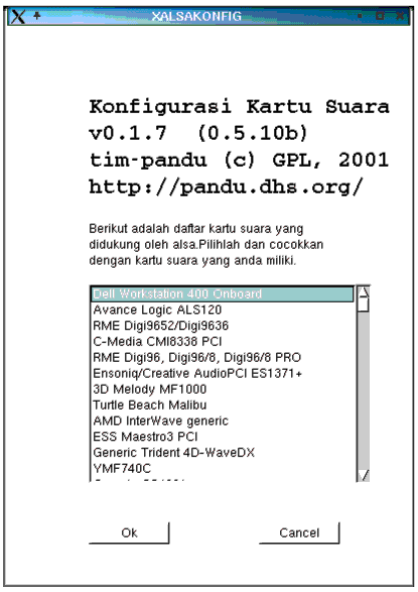
Gambar: Aplikasi Installer WinBI 1



Gambar: Aplikasi Installer WinBI 2



Gambar: Aplikasi XAlsakonfig



1.1 Alur Eksekusi Program

Sebuah program adalah sejumlah instruksi yang berisi perintah - perintah dalam bahasa pemograman komputer untuk menyelesaikan masalah dengan bantuan komputer. Masalah - masalah komputasi tersebut mungkin seperti permasalahan matematika, seperti menyelesaikan sebuah fungsi eksponen, rumus - rumus dalam matematika, tetapi dapat juga berupa mencari dan menggantikan teks, menyusun teks dalam dokumen, dan sebagainya.

Beberapa komponen pada bahasa pemograman komputer mungkin berbeda - beda, tetapi beberapa instruksi umumnya sama di semua bahasa pemograman komputer.

1. input : Masukan dari keyboard, file, atau beberapa device.

2. ouput : Hasil / keluaran program ke monitor display, file, atau beberapa device.

3. math : Perhitungan matematika atau kalkulasi matematika seperti pengurangan, penjumlahan, perkalian, pembagian dan sebagainya.

4. kondisi : Memeriksa beberapa kondisi dan mengeksekusi beberapa perintah tertentu, sesuai dengan kondisi yang telah diperiksa.

5. Perulangan : Menjalankan beberapa perintah secara berulang - ulang kali, biasanya dengan beberapa variasi.

Semua program yang Anda pernah pakai, betapapun rumitnya program tersebut dibuat dengan beberapa instruksi yang telah disebutkan di atas, walaupun kelihatannya sebuah program di bagi - bagi ke modul yang lebih kecil dan dari modul - modul tersebut dibagi - bagi lagi menjadi sub modul - sub modul untuk mengerjakan fungsi - fungsi dasar program tersebut. Ini yang kemudian di kenal dengan Algoritma, akan kita bahas selanjutnya.

1.2 Bagian program Python

Python dikenal sebagai bahasa pemograman interpreter, karena Python dieksekusi dengan sebuah interpreter. Terdapat dua cara untuk menggunakan Interpreter,yaitu dengan mode baris perintah dan modus script. Pada mode baris perintah, Anda memanggil program Python dan sebuah interpreter langsung menampilkan hasilnya :

```
> python
Python 2.1.1 (#1, Sep 24 2001, 05:28:47)
```

```
[GCC 2.95.3 20010315 (SuSE)] on linux2

Type "copyright", "credits" or "license" for more information.

>>> print 1 + 1

2
```

Pada baris pertama pada contoh di atas, perintah python tersebut memanggil program interpreter Python, baris kedua dan selanjutnya menampilkan pesan dan versi dari Interpreter tersebut. baris kelima dengan tanda ">>>", adalah prompt dimana interpreter mengindikasikan bahwa interpreter telah siap untuk diberi perintah. Ketika kita ketikkan print 1 + 1, interpreter langsung meresponnya dengan tampilan output 2.

Cara lain dari modus baris perintah adalah dengan menyimpan perintah - perintah python dalam satu file, yang disebut selanjutnya sebagai script. Contohnya kita mengetikkan perintah-perintah python dengan menggunakan text editor seperti vi, kwrite, emacs, dan lainnya. Pada umumnya file yang berisi script tersebut di simpan dengan extension / akhiran ".py".Misalnya,

```
print 1 + 1
```

perintah tersebut kita simpan dengan nama file "contoh1.py", kemudian untuk mengeksekusinya kita panggil program interpreter python dengan cara :

```
> python contoh1.py

2
```

maka interpreter mengeksekusi script tersebut dengan hasil 2.

1.3 Menggunakan Interpreter

Interpreter Python biasanya di tempatkan di Lingkungan Linux "usr/bin/python". Tergantung di environment PATH nya, bisa dilihat dengan perintah env .

```
PATH=/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/lib/java/bin:/usr/game
s/bin:/usr/games:/opt/gnome/bin:/opt/kde2/bin:/opt/kde/bin:.:/opt/gnome/b
in
```

Pada Unix biasanya berada di /usr/local/bin, sedangkan untuk Linux biasanya di directory /usr/bin. Untuk keluar dari Interpreter Anda dapat menekan tombol kunci [Ctrl][d] . Jika tidak dapat keluar juga, Anda dapat mengetikkan perintah berikut :

```
import sys

sys.exit()
```

1.4 Program singkat Hello Indonesia!

Biasanya, dalam mempelajari sebuah bahasa pemograman komputer, selalu diawali dengan program yang terkenal dengan program "Hello World!". Dalam Python, program tersebut dapat langsung dijalankan sebagai berikut:

```
print "Hello Indonesia!"
```

Hasilnya akan menampilkan string "Hello World!" sebagai berikut :

```
Hello Indonesia!
```

Bandingkan dengan Kedua bahasa pemograman di bawah ini :

Java :

```
class helloindonesia {
     public static void main(String argsp[]) {
          System.out.println(“Hello Indonesia”);
     }
}
```

C/C++ :

```
# Untuk C
#include <stdio.h>
int main() {
     printf(“Hello Indonesia!”);
     return 0;
}

#Untuk C++
#include <iostream.h>
int main() {
     cout << “Hello Indonesia!”);
}
```

tanda kutip dua ("), yang berarti tempat nilai string diletakan pada program ini tidak akan ditampilkan pada layar. Beberapa orang menilai kualitas bahasa pemograman dari kemudahan pembuatan program Hello world!, dalam hal ini bahasa pemograman Python dapat memenuhi kriteria tersebut.

Sebuah kalimat perintah adalah sebuah instruksi yang dapat dieksekusi oleh interpreter Python. Kita telah melihat dua jenis kalimat perintah, yakni print dan pendeklarasian nilai.

Pada saat Anda mengetikkan kalimat perintah pada prompt perintah, maka python mengeksekusinya dan langsung menampilkan hasilnya. Jika ada, hasil dari perintah print adalah sebuah nilai. Pendeklarasian nilai tidak menampilkan hasil.

Pada sebuah script biasanya berisikan beberapa kalimat perintah. Jika lebih dari satu kalimat perintah, hasilnya akan tampil sesuai dengan kalimat perintah yang dieksekusi.

Contohnya;

```
print 1

x = 2

print x
```

Menghasilkan hasil;

```
1

2
```

Sekali lagi, sebuah pendeklarasian nilai tidak menghasilkan output.

1.5 Membuat script di Python

Pada sistem UNIX, Anda dapat langsung membuat script dari Python menjadi executable, dengan memberikan baris perintah sebagai berikut :

```
#!/usr/bin/env python
```

dan mengubah atribut file script python tersebut menjadi executable, dengan :

```
chmod +x <nama-file>
```

Anda dapat melihat atribut - atribut dari file dengan perintah shell, sebagai berikut :

```
ls -l

-rwxr-xr-x  1 skydance root 3066 Feb 28 21:06 playmusic.py
```

1.6 Struktur Identasi

Satu hal yang telah kita ketahui bahwa bahasa pemograman Python adalah bahasa pemograman yang mudah dibaca dan terstruktur, hal ini karena di gunakannya sistem identasi. Yaitu memisahkan blok - blok program susunan identasi. Jadi untuk memasukan sub - sub program dalam suatu blok, sub - sub program tersebut diletakkan satu atau lebih spasi dari kolom suatu blok program. Misalnya :

```
if a = b :
  |print a, 'sama dengan', b
else :
 |print a, 'tidak sama dengan', b
```

Pada contoh diatas kita dapat melihat jika suatu kondisi a = b dipenuhi maka program akan menjalankan baris perintah yang ada di dalam suatu blok kondisi tersebut, yang ditandai dengan penggunaan satu spasi atau lebih dari blok kondisi sebelumnya, dalam contoh diatas perintah yang akan dilaksanakan jika suatu kondisi diatasnya terpenuhi menggunakan dua(2) spasi, sedangkan pada pernyataan else, menggunakan satu spasi. Perbedaan penggunaan spasi ini tidak dianjurkan meskipun dalam program Python dibenarkan, karena struktur program akan lebih sulit dibaca. Seharusnya blok - blok program diatas adalah sebagai berikut :

```
if a = b :
  |print a, 'sama dengan', b #Menggunakan 2 spasi
else :
  |print a, 'tidak sama dengan', b #Menggunakan 2 spasi
```

1.7 Kecepatan Python

Diakui oleh pembuatnya sendiri, Guido Van Rossum bahwa Python memproses suatu program lebih lambat dibandingkan dengan C, bahasa tingkat tinggi lainnya seperti Perl, C++, JAVA, Pascal berada satu tingkat dengan Python. Akan tetapi pada kenyataannya Python boleh dibilang bahasa pemograman yang kecepatannya melebihi C, lho ! kenapa? bukankah Pembuat Pythonnya sendiri yang membuat pernyataan tersebut.

Hal ini dikarenakan karena para Pengembang software lebih cenderung memilih kecepatan dalam menyelesaikan suatu proyek dibandingkan dengan kecepatan proses dari program tersebut, mengenai kecepatan program di jawab dengan

kecepatan prosesor dan memori yang sangat berkembang saat ini, mengakibatkan tidak terlihatnya kelambatan dari program Python, dalam kata lain kecepatan suatu program Python dapat sebanding dengan program yang dibuat dengan bahasa C.

Python diakui oleh banyak para pengembang software memiliki kemudahan dalam pengimplementasian ide - ide mereka, dan memerlukan waktu yang lebih sedikit untuk membuat suatu aplikasi Python, karena dalam Python ini terdapat fasilitas pembuatan tipe data variabel secara dinamik, pemograman berorientasi objek dan memiliki indentasi yang menarik pada struktur bahasa Python yang memudahkan programmer lain untuk membaca program dan memakai ulang modul - modul program tersebut.

Kelemahan python adalah tidak dapat memprogram aplikasi tingkat rendah, seperti pembuatan OS dengan menggunakan C dan assembly, akan tetapi memungkinkan program Python disisipkan kedalam program dengan menggunakan bahasa C itu sendiri.

1.8 Python semi-compiled

Disebut sebagai semi-compiled karena sewaktu program interpreter Python menjalankan program python tersebut, maka secara otomatis interpreter tersebut membuat program python yang terkompilasi dengan nama byte-code , file program python dengan ekstensi berakhiran .pyc yang dapat digunakan secara berulang – ulang. Tetapi tetap harus ada interpreter Python untuk menjalankannya dan ada suatu program wrapper yang di dapat dilihat sourcenya, selebihnya dapat berupa modul yang berekstension .pyc (File binary yang tidak dapat dilihat sourcenya).

Bab 2 Variabel dan Jenis Tipe Data

2.1 Nilai dan Tipe data

Sebuah nilai adalah hal yang paling mendasar seperti sebuah huruf atau sebuah angka yang akan di manipulasi oleh program. Nilai yang selama ini kita lihat adalah 2 (hasil yang kita dapat, ketika kita menambahkan 1 + 1), dan "Hello World!".

Nilai - nilai tersebut berbeda tipe data, yakni 2 sebagai sebuah integer, dan "Hello World!" sebagai sebuah string, disebut string, karena terdiri dari sebuah kata yang terdiri dari beberapa huruf - huruf. Anda dapat mengidentifikasikan string karena mereka di dalam tanda kutip dua(")

Perintah print juga dapat menampilkan integer

```
>>> print 4
```

```
4
```

Bila Anda tidak yakin dengan tipe data yang Anda sebutkan, interpreter dapat memberitahu Anda yaitu dengan menggunakan fungsi built_in `type()` yang ada bersama interpreter.

```
>>> type ("Hello World!")
```

```
<'type string'>
```

```
>>> type 5
```

```
<'type int'>
```

Lebih lanjut, angka desimal dengan tanda (.) dibelakang angka dikenal dengan bilangan pecahan atau float karena angka tersebut merepresentasikan suatu bentuk dengan nama floating point.

```
>>> type (6.5)
```

```
<'type float'>
```

Bagaimana dengan nilai "17.5" dan "5"? Mereka seperti angka - angka, tetapi mereka berada di dalam tanda kutip ("), nah! berarti mereka adalah string.

```
>>> type ("17.5")
```

```
<'type string'>
```

```
>>> type ("5")

<'type string'>
```

Pada saat Anda ingin menuliskan nilai integer yang besar, Anda mungkin menggunakan koma diantara 3 kelompok digit, seperti 1,000,000. Angka tersebut bukan integer yang baik di Python, tetapi itu bisa dilakukan di Python:

```
>>> print 1,000,000

1 0 0
```

Itu bukan tampilan yang kita harapkan bukan? contoh di atas menunjukan bahwa 1,000,000 adalah sebuah tuple (larik / baris), kita akan membahas hal tersebut di bab selanjutnya, jadi sekarang jangan lupa untuk tidak menempatkan koma pada integer - integer Anda.

2.2 Variabel

Fitur yang paling kuat dalam sebuah bahasa pemograman komputer adalah kemampuan untuk memanipulasi variabel - variabel. Sebuah variabel adalah sebuah nama yang mempunyai sebuah nilai.

Pengdeklarasian kalimat membuat sebuah variabel - variabel baru dan memberinya nilai.

```
>>> pesan = "nasi goreng satu!"

>>> banyak = 4

>>> phi = 3.14159
```

Pada contoh di atas, pendeklarasian tersebut menciptakan 3 variabel baru. Pendeklarasian pertama, menunjukkan string "nasi goreng satu!" ke sebuah variabel yang bernama pesan. Kedua, variabel banyak diberi nilai 4 sebagai integer. Dan yang terakhir variabel phi diberi nilai 3.14159 sebagai nilai pecahan.

Cara yang umum untuk pemberian nama variabel adalah dengan tanda panah menunjuk ke nilai variabel tersebut. Jenis ini dinamai dengan state diagram karena menunjukkan nilai - nilai yang merupakan nilai dari variabel - variabel tersebut, contohnya :

```
pesan  => "nasi goreng satu!"

banyak => 4
```

```
phi    => 3.14159
```

perintah print juga berlaku untuk kalimat di atas.

```
>>> print pesan
nasi goreng satu!
>>> print banyak
4
>>> print phi
3.14159
```

Dalam kasus ini, hasilnya adalah nilai dari masing - masing variabel yang disebutkan, kita dapat meminta interpreter untuk memeriksanya.

```
>>> type pesan
<'type string'>
>>> type banyak
<'type int'>
>>> type phi
<'type float'>
```

Tipe dari variabel tersebut adalah tipe data dari nilai yang ada di variabel tersebut.

2.3 Nama variabel dan kata kunci

Pada umumnya, programmer memakai nama variabel sesuai dengan keterangan isi dari variabel tersebut. Nama variabel dapat berupa acak atau bisa apa saja. Dapat berupa angka atau huruf, tetapi harus diawali dengan huruf. Dapat berupa huruf Kapital juga, tetapi harus diingat di Python merupakan case-sensitive, nama Kapital dengan kapital adalah variabel yang berlainan.

Tanda garis bawah (_) dapat muncul di sebuah nama variabel. tanda garis bawah tersebut biasanya digunakan untuk kata - kata yang lebih dari satu, seperti nama_saya, nama_kamu, harga_komputer_di_dusit.

Jika Anda memberikan nama variabel yang sama, Anda mendapatkan sebuah kesalahan sintaks:

```
>>> 123satu = "angka"

SyntaxError: invalid syntax

>>> lebih$ = 50000

SyntaxError: invalid syntax

>>> class = "Pemograman dengan Python"

SyntaxError: invalid syntax
```

variabel `123satu` adalah penamaan variabel tidak benar karena diawali dengan sebuah angka, `lebih$` juga tidak benar karena terdapat karakter yang tidak semestinya ada dalam penamaan variabel, lalu bagaimana dengan class ?

class adalah salah satu kata kunci di bahasa pemograman Python. Kata kunci mendefinisikan aturan -aturan dan struktur bahasa, dan mereka tidak dapat digunakan sebagai nama variabel.

Python mempunyai 28 kata kunci:

```
and     continue  else     for    import   not    raise

assert  def       except   from   in       or     return

break   del       exec     global is       pass   try

class   elif      finally  if     lambda   print  while
```

Anda mungkin ingin menyimpan daftar - daftar ini, pada saat interpreter mengeluarkan kesalahan sintaks dari salah satu nama variabel Anda dan Anda tidak mengetahui penyebabnya, lihat mereka pada daftar ini.

2.4 Mengevaluasi ekspresi

Sebuah ekspresi adalah kombinasi dari nilai - nilai, variabel - variabel dan operator - operator. Jika Anda mengetikkan sebuah ekspresi pada modus baris perintah, interpreter langsung mengevaluasinya dan menampilkan hasilnya:

```
>>> 1 + 1

2
```

Sebuah nilai juga dikenal sebagai sebuah ekspresi, begitu juga dengan variabel.

```
>>> 65

65

>>> x

8
```

Tetapi ingat !, mengevaluasi sebuah ekspresi tidak sama dengan mencetak sebuah nilai.

```
>>> pesan = "nasi goreng satu!"

>>> pesan

"nasi goreng satu!"

>>> print pesan

nasi goreng satu!
```

Pada saat Python menampilkan nilai dari sebuah ekspresi, format yang sama juga akan digunakan untuk mengambil sebuah nilai. Contoh kasus pada strings, yang berarti tanda kutip 2 (") juga di tampilkan pada saat mengevaluasi sebuah nilai. Tetapi pada saat mengeksekusi perintah print, print menampilkan nilai dari string tersebut (tanpa tanda kutip 2(")).

Pada sebuah script, sebuah ekspresi dapat pula berupa kalimat perintah yang benar, tetapi tidak akan menghasilkan nilai dan tampilan hasil apapun. Contohnya:

```
17

3.2

"Hello World!"

1 + 1
```

tidak menghasilkan nilai apapun sama sekali.

2.5 Operator dan Operand

operator adalah simbol-simbol khusus yang merepresentasikan komputasi seperti penambahan dan perkalian. Nilai yang digunakan oleh operator, kemudian disebut sebagai operand.

Berikut adalah ekspresi - ekspresi yang benar dalam Python.

```
20+3  hour-1  hour*60+minute  minute/60 5**2 (5+9)*(15-7)
```

Simbol-simbol +, -, *,/ dan kurung buka dan kurung tutup adalah ekspresi matematika sehari - hari dan dapat berlaku di Python, tanda asteriks (*) berarti perkalian dan tanda asteriks 2(**) berarti tanda eksponen (pangkat).

Pada saat variabel ditempatkan sebagai operand, maka variabel tersebut digantikan dengan nilai dari variabel sebelum perintah tersebut dijalankan.

Anda mungkin akan terheran - heran dengan pembagian, operasi berikut menghasilkan hasil yang tidak kita inginkan.

```
>>> minute = 66

>>> minute / 60

1
```

nilai yang seharusnya muncul adalah 1.1 bukan 1. Hal ini dikarenakan Python melakukan pembagian integer. Karena integer di bagi integer akan menghasilkan nilai integer, maka kita harus melakukan pembagian pecahan dengan cara membuat salah satu operand menjadi float.

```
>>> minute = 66.0

>>> minute / 60

1.1000000000000001
```

2.6 Operator Logika

Terdapat 3 operator logika, yaitu `and, or,` dan `not` . Arti ketiga operator logika tersebut sama halnya dengan arti yang sebenarnya dalam bahasa inggris, Misalnya `x > 8 and x < 20` adalah benar jika kedua kondisi tersebut terpenuhi keduanya dalam arti jika x lebih besar dari 8 dan lebih kecil dari 20 .

`x % 2 == 0 or x % 3 == 0` menghasilkan nilai true jika salah satu di antara kedua kondisi tersebut benar, dan nilai itu akan benar jika nilai x dapat dibagi dengan 2 atau 3.

operator not me-negasikan sebuah ekspresi boolean, jadi `not (x > y)` mempunyai nilai `true`, jika `if (x > y)` mempunyai nilai false .

Operan - operan dalam operator logika harus dalam bentuk ekspresi boolean,

tetapi Python tidak terlalu menegaskan hal tersebut. Semua angka yang bukan merupakan bilangan nol (0) di interpretasikan sebagai kondisi true (benar) atau mempunyai nilai 1(satu). Misalnya :

```
>>> x = 5
>>> x and 1
1
>>> y = 0
>>> y and 1
0
```

2.7 Operator Modulus

Operator modulus bekerja pada bilangan integer (dan ekspresi integer) yang berarti bahwa menghasilkan nilai sisa hasil operan pertama dibagi dengan operan kedua. Di Python, operator modulus di wakili simbol persentase (%). Sintaksnya sama dengan operator - operator lain pada umumnya.

```
>>> pembagian = 5 / 3
>>> print pembagian
1


>>> sisa = 5 / 3
print sisa
2
```

Jadi 5 dibagi 3 hasilnya adalah 1 sisa 2, Modulus operator memiliki kegunaan yang menarik yakni untuk memeriksa suatu angka dapat dibagi dengan angka yang lainnya, Jika x % y menghasilkan nilai 0 maka x dapat dibagi dengan y, contoh fungsi genap :

```
def genap (bil):
  for i in range(bil):
    if (i % 2) == 0 :
```

```
      if i != 0 :
          print i
```

fungsi tersebut akan menampilkan angka - angka genap. Perulangan for akan di bahas pada bab selanjutnya, lalu bagaimana dengan fungsi prima?

```
def prima (bil):
  for i in range (2, bil +1):
    for a in range (2, a + 1):
       if (i % a) == 0  :
           break
    if i == a:
       print i
```

2.8 Aturan pada operasi

Jika terdapat lebih dari satu operator dalam sebuah ekspresi, maka aturan pada operasi tergantung dari aturan presedansi. Python mengikuti aturan presedansi dari presedansi matematika pada umumnya :

1. Operasi yang berada di dalam kurung memiliki nilai presedensi yang tinggi, dan operasi yang di dalam kurung tersebut di proses terlebih dahulu. Misalnya 4 * (5+4) maka hasilnya sama dengan 36, ditambahkan terlebih dahulu 5 dan 4, kemudian baru dikalikan dengan 4.

2. Kemudian nilai presedensi yang tinggi setelah dalam kurung, adalah tanda pangkat, misalnya 3**2 + 8 adalah 17 bukan 13. dan 3 *1**4 hasilnya sama dengan 3 bukan 12.

3. Pembagian dan perkalian memiliki nilai presedansi yang sama, didahulukan terlebih dahulu dibandingkan dengan penambahan dan pengurangan, misalnya 1 + 3 * 2 hasilnya adalah 7 bukan 8.Penambahan dan pengurangan juga memiliki nilai presedensi yang sama.

4. Apabila terdapat satu atau lebih operator yang memiliki presedensi yang sama, maka yang diproses terlebih dahulu adalah bagian sebelah kiri sampai ke kanan, dalam kata lain di evaluasi dari kiri ke kanan, misalnya 4 + 5 - 2 hasilnya adalah 7, 4 *3 / 2 hasilnya adalah 6.

2.10 Operasi pada String

Pada umumnya, Anda tidak dapat melakukan operasi matematika pada string, walaupun string tersebut berupa angka. Berikut adalah contoh - contoh yang salah.

"nasi goreng satu!" + 1 pesan * 5 "7" + 2

Tetapi operator tambah (+) dapat berlaku sesama string, walaupun tidak seperti yang dilakukan pada operasi matematika. Pada operator tambah (+) dalam operasi string, operator tambah (+) dapat diasumsikan sebagai penggabungan antara dua string atau lebih. Contohnya :

```
>>> pesan = "nasi goreng satu!"

>>> banyak = "5"

>>> print "pesan" + pesan + banyak

pesannasi goreng satu!5
```

Bisa kita lihat dalam penggabungan tersebut, antara string dengan string yang lainnya langsung digabungkan tanpa tanda pemisah, seperti spasi atau [tab].

Operator perkalian (*) juga berlaku dalam operasi string, tetapi tidak dapat melakukan perkalian string antar string, melainkan string dengan integer. Operator perkalian ini di analogikan dengan penggandaan string, Misalnya :

```
>>> "ulang" * 3

'ulangulangulang'
```

Masuk diakal juga bila penggabungan dan penggandaan string di analogikan dengan penambahan dan perkalian, seperti yang Anda lihat 4*3 sama dengan 4+4+4, sama halnya seperti ulang*3 dengan ulang+ulang+ulang.Tanda koma (,) dalam operasi string sebagai tanda pemisah (spasi) di antara string.Misalnya :

```
>>> print "ulang", 3, 4, 5

ulang 3 4 5
```

Tipe data sekuensial

Tipe data sekuensial adalah tipe data yang didalamnya dapat terdapat satu atau lebih jumlah elemen anggota, yang dapat diurutkan menurut indeks.

2.10 Lists

Python dapat mengelompokan beberapa tipe data yang berbeda menjadi satu kelompok, yang kemudian dikenal sebagai List , dapat didefinisikan dengan pemisah tanda koma ",".

```
>>> a = ["satu", 2, 3.0, "empat"]
>>> print a
['satu', 2, 3.0, 'empat']
```

Lists bisa dianalogikan sebagai array dan urutan pengaksesannya dimulai dari 0.

```
>>> a[0]
'satu'
>>> a[1]
2
>>> a[-2]
3.0
>>> a[3]
'empat'
```

pengaksesan List pada urutan terakhir dengan nilai -1 .

```
==============================
|'satu' |  2   | 3.0  |'empat'|
| a[0]  | a[1] | a[2] | a[3]  |
|'satu' |   2  | 3.0  |'empat'|
| a[-4] | a[-3]| a[-2]| a[-1] |
==============================
```

List juga dapat dipisah - pisahkan dan dapat digabungkan, ditambahkan dan lainnya.

```
>>> a[0:2]
```

```
['satu', 2]

>>> a[-4:-1]

['satu', 2, 3.0]
```

Tanda titik dua ":" mempunyai argumen [<indeks>:<indeks-n>], berarti dimulai dari indeks sampai indeks ke -n (batas indeks-n, tidak ditampilkan). Di tambahkan, Misalnya :

```
>>> a + ['lima', 'enam']

['satu', 2, 3.0, 'empat', 'lima', 'enam']
```

Penambahannya hanya dapat dilakukan antar lists. Begitupun operasi penggandaan suatu lists, sebagian anggota list ataupun salah satu anggota list.

```
>>> 3*a[:3] + ['tujuh']

['satu', 2, 3.0, 'satu', 2, 3.0, 'satu', 2, 3.0, 'tujuh']

>>> [a[3]] + [a[2]] + ['delapan']

['empat', 3.0, 'delapan']

>>> print a[3]

'empat'
```

Untuk melakukan perubahan terhadap satu anggota atau sebagian anggota list , kita hanya meng-assignkan nilainya, Misalnya :

```
>>> a[2]

3.0

>>> a[2] = a[1] + 5


>>> a

['satu', 2, 7, 'empat']
```

yang berarti nilai a[2] digantikan menjadi nilai a[1] = 2 ditambahkan dengan 5, maka nilai a[2] menjadi 7. Untuk menggantikan sebagian anggota list secara berurutan juga diperbolehkan. Misalnya :

```
>>> a[0:2] = [1,'dua'] #Menggantikan elemen a[0], a[1]
                       #Menjadi a[0] = 1, a[1] = 'dua'
>>> print a
[1, 'dua', 7, 'empat']
```

Menghilangkan beberapa elemen anggota.

```
>>> a[0:2] = []
>>> print a
[7, 'empat']
```

Menyisipkan suatu nilai.

```
>>> a[0:-1] = ['satu']
>>> a
['satu', 'empat']
```

Contoh diatas, berarti menempatkan elemen di antara 0,1 sampai -1. Untuk mengetahui jumlah elemen anggota List, digunakan fungsi built-in len yang berlaku juga untuk menghitung character suatu string.

```
>>> len(a)
2
```

Untuk menambahkan anggota elemen list digunakan metode append yang berlaku pada list. Misalnya :

```
>>> a.append('lima')
>>> a.append('enam')
>>> a
['satu', 'empat', 'lima', 'enam']
```

List di dalam List.

```
>>> b = ['tujuh']
```

```
>>> a.append(b)
>>> a
['satu', 'empat', 'lima', 'enam', ['tujuh']]
```

Berikut metode - metode yang dapat dilakukan dengan object List :

`append (x)`

Menambahkan satu elemen anggota dan diletakkan di bagian indeks akhir pada segment LIST

`extend (L)`

Menggantikan seluruh anggota elemen pada List menjadi seluruh elemen list L

`insert(i, x)`

Menyisipkan satu elemen anggota List pada posisi tertentu

`remove(x)`

Menghilangkan satu anggota list

`pop([i])`

Menghilangkan salah satu anggota tertentu yang telah ditentukan posisinya

`index(x)`

Mengembalikan nilai indeks suatu anggota list

`count(x)`

Memeriksa jumlah x di dalam List

`sort()`

Mensorting list atau mengurutkan anggota list

`reverse()`

Kebalikan dari fungsi `sort()`

Suatu script dari Josh Cogliati jjc@iname.com,

```
menu_item = 0
list = []
while menu_item != 9:
        print "--------------------"
        print "1. Print the list"
        print "2. Add a name to the list"
        print "3. Remove a name from the list"
        print "4. Change an item in the list"
        print "9. Quit"
        menu_item = input("Pick an item from the menu: ")
        if menu_item == 1:
                current = 0
                if len(list) > 0:
                        while current < len(list):
                                print               current,".
",list[current]
                                current = current + 1
                else:
                        print "List is empty"

        elif menu_item == 2:
                name = raw_input("Type in a name to add: ")
                list.append(name)

        elif menu_item == 3:
                del_name = raw_input("What name would you like
```

```
to remove: ")
                if del_name in list:
                        item_number = list.index(del_name)
                        del list[item_number]
                        clr
                else:
                        print del_name," was not found"

        elif menu_item == 4:
                old_name = raw_input("What name would you like
to change: ")
                if old_name in list:
                        item_number = list.index(old_name)
                        new_name = raw_input("What is the new
name: ")
                        list[item_number] = new_name

                else:
                        print old_name," was not found"



print "Goodbye"
```

Latihan

Buatlah program kalender bulan, dengan tampilan sebagai berikut :

Bulan apa [1-12]? 7

Bulan yang Anda pilih ? Juli

Petunjuk : Menggunakan List sebagai daftar nama bulan.

Jawabannya :

```
#!/usr/bin/env python


Bulan = ['Januari', 'Febuari', 'Maret', 'April', 'Mei', 'Juni',
'Juli', 'Agustus', 'September',\

 'Oktober', 'Nopember', 'Desember']

Pilih = input ("Bulan apa [1-12]? ")

if 1 <= Pilih <= 12 :

        print "Bulan yang Anda pilih ?", Bulan[Pilih-1]
```

2.10.1 Fungsi del

Untuk menghapus suatu elemen anggota tertentu dari suatu List, python menyediakan fungsi built-in del. Fungsi tersebut dapat menghilangkan sebagian elemen yang dipenggal - penggal atau digunakan untuk menghilangkan keseluruhan List. Contoh penggunaan List :

```
>>> a = ["satu", 2, 3.0, "empat"]

>>> del a[1]

>>> a

['satu', 3.0, 'empat']

>>> del a[0:2] # atau dengan del a[0:(len(a)-1)]

>>> a

['empat']

>>> del a # Menghapus list a

>>> a

Traceback (most recent call last):

  File "<stdin>", line 1, in ?
```

```
NameError: name 'a' is not defined
```

2.11 String

String dalam pemograman bahasa C dianggap sebagai array of character , hal ini juga berlaku di pemograman bahasa Python.Misalnya :

```
>>> kata = "kata"
>>> kata[0]
'k'
>>> kata[1]
'a'
>>> kata[2]
't'
```

Tidak seperti List, elemen anggota karakter dalam string tidak dapat digantikan,

```
>>> kata[2] = 'p'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Akan tetapi untuk melakukan penambahan karakter pada string dapat dilakukan dengan operator tambah (+). Misalnya:

```
>>> 'per' + kata[0:]
'perkata'
```

Juga terdapat pemenggalan-pemenggalan karakter dalam string.

```
>>> kata[0:2]
'kat'
>>> kata[1:3]
```

```
'ata'
```

2.12 Tuples

Kita telah lihat bahwa string dan list mempunyai banyak kesamaan dalam operasi, pengindeksan dan pemenggalan. Di Python juga terdapat tipe data baru yang dinamakan Tuples yang terdiri atas nilai - nilai angka dan string. Operasi pada tuples hampir sama dengan lists tetapi sifat dari tuples lebih condong ke string, karena beberapa elemen anggotanya tidak dapat langsung di ubah - ubah. Contoh penggunaan tuples :

```
>>> c = 123, 345, 567, 'hello'

>>> c

(123, 345, 567, 'hello')

>>> c[3]

'hello'

>>> len(c)

4
```

Seperti yang kita lihat di atas, tuples menggunakan tanda kurung"()" untuk menyatakan anggota elemen - elemen di dalamnya. Tuple memiliki banyak kegunaan, sebagai fungsi koordinat (x, y)

2.13 Dictionaries

Tipe data lain yang sangat berguna yang terdapat di Python adalah dictionary . Dictionaries pada beberapa bahasa pemograman komputer lain dikenal sebagai "associative memories" atau "associative arrays". Tidak seperti data sekuensial yang diurutkan menurut urutan indeks, dictionary di indeks dengan keys dari dictionary itu sendiri, dimana suatu key dapat berupa angka - angka, bilangan pecahan atau string. Dapat dikatakan dictionary adalah sekumpulan key yang tidak teratur, yang memerlukan suatu key yang unik. Untuk membuat sebuah dictionary hanya memerlukan sepasang tanda kurung kurawal "{}". Seperti arti sebenarnya dalam bahasa inggris, dictionary dapat kita analogikan sebagai kamus. Contoh :

```
>>> indkeing = {}

>>> indkeing['satu'] = 'one'

>>> indkeing['dua'] = 'two'
```

```
>>> print indkeing
{'dua': 'two', 'satu': 'one'}
```

Seperti yang kita lihat, masing - masing anggota elemen dictionary dipisahkan dengan tanda koma "," dan susunan antara key dan nilainya adalah {key:nilai}. Tampilan antara satu elemen dan elemen anggota lainnya tidaklah berurutan, hal ini dikarenakan dictionary bukanlah tipe data sekuensial yang pendataannya berdasarkan indeks, Dictionary metode pendataannya berdasarkan keys jadi tidak perlu menyusun elemen anggota - anggota baru dibuat atau sudah lama ada. Dalam contoh diatas elemen dengan key 'satu' adalah elemen yang pertama kali dibuat tetapi dalam urutannya bukan yang pertama.

2.13.1 Mengakses dictionary

Untuk mengakses suatu nilai dalam dictionary, hal pertama yang harus kita lakukan adalah mengetahui keynya terlebih dahulu. Contoh :

```
>>> print indkeing['satu']
'one'
>>> print indkeing['dua']
'two'
>>> print indkeing['one'] #Operasi ini tidak dapat dilakukan
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: one
>>> #Karena menimbulkan pesan kesalahan kunci. Karena didalam
>>> #dictionary indkeing tidak mempunyai key 'one'
```

2.13.2 Operasi pada dictionary

Perintah del juga berlaku pada dictionary, Misalnya di sebuah toko buah, yang menyediakan berbagai macam buah, dengan stock barang sebagai berikut :

```
>>> stock = {'apel':1, 'jeruk':2, 'manggis':3, 'pepaya':4}
>>> print stock
```

```
{'manggis': 3, 'apel': 1, 'jeruk': 2, 'pepaya': 4}
```

Jika seseorang membeli semua jeruk yang ada di persediaan gudang, maka dalam stock yang ada dalam gudang tersebut habis.

```
>>> del stock['jeruk']
>>> print stock
{'manggis': 3, 'apel': 1, 'pepaya': 4}
```

Atau jika kita menginginkannya lagi atau meminta supplier, kita dapat membuat jumlah jeruk menjadi 0.

```
>>> stock['jeruk']=0
>>> print stock
{'manggis': 3, 'apel': 1, 'jeruk': 0, 'pepaya': 4}
```

Fungsi len juga berlaku di dictionary, yang berarti menghitung jumlah anggota suatu dictionary.

```
>>> len(stock)
4
```

2.13.3 Metode pada Dictionary

Sebuah metode dapat diartikan sebagai sebuah fungsi, meminta argumen parameter dan mengembalikan suatu nilai. Contohnya metode keys , menghasilkan semua keys yang ada dalam sebuah dictionary, dan mengembalikan nilainya dalam bentuk list. Contohnya :

```
>>> stock.keys()
['manggis', 'apel', 'jeruk', 'pepaya']
```

tanda titik "." setelah stock adalah metode / fungsi yang berlaku pada objek tersebut. Pembahasan objek lebih jelasnya akan dijelaskan pada Konsep bahasa pemograman berorientasi objek. Penggunaan metode values sama halnya dengan penggunaan metode keys. Contohnya :

```
>>> stock.values()
[3, 1, 0, 4]
```

Sedangkan metode items menghasilkan kedua nilai, key dan value dan dikembalikan kedalam bentuk list dengan sejumlah tupples.

```
>>> stock.items()
[('manggis', 3), ('apel', 1), ('jeruk', 0), ('pepaya', 4)]
```

Untuk memeriksa keadaan suatu key dalam dictionary, kita dapat menggunakan metode has_key(<nama-key>). Contohnya :

```
>>> stock.has_key('manggis')
1
>>> stock.has_key('lengkeng')
0
```

Yang menghasilkan nilai 1 jika key yang dicari ada, dan nilai 0 jika key tersebut tidak ada.

2.13.4 Alias dan penggandaan

Dictionary adalah tipe data yang elemen anggotanya dapat di ubah - ubah setiap waktu, karena hal inilah Anda harus berhati - hati dalam aliassing, yang berarti 2 objek dengan nama yang berbeda merujuk ke satu objek yang sama, dapat memberikan dampak satu sama lain. Jika Anda ingin memodifikasi sebuah dictionary dan menyimpan dictionary yang aslinya untuk tidak diubah - ubah, Anda dapat menggunakan metode copy. Contohnya :

```
>>> stock = {'apel':1, 'jeruk':2, 'manggis':3, 'pepaya':4}
>>> stock_hsl_trans = stock.copy()
>>> stock_hsl_trans['apel'] = (stock_hsl_trans['apel'] - 1)
>>> stock_hsl_trans
{'manggis': 3, 'apel': 0, 'jeruk': 0, 'pepaya': 4}
>>> stock
{'manggis': 3, 'apel': 1, 'jeruk': 0, 'pepaya': 4}
```

Dapat kita lihat dari contoh diatas, dictionary stock dan stock_hsl_trans tidak mempengaruhi satu sama lain. Sedangkan aliassing keduanya saling mempengaruhi, Misalnya :

```
>>> inventory = stock
>>> inventory
{'manggis': 3, 'apel': 1, 'jeruk': 0, 'pepaya': 4}
>>> inventory['apel'] = 2
>>> stock
{'manggis': 3, 'apel': 2, 'jeruk': 0, 'pepaya': 4}
```

Pengaksesan kedua objek saling mempengaruhi satu sama lain.

Bab 3 Percabangan dan perulangan

3.1 Ekspresi Boolean

Ekspresi boolean adalah ekspresi yang menghasilkan nilai benar atau salah. Di Python ekspresi yang salah di representasikan sebagai nilai 0(Nol) dan ekspresi yang benar sebagai nilai 1(Satu).

Operator == membandingkan 2 nilai dan menghasilkan sebuah ekpresi boolean :

```
>>> 8 == 8

1

>>> 8 == 4

0
```

pada kalimat di atas, 2 operan mempunyai nilai yang sama dan dibandingkan dengan operator boolean == yang menghasilkan nilai 1 (Benar); pada kalimat kedua, 8 tidak sama dengan 4, maka menghasilkan nilai 0 (Salah).

Operator == adalah salah satu dari operator pembanding, yang lainnya adalah :

Table: Operator pembanding

`x != y`	x tidak sama dengan y
`x > y`	x lebih besar dari y
`x < y`	x lebih kecil dari y
`x >= y`	x lebih besar sama dengan y
`x <= y`	x lebih kecil sama dengan y

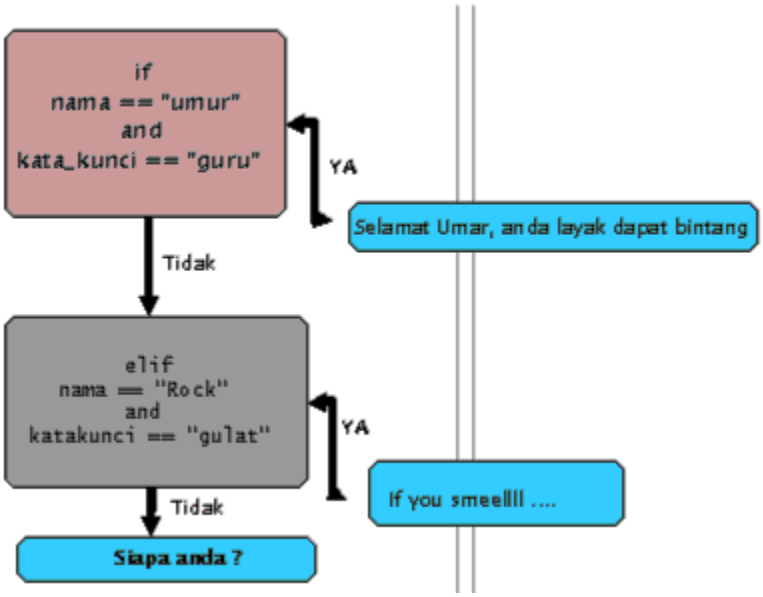
Operator pembanding berikut mungkin tidak asing bagi Anda. Simbol - simbol pada Python berbeda dengan simbol matematika pada umumnya. Pada umumnya kesalahan terjadi pada penggunaan simbol satu = (sama dengan) dengan penggunaan simbol 2 = (sama dengan). Perhatikan! bahwa simbol satu = (sama dengan) adalah operator pendeklarasian dan simbol 2 = (sama dengan) adalah

operator pembanding. Dalam Python juga terdapat simbol >= (lebih besar sama dengan) dan <= (lebih kecil sama dengan).

Contoh operasi boolean yaitu memeriksa kedua kondisi, antara nama dan kata-kunci dengan operator and, yang mengharuskan kedua operand bernilai true.

```
nama = raw_input("Nama Anda ? ")
katakunci = raw_input("Kata kunci ? ")
if nama == "Umar" and katakunci == "guru":
    print "Selamat Umar, Anda layak dapat Bintang!"
elif nama == "Rock" and katakunci == "gulat":
    print "If you smeellll ...."
else:
    print "Siapa Anda ?"
```

Gambar: Flowchart-Umar



3.2 Perintah if

Pada umumnya dalam membuat program, selalu ada kondisi dimana diperlukan pengecekan suatu kondisi untuk mengarahkan program berjalan sesuai keinginan. Seperti halnya kalimat - kalimat perintah lainnya, kalimat perintah if juga mempunyai struktur kalimat, yang terdiri dari bagian atas dan blok - blok perintah di dalamnya.

```
HEADERS :
   KALIMAT PERINTAH PERTAMA ...
   ...
   ...
   KALIMAT PERINTAH TERAKHIR
```

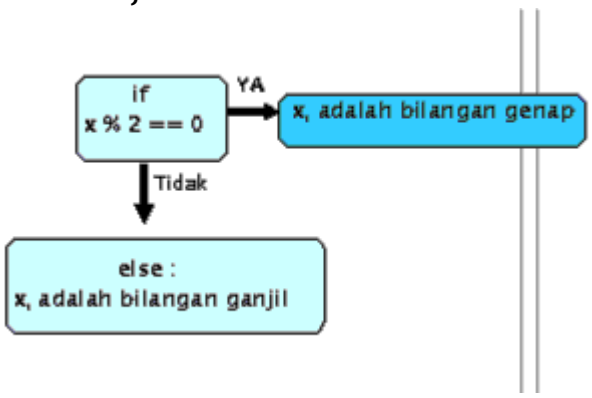
pada bagian HEADERS dimulai dari baris baru dan di akhiri dengan tanda titik dua (:). Kalimat - kalimat perintah yang teridentasi kemudian disebut dengan block baris perintah. Dan isi dari baris perintah tersebut bisa lebih dari satu baris perintah. Misalnya :

```
if i == a :
  print i
```

Selain itu, jika suatu kondisi tidak sesuai dengan kondisi tertentu, Anda dapat membuat kondisi alternatif lainnya, yang berarti terdapat dua kemungkinan dan memeriksanya dengan suatu kondisi untuk menjalankan perintah dari salah satu kondisi tersebut. Misalnya :

```
if x % 2 == 0 :
   print x, "adalah bilangan genap"
else :
   print x, "adalah bilangan ganjil"
```

Gambar: if-else



Jika sisa hasil dari pembagian x dengan 2 adalah 0 maka bilangan x tersebut adalah bilangan genap, jika bukan nol (0) atau salah, maka perintah di dalam HEADER else akan dijalankan.

3.2.1 Perintah if berantai

Terkadang kita memerlukan suatu data memenuhi lebih dari satu syarat, dan kita memerlukan 2 cabang kondisi, salah satu cara untuk mengekspresikan model komputasi tersebut adalah dengan metode kondisi berantai :

```
if x < y :
   print x, "lebih kecil dari", y
elif x > y :
   print x, "lebih besar dari", y
else :
   print x, y, "mempunyai nilai yang sama!"
```

Gambar: Kondisi berantai

elif adalah singkatan dari else if, yang berarti jika kondisi if dilaksanakan maka kondisi berikutnya akan di uji juga, tidak ada batasan mengenai penggunaan kondisi elif, akan tetapi seharusnya dalam akhir kondisi diperlukan adanya kondisi else, sebagai jalan alternatif jika semua kondisi tidak terpenuhi.

Sebagai latihan dari bagian bab ini, coba Anda buat suatu kondisi menu.

```
number = 78

tebak = 0


while tebak != number :

#Selama tebak tidak sama dengan number

   tebak = input ("Tebak suatu angka: ")


   if tebak > number :

      print "Terlalu tinggi"


   elif tebak < number :

      print "Terlalu rendah"


print "Yap ! Anda benar!!!!"
```

3.2.2 Kondisi bersarang

Kondisi bersarang adalah suatu kondisi di dalam kondisi tertentu, Jika terdapat 2 cabang kondisi maka di dalam salah satu cabang kondisi tersebut dapat pula di isi suatu kondisi tertentu.Misalnya :

```
if x == y:
   print x, y "mempunyai nilai yang sama"
else :
   if x > y :
      print x, "lebih besar dari", y
   if x < y :
      print x, "lebih kecil dari", y
```

Kondisi pertama mempunyai 2 pilihan kondisi, kondisi pertama mempunyai perintah baris yang sederhana, sedangkan kondisi kedua mempunyai 2 pilihan kondisi lagi didalamnya. Walaupun pengidentasian dalam Python sangat mudah untuk di baca, akan tetapi akan lebih sulit untuk membacanya secara cepat. Pada umumnya, lebih baik menghindari kondisi bersarang seperti ini.

Operator logika menyediakan suatu cara untuk menyederhanakan kondisi bersarang. Misalnya kita dapat menjalankan perintah berikut dengan menggunakan satu kondisi :

```
if 0 < x :
   if x < 10 :
     print x, "bilangan positif yang terdiri dari satu digit"
```

perintah print akan dijalankan jika kedua kondisi di atas terpenuhi, jadi kita dapat menulisnya dengan cara menggunakan operator logika and :

```
if 0 < x and x < 10 :
   print x, "bilangan positif yang terdiri dari satu digit"
```

Python juga menyediakan struktur kalimat matematika pada umumnya, seperti :

```
if 0 < x < 10 :
```

```
   print x, "bilangan positif yang terdiri dari satu digit"
```

contoh diatas sama artinya dengan contoh - contoh sebelumnya yang menggunakan kondisi berantai dan operator logika.

3.3 Perulangan for

Perintah for dalam python mempunyai ciri khas tersendiri dibandingkan dengan perulangan for pada bahasa pemograman C ataupun Pascal. Tidak hanya mengulang bilangan - bilangan sebuah ekspresi aritmatik(dalam Pascal), atau memberikan keleluasaan si user untuk mendefinisikan perulangan iterasi dan menghentikan perulangan pada saat kondisi tertentu (dalam C). Dalam Python mengulang berbagai macam tipe data sekuensial seperti list, string, dan tuple.

Contohnya:

```
>>> a = ['satu', 'dua', 'tiga', 'empat']
>>> for i in a :
...   print i
...
satu
dua
tiga
empat
```

Contoh diatas berarti fungsi for <iterasi> in <objek>:.

3.4 Perulangan while

Perulangan while akan mengulang didalam ruang lingkup while, selama suatu kondisi terpenuhi. Misalnya :

```
>>> n = 9
>>> while n < 20 :
...  print n
...  n = n + 1
```

```
...
9
10
11
12
13
14
15
16
17
18
19
```

Pada contoh diatas, nilai variabel n akan ditambahkan 1 secara terus menerus sampai kondisi n lebih kecil dari 20.

3.5 Fungsi range()

Jika Anda ingin melakukan perulangan sejumlah yang diinginkan, fungsi built-in range sangat membantu. Fungsi tersebut menghasilkan sejumlah indeks dari nilai yang telah ditentukan. Contohnya :

```
>>> range(15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Ataupun sebagian angka yang diinginkan. Contohnya :

```
>>> range (8, 15)
[8, 9, 10, 11, 12, 13, 14]
>>> range(0,9,3)
[0, 3, 6]
>>> range(0, 20, 3)
```

```
[0, 3, 6, 9, 12, 15, 18]
```

Contoh diatas menunjukan kelipatan dari suatu interval bilangan yang mempunyai sintaks range(<nilai-awal>, <nilai-akhir>, <kelipatan-angka>). Selanjutnya adalah Contoh perulangan for dengan range() :

```
>>> for i in range(10):
...   print i
...
0
1
2
3
4
5
6
7
8
9
```

Mengulang perulangan for sejumlah anggota elemen suatu tipe data sekuensial. Contohnya :

```
>>> a
['satu', 'dua', 'tiga', 'empat']
>>> for i in range(len(a)):
...  print i
...
0
1
2
```

```
3
```

3.6 Perintah break, continue dan else

Perintah break seperti dalam bahasa C, berarti keluar dari ruang lingkup yang terkecil dari kondisi for atau while.

Perintah continue sama halnya dengan di C, yang berfungsi melanjutkan kalimat perintah berikutnya dalam kondisi perulangan.

Pada kondisi perulangan juga diperbolehkan untuk menggunakan kalimat perintah else, yang dijalankan pada saat kondisi perulangan for tidak menemui suatu kondisi atau jika suatu kondisi tersebut mengalami kesalahan / false (dengan while), tetapi bukan pada saat kondisi perulangan dihentikan dengan perintah break. Berikut adalah contohnya :

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, 'sama dengan', x, '*', n/x
            break
    else:
        print n, 'adalah bilangan prima'
```

2 adalah bilangan prima

3 adalah bilangan prima

4 sama dengan 2 * 2

5 adalah bilangan prima

6 sama dengan 2 * 3

7 adalah bilangan prima

8 sama dengan 2 * 4

9 sama dengan 3 * 3

Penjelasannya adalah apabila suatu kondisi dalam perulangan for x in range(2, n) tidak ada yang terpenuhi maka alur perulangannya akan lari ke ruang lingkup perintah else.

3.7 Perintah return

Perintah return dapat menghentikan suatu proses dari fungsi sebelum mengakhiri fungsi tersebut. Alasan Anda menggunakan perintah return adalah jika Anda menemui sebuah kesalahan kondisi, yang berarti nilai suatu fungsi tersebut mengembalikan nilai null (kosong) :

```
import math


def print_log (x):
  if x <= 0 :
    print x, "Bilangan lebih sama dengan 0"
    return
  hasil = math.log (x)
  print "Hasil log dari", x, "adalah", hasil
```

Fungsi `print_log()` mengambil sebuah parameter x. Hal yang dilakukan pertama kali adalah memeriksa apakah nilai x lebih kecil atau sama dengan 0, yang dapat menghasilkan pesan kesalahan jika di proses dalam instruksi perintah selanjutnya maka diberi perintah return untuk keluar dari fungsi tersebut. Alur jalannya program segera dikembalikan ke pemanggil dari fungsi tersebut, dan instruksi - instruksi berikutnya tidak akan dijalankan. Perhatikan! untuk memanggil fungsi dari modul math harus menggunakan perintah import <nama-modul> .

3.8 Rekursif

Telah disebutkan sebelumnya, bahwa kita dapat memanggil suatu fungsi di dalam fungsi lainnya, dan Anda telah melihat beberapa contoh - contohnya. Kita juga dapat memanggil fungsi itu sendiri yang kemudian di kenal dengan istilah rekursif. Mungkin sekilas hal itu tidak memberi alasan mengapa rekursif ini adalah hal yang baik, tetapi akan berubah menjadi program yang menarik.Sebagai contohnya, lihat fungsi berikut :

```
def hitung_mundur (x):
```

```
  if x == 0 :
     print "Sudah nol koq!"
  else :
     print x
     hitung_mundur (x-1)
```

fungsi diatas menampilkan program hitung mundur dari nilai parameter x yang diberikan, parameter tersebut seharusnya sebuah bilangan integer yang positif, dan jika nilai x sama dengan 0 akan menampilkan string yang memberitahu bahwa nilai x adalah 0.Kalau tidak nol(0) maka akan memanggil fungsi itu sendiri dan memberi nilai x-1 sebagai parameternya.

Prosesnya adalah sebagai berikut, jika kita memanggil fungsi tersebut dengan nilai 3, maka nilai 3 akan di check apabila bukan nol (0) maka akan di cetak, dan memanggil fungsi itu sendiri dengan parameter 3-1, yaitu nilai 2, kemudian nilai 2 akan di periksa apakah nilai 2 sama dengan 0, jika bukan maka akan di cetak, dan memanggil fungsi tersebut dengan nilai parameter 1, di check kembali bila bukan nol (0) maka akan akan memberikan parameter x-1 ke fungsi itu sendiri, setelah itu maka fungsi tersebut di beri paramater 0 maka string "Sudah nol koq!" dicetak, kemudian kembali lagi ke fungsi sebelumnya dengan nilai 1, kembali ke nilai 2, kembali ke nilai 3.

Jadi tampilan hasilnya akan seperti berikut.

```
3
2
1
Sudah nol koq!
```

Hal ini akan berbeda jika perintah print diletakkan setelah pemanggilan fungsi rekursif itu sendiri. Misalnya :

```
def hitung_maju(x):
  if x == 0 :
     print "Sudah nol, Mulai!"
  else :
```

```
    hitung_maju (x-1)
    print x
```

maka tampilannya akan menjadi :

```
Sudah nol, Mulai!
1
2
3
```

Latihan

Buatlah fungsi bilangan faktorial, dengan menggunakan metode rekursif diatas.

Jawaban

```
def factorial(n):
   if n <= 1:
      return 1
   return n*factorial(n-1)
```

3.9 Rekursif tanpa batas

Jika fungsi rekursif tidak pernah mencapai kondisi dasar, maka akan memanggil fungsi rekursif secara berulang - ulang selamanya, dan program tersebut tidak akan pernah berakhir. Keadaan ini dikenal dengan Rekursif tanpa batas , dan kondisi ini sangat tidak disarankan. Contoh programnya :

```
def rekursif():
   rekursif()
```

Dalam lingkungan pemograman, sebuah program dengan rekursif tanpa batas tidak pernah benar - benar berjalan selamanya. Interpreter akan menampilkan pesan kesalahan apabila batas maksimal rekursif telah dicapai, Pesan kesalahannya :

```
Traceback (most recent call last):
```

```
 File "<stdin>", line 2, in rekursif

(98 Perulangan telah terjadi)

 File "<stdin>", line 2, in rekursif

RuntimeError: maximum recursion depth exceeded
```

Sebenarnya tampilannya tidak akan seperti di atas bila dijalankan di interpreter Python.

Bab 4 Fungsi

4.1 Pemanggilan fungsi

Anda telah melihat satu contoh pemanggilan fungsi pada bab sebelumnya :

```
>>> type ("54")
<'type string'>
```

Nama fungsi tersebut adalah type dan fungsinya untuk menampilkan tipe data suatu nilai atau suatu variabel. Nilai atau variabel sebagai argumen parameter pada fungsi, harus diletakkan di dalam tanda kurung. Biasanya suatu fungsi mengambil satu atau beberapa argumen dan menampilkan hasil dari fungsi tersebut yang dinamakan nilai hasil.

Walaupun suatu nama variabel tersebut merupakan suatu fungsi, akan tetap menghasilkan nilai dari fungsi tersebut. Misalnya,

```
>>> fungsi = type ("satu")
>>> print fungsi
<'type string'>
```

Sebagai contoh lain adalah fungsi id, yang mengambil suatu nilai atau variabel dan mengembalikan suatu nilai integer yang mewakili identitas dari suatu nilai atau variabel tersebut.Misalnya :

```
>>> id (9)
135109956
>>> angka = 9
>>> id (angka)
135109956
```

Dapat kita lihat bahwa id suatu variabel merupakan id dari nilai variabel tersebut, setiap nilai mempunyai sebuah nomor id , yang unik atau berbeda satu sama lain dan angka tersebut disimpan pada memori komputer.

4.2 Peubah tipe data

Python menyediakan fungsi - fungsi built-in yang dapat mengubah suatu nilai dari suatu tipe data ke tipe data lainnya. Fungsi int mengambil suatu nilai yang mungkin ke tipe data integer, atau menimbulkan pesan kesalahan nilai, Misalnya :

```
>>> int ("30.5")#Ingat segala sesuatu yang di apit dalam  #tanda kutip adalah tipe data string

30

>>> int ("Hello World!")

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

ValueError: invalid literal for int(): Hello World!
```

fungsi int juga dapat digunakan untuk mengubah tipe data float menjadi tipe data int, akan tetapi nilai yang berada di belakang nol akan di hilangkan, Misalnya :

```
>>> int (3.14)

3

>>> int (-7.5)

-7
```

sedangkan fungsi float dapat mengubah nilai integer dan string menjadi nilai float. Misalnya :

```
>>> float (43)

43.0

>>> float ("123")

123.0
```

dari sini terlihat bahwa nilai 43 dalam integer dan nilai 43 dalam float mempunyai nilai matematik yang sama, tetapi berbeda tipe data. Perbedaan tipe data ini terjadi di dalam memori komputer.

Yang terakhir adalah fungsi pengubah nilai string, fungsi str akan mengubah nilai integer dan nilai float menjadi tipe data string. Misalnya

```
>>> str (23)
```

```
'23'

>>> str (45.9)

'45.9'
```

Dalam Python juga terdapat fitur pengubah nilai secara otomatis, yang akan mengubah nilai hasil jika salah satu operandnya mempunyai tipe data yang berbeda, pada contohnya pembagian nilai dengan tipe data integer dengan nilai dengan tipe data float akan menghasilkan nilai dengan tipe data float.

```
>>> 60 / 4.5

13.333333333333334
```

nilai 60 adalah nilai dari tipe data integer, sedangkan 4.5 adalah nilai dari tipe data float, pada python perbedaan tipe data ini akan secara otomatis dirubah hasilnya menjadi nilai yang sesuai.

4.3 Membuat Fungsi baru

Sejauh ini kita hanya menggunakan fungsi - fungsi yang tersedia di Python, kita juga dapat membuat sebuah fungsi baru untuk menyelesaikan masalah yang spesifik.

Dalam konteks pemograman, sebuah fungsi adalah sebuah himpunan dari beberapa instruksi untuk menyelesaikan suatu masalah spesifik yang diinginkan. Sebelum membuat fungsi kita harus terlebih dahulu menspesifikasikan pendefinisian fungsi itu.

Sintaks Pendefinisian fungsi adalah :

```
def NAMA_FUNGSI ( DAFTAR PARAMETER):

  PERINTAH - PERINTAH DALAM FUNGSI
```

Anda dapat menggunakan nama fungsi apa saja sesuai keinginan dan yang menunjukkan kegunaan fungsi tersebut, tetapi Anda tidak dapat menggunakan kata kunci dalam Python sebagai nama fungsi. DAFTAR PARAMETER merupakan parameter - parameter apa saja yang ingin di proses oleh fungsi tersebut, Anda juga diperbolehkan untuk tidak meletakan parameter fungsi tersebut jika tidak ada data yang diproses oleh fungsi tersebut. Perintah - perintah dalam fungsi dapat di letakkan di bawah pendefinisian fungsi dengan kata kunci def yang diletakkan setelah identasi dari kiri di atas, dalam contoh di atas dipergunakan identasi sebanyak 2 spasi.

Contoh fungsi pertama yang tidak mempergunakan parameter :

```
def baris_baru ():
  print
```

Fungsi diatas mempunyai nama baris_baru dan tidak mempunyai parameter di dalam tanda kurung (tempat dimana parameter diletakkan)mempunyai satu baris perintah yang menghasilkan baris kosong yang baru (ini terjadi jika anda menggunakan perintah print dengan tidak meletakan argumen-argumennya). Penggunaan fungsi diatas adalah :

```
print "Contoh fungsi baris baru"
baris_baru ()
print "Dibawah baris baru yang kosong"
```

Tampilan dari perintah di atas adalah :

```
Contoh fungsi baris baru


Dibawah baris baru yang kosong
```

Perhatikan baris kosong diatas, jika kita menginginkan baris kosong lebih dari satu, kita dapat langsung memanggil fungsi baris_baru berulang kali atau membuat fungsi baru yang isi perintahnya terdapat fungsi baris_baru, Misalnya :

```
def tiga_baris ():
  baris_baru ()
  baris_baru ()
  baris_baru ()


print "Contoh fungsi baris baru"
tiga_baris ()
print "Dibawah baris baru yang kosong"
```

Fungsi tiga_baris mempunyai 3 baris perintah yang memanggil fungsi

`baris_baru()`. Jadi dapat kita simpulkan bahwa :

1. Kita dapat memanggil suatu fungsi berulangkali.

2. Kita dapat membuat fungsi yang dapat memanggil fungsi lain.

3. Membuat suatu fungsi memudahkan Anda membuat suatu kelompok kecil untuk menyederhanakan penyelesaian masalah yang kompleks.

4. Kegunaan suatu fungsi adalah membuat program lebih kecil, karena dapat merepresentasikan penggunaan kode yang berulang - ulang.

4.4 Alur eksekusi program

Untuk memastikan bahwa suatu fungsi telah didefinisikan sebelum fungsi itu digunakan, Anda harus mengetahui perintah yang mana yang sekarang Anda jalankan, ini yang di sebut sebagai Alur eksekusi program.

Pada saat program dijalankan selalu dimulai dari perintah pertama dalam program. Perintah - perintah tersebut dijalankan satu kali, dari atas sampai bawah.

Dalam pendefinisian suatu fungsi kita dapat menggunakan fungsi di dalam suatu fungsi, yang berarti kita harus memanggil fungsi yang di atas untuk menjalankan fungsi yang di bawah. Pemanggilan suatu fungsi bisa juga merupakan pengubah alternatif pembacaan alur eksekusi program, karena pemanggilan fungsi di dalam fungsi harus memperhatikan fungsi yang ada diatasnya untuk mendapatkan hasil dari fungsi didalam fungsi tersebut. Jadi intinya Ketika Anda melihat suatu program jangan melihat program tersebut dari atas ke bawah, melainkan perhatikanlah alur jalannya program tersebut.

4.5 Argumen parameter

Beberapa fungsi built-in yang telah Anda gunakan memerlukan beberapa argumen parameter, nilai yang di proses dalam suatu fungsi. Contohnya pada saat menggunakan fungsi built-in type yang memerlukan nilai untuk menentukan tipe data dari nilai tersebut. Beberapa fungsi juga memerlukan argumen lebih dari satu, misalnya fungsi pow yang memerlukan 2 argumen, nilai dasar dan nilai pangkat. Contohnya :

```
>>> pow (3, 1)
3
```

fungsi pow digunakan untuk memperoleh nilai dari eksponen suatu nilai. Jika fungsi pow diisi dengan satu argumen, maka akan timbul pesan kesalahan.

```
>>> pow (3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: pow() takes at least 2 arguments (1 given)
```

Berikut contoh fungsi yang menggunakan suatu argumen.

```
def cetak (kata):
  print "Fungsi ini mencetak=>",kata
```

Fungsi cetak ini memerlukan satu argumen parameter yang bernama "kata", nilai parameter ini dapat kita tentukan untuk menjalankan fungsi ini.

```
cetak ("Hello Fungsi")
```

Fungsi ini mencetak=> Hello Fungsi

```
cetak (54321)
```

Funsi ini mencetak=> 54321

Pada pemanggilan fungsi di atas, nilai argumen pertama bertipe data string, yang kedua bertipe data integer. Jika kita jalankan fungsi cetak tanpa argumen akan menghasilkan pesan kesalahan :

```
>>> cetak ()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cetak() takes exactly 1 argument (0 given)
```

Yang menjelaskan bahwa fungsi cetak, memerlukan 1 argumen parameter.

4.6 Variabel lokal dalam Fungsi

Ketika Anda membuat variabel lokal pada suatu fungsi, variabel tersebut hanya dapat dikenali dalam fungsi itu saja, dan Anda tidak dapat menggunakannya di luar dari fungsi itu. Misalnya :

```
def cetak_lokal (kata):
```

```
  lokal = kata + " tambah lokal"
  print lokal

>>> cetak_lokal ("apa saja")
apa saja tambah lokal

>>> print lokal
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'lokal' is not defined

>>> print kata
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'kata' is not defined
```

perhatikan! bahwa fungsi diatas memerlukan argumen parameter yang bertipe data string, karena pada variabel lokal terdapat operator tambah (+) yang berarti penggabungan antar string. Dan tidak diperbolehkan untuk menampilkan variabel lokal yang terdapat di dalam fungsi cetak_lokal. Parameter suatu fungsi juga bersifat lokal pada fungsi tersebut.

Bab 5 Modul

Jika Anda bekerja dalam lingkungan interpreter, semua modul atau fungsi yang Anda buat akan hilang pada saat Anda keluar dan masuk ke interpreter lagi. Jadi jika Anda ingin membuat aplikasi sebaiknya disimpan dalam sebuah file yang diberi nama script yang telah dijelaskan pada bab sebelumnya.

Sebuah modul adalah sebuah file yang berisi sekumpulan fungsi - fungsi dan instruksi - instruksi program python. Modul tersebut disimpan dengan ekstension .py, pemanggilan modul dijalankan dengan perintah import. Contohnya pada modul luas.py berikut:

```
#Contoh Modul untuk menghitung Luas

#Pendeklarasian VARIABEL

PHI = 3.14

########################


def L_lingk(radius):

        return PHI * (radius*radius)


def K_lingk(radius):

        return PHI * (radius*2)


def L_SegiEmpat(sisi):

        return sisi*sisi


def K_SegiEmpat(sisi):

        print 4 * sisi


def usage():
```

```
  print "Usage : %s [-options] [nilai]" %PROG_NAME
  print """options :
                          -k kelilingS4=sisi          Menghitung
keliling segi empat
                          -L Luaslingkaran=radius  Membaca satu
baris dari isi file
                          -K  --kelilinglingkaran=r  Menentukan
nama file
                          -h help                Menampilkan help
ini
"""
  sys.exit(2)



def main():
        if len(sys.argv[1:]) == 0 :
                usage()
        try :
                optlist, args = getopt.getopt(sys.argv[1:],
'k:L:K:h',    ['kelilingS4=','Luaslingkaran','kelilinglingk',
'help'])


        except getopt.GetoptError :
                usage()



        for o, arg in optlist :
                if o in ['-k', '--kelilingS4='] :
```

```
                        K_SegiEmpat (float(arg))
                if o in ['-L', '--Luaslingkaran'] :
                        L_lingk(float(arg))


                if o in ['-K', '--kelilinglingk'] :
                        K_lingk(float(arg))

                if o in ['-h', '--help'] :
                                usage()




if __name__ == '__main__' :
        main()
```

Bab 6 Input/ Output dan Operasi File

6.1 Input dari Keyboard

Program yang selama ini kita buat, nilainya telah ditentukan sebelumnya, program - program tersebut tidak mendapatkan input dari user. Program - program tersebut hanya melakukan hal yang sama setiap waktu.

Python menyediakan fungsi built-in yang mengambil nilai langsung dari input keyboard. Fungsi yang paling sederhana dinamakan raw_input . Ketika fungsi ini dipanggil, program dihentikan dan menunggu masukan dari user untuk mengetikkan sesuatu. Pada saat user menekan tombol [Enter<-|], maka program tersebut dilanjutkan dan fungsi raw_input mengembalikan apa yang user ketik, sebagai tipe data string, Contoh :

```
>>> input = raw_input()

>>> silahkan ketik disini

>>> print input

silahkan ketik disini
```

Sebelum memanggil fungsi raw_input , sebaiknya kita memasukkan parameter argumen sebagai prompt pada raw_input :

```
>>> nama = raw_input ("Siapa nama anda?")

Siapa nama anda?andriani

>>> print nama

andriani
```

Jika kita mengharapkan input yang akan dimasukan bertipe data integer, kita dapat menggunakan fungsi input :

```
>>> prompt = "Siapa nama anda?"

>>> input (prompt)

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

TypeError: object of type 'string' is not callable
```

Tetapi jika yang dimasukan adalah digit angka atau bilangan maka secara otomatis akan dirubah menjadi tipe data integer, pada contoh di atas yang menjadi parameter dari input bukan berupa bilangan atau integer, maka program akan terhenti dan menampilkan pesan kesalahan. Untuk menghindarinya, sangat disarankan untuk menggunakan fungsi raw_input untuk mengambil nilai dengan tipe data string, kemudian menggunakan fungsi peubah tipe data untuk diubah menjadi tipe data lain yang diinginkan.

Pada saat suatu program berjalan, data tersimpan dalam memori. Kemudian pada saat program berhenti, atau komputer dimatikan, semua data yang tersimpan di memori akan hilang. Untuk menyimpan sebuah data secara permanen, Anda harus meletakan file tersebut ke dalam media penyimpanan, seperti Hard-Drive, CD-ROM, atau floppy disk.

Ketika file - file mencapai jumlah yang banyak, biasanya file - file tersebut dipisah - pisah dan di simpan ke dalam direktori - direktori. Masing - masing file teridentifikasi dengan nomor yang unik, atau dengan kombinasi nama dan tempat direktori file masing-masing.

Mengoperasikan file sama halnya dengan mengoperasikan sebuah buku. Untuk membaca buku, Anda harus membukanya terlebih dahulu. Dan jika Anda selesai dengan buku tersebut Anda harus menutupnya kembali. Ketika suatu buku dalam keadaan terbuka, buku dapat ditulisi dan di baca. Operasi tersebut berlaku juga pada file.

6.2 Membuka File

Membuka sebuah file sama halnya dengan membuat sebuah objek file. Contoh berikut variabel f adalah objek file.

```
>>> f = open ("bulan.py", "r")
>>> type(f)
<type 'file'>
>>> print f
<open file 'bulan.py', mode 'r' at 0x8134fb0>
```

fungsi open, memerlukan 2 argumen parameter. argumen pertama adalah nama file yang akan dibuka, dan kedua adalah mode pembukaan file, contoh diatas file bulan.py dibuka dengan metode r yang berarti hanya dapat dibaca dan tidak bisa ditulisi. Dan jika kita membuka file yang tidak ada, akan muncul pesan kesalahan,

seperti berikut :

```
>>> f = open ("test", "r")

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

IOError: [Errno 2] No such file or directory: 'test'
```

1. metode r, hanya dapat ditulisi jika file tidak ada, akan muncul pesan kesalahan seperti contoh di atas

2. metode w, membuat suatu file baru hanya untuk di tulisi, jika terdapat nama file yang sama dalam suatu direktori maka file yang baru akan menimpa file yang lama.

3. metode a, ditambah dan setiap data yang akan ditulis akan diletakkan pada akhir file.

4. metode r+, dapat dibaca dan ditulisi.

6.3 Metode pada objek File

Untuk mengetahui lebih jelasnya tentang metode - metode yang dapat digunakan pada suatu objek file, kita dapat menggunakan fungsi dir(). Contohnya :

```
>>> dir(f)

['close', 'closed', 'fileno', 'flush', 'isatty', 'mode', 'name',
'read',   'readinto',   'readline',   'readlines',   'seek',
'softspace',   'tell',   'truncate',   'write',   'writelines',
'xreadlines']
```

* metode read(), menampilkan seluruh isi suatu file dengan tipe data string. Misalnya :

```
>>> f.read()

'#!/usr/bin/env python\n\nBulan = [\'Januari\',

\'Febuari\', \'Maret\', \'April\', \'Mei\',

\'Juni\', \'Juli\',\'Agustus\', \'September\',

\\\n \'Oktober\', \'Nopember\', \'Desember\']\n
```

```
Pilih = input ("Bulan apa [1-12]? ")\n

if 1 <= Pilih <= 12 :\n

\tprint "Bulan yang anda pilih ?", Bulan[Pilih-1]'
```

* metode read(), menelusuri isi file sampai akhir file dan apabila metode read() sudah mencapai akhir dari file (atau sudah menggunakan metode read() sebelumnya).

```
>>> f.read()

''
```

* metode f.readline(), membaca isi file perbaris atau sampai menemukan karakter ASCII "\n". Sama seperti halnya penggunaan read(). Metode readline() akan menampilkan string kosong jika sudah mencapai akhir dari file. Contohnya :

```
>>> f.readline()

'#!/usr/bin/env python\n'

>>> f.readline()

'\n'

>>> f.readline()

"Bulan = ['Januari', 'Febuari', 'Maret', 'April', 'Mei', 'Juni',
'Juli', 'Agustus', 'September',\\\n"

>>> f.readline()

" 'Oktober', 'Nopember', 'Desember']\n"

>>> f.readline()

'Pilih = input ("Bulan apa [1-12]? ")\n'

>>> f.readline()

'if 1 <= Pilih <= 12 :\n'

>>> f.readline()

'\tprint "Bulan yang anda pilih ?", Bulan[Pilih-1]'

>>> f.readline()

''
```

* metode write(<string>), memerlukan argumen parameter string, untuk dituliskan ke dalam file dan tidak menghasilkan tampilan hasil. Perhatikan metode file yang digunakan, metode r akan menimbulkan pesan kesalahan jika file dibuka dengan metode write.

```
>>> f = open ("tes", "w")

>>> f.write("Tes untuk menulis ke dalam file")


>>> w = open ("bulan.py", "r")

>>> w.write("TES")

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

IOError: [Errno 9] Bad file descriptor
```

* metode seek(<offset>, <dari>), metode ini akan melihat isi byte pada file pada posisi tertentu. Parameter <dari> jika bernilai 0 maka penelusuran byte dimulai dari awal file, jika bernilai 1 maka pada posisi byte yang sekarang(current position), dan jika bernilai 2 maka dimulai pada posisi byte akhir dari file.

```
>>> f = open ("tes.dat", "r+")

>>> f.write('0123456789')

>>> f.seek(2) #Menuju byte ke dua dari awal file

>>> f.read(1)

'2'

>>> f.seek(-3, 2) #Dimulai dari akhir file, baca 3 byte
#kebelakang.

>>> f.read(1)

'7'
```

* Jika Anda selesai menggunakan file tersebut, Anda dapat menggunakan metode close() untuk menghilangkannya dari memori, setelah melakukan metode close() maka secara otomatis penggunaan kembali file tersebut akan menampilkan pesan kesalahan.

```
>>> f.close()

>>> f.write()

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

ValueError: I/O operation on closed file
```

Contoh Script untuk membuka file

```
#!/usr/bin/env python

##################################################################
######

# Script untuk mengubah/mengupdate file spec dan menyisipkannya
###

# Author Hendri                                                  ###

# GPL License                                                    ###

##################################################################
######
```

```
import os

import sys

dir_spec = sys.argv[1]

dir_trans = sys.argv[2]


os.system("ls %s > daftar.spec" %dir_spec)

os.system("ls %s > daftar.trans" %dir_trans)
```

```
fspec = open ("daftar.spec", "r")

for i in fspec :
        f = open ("%s" %i,"r")


        temp = ""
        for i in f.readlines():
                for  kata in i :
                        if kata[0..6] == "Summary" :
                                temp = temp +  "\n"

                temp = temp + kata
fspec.close()

ftrans = open ("daftar.trans", "r")
trans = []
desc = []

for i in ftrans :
        f = open ("%s","r")
        for i in f.readlines():
                for kata in i:
                        if kata[0..13] == "Summary(id_ID)" :
```

```
                                    trans.append("%s" %kata)
                            if  kata[0..20]  ==  "description  -l
id_ID" :
                                    do
                                     desc.append("%s" %kata)
                                    while ( kata[0..4] == "%prep")
ftrans.close()
```

Bab 7 Konsep OOP Pada Python

Python adalah bahasa pemograman komputer berorientasi objek, yang berarti bahasa Python ini menyediakan fitur - fitur yang mendukung pemograman berorientasi objek. Dan Anda mungkin pernah mendengar tentang konsep bahasa pemograman yang mempunyai banyak kelebihan dibandingkan dengan pemograman prosedural, yaitu reusability dan inheritance yang akan dibahas pada bab ini.

Tidak mudah menjelaskan pemograman berorientasi objek, tetapi kita telah melihat beberapa karakteristiknya :

1. Program - program dibuat dari pendefinisian objek - objek dan fungsi - fungsi, dan kebanyakan perhitungan komputasi diekspresikan kedalam operasi pada objek.

2. Masing - masing pendefinisian objek merujuk ke beberapa objek atau konsep yang sebenarnya pada dunia nyata, dan fungsi - fungsi pada objek dianalogikan sebagai interaksi pada objek.

Pada suatu objek terdapat metode - metode yang dapat digunakan, kita telah melihat beberapa metode, seperti keys dan values,

7.1 Class

Pada saat Anda membuat sebuah class, Anda dapat meminta Python untuk menentukan objek apa yang dapat dilakukan pada suatu class. Hal yang membuat objek dalam class melakukan sesuatu adalah metode. Metode seperti fungsi yang berada di dalam sebuah objek.

```
>>> class Cetak:
...   def cetak_sesuatu (self, string):
...       print "Anda mencetak", string
```

Pada contoh diatas Anda telah membuat class Cetak dengan metode cetak_sesuatu yang mempunyai fungsi untuk mencetak string yang ditentukan. Kemudian buat suatu objek dengan class Cetak:

```
>>> cetak = Cetak()
>>> cetak.cetak_sesuatu("Hello World!")
Anda mencetak Hello World!
```

Sekarang variabel cetak merupakan suatu objek dari class Cetak dan objek cetak dapat melakukan metode yang terdapat pada class Cetak.

Kita dapat memberikan variabel - variabel di dalam metode. Misalnya :

```
>>> class Mencetak :
...  def tentukan_string(self, string):
...      self.kata = string
...  def cetak(self):
...      print "Kata yang anda ingin cetak", self.kata
```

Sama seperti halnya menentukan suatu objek dari class Cetak, maka kita juga dapat melakukannya dengan class Mencetak. Misalnya nama objek itu adalah ngeprint. Berarti objek ngeprint tersebut memiliki dua metode yang terdapat pada class Mencetak, yaitu tentukan_string dan cetak. Pada objek ngeprint juga terdapat anggota variabel yang dideklarasikan dengan kata kunci self, yang berarti anggota dari suatu class.

```
>>> ngeprint = Mencetak()
>>> ngeprint.kata = "Hello"
>>> ngeprint.cetak()
Kata yang anda ingin cetak Hello
```

Kita dapat menentukan anggota variabel lokal dari class tersebut secara langsung atau dengan menggunakan metode tentukan_string. Misalnya :

```
>>> ngeprint.tentukan_string("Hello Class!")
>>> ngeprint.cetak()
Kata yang anda ingin cetak Hello Class!
```

7.2 Inheritance (Turunan)

Seperti yang telah disebutkan diatas, salah satu kemampuan pemograman berorientasi objek adalah inheritance, yang berarti turunan dari suatu class. Pendefinisian sintaksnya adalah sebagai berikut :

```
class NamaClassTurunan (NamaClassDasar):
```

```
   <kalimat perintah ke -1>
   ...
   ...
   ...
   <kalimat perintah ke -n>
```

Nama ClassDasar harus didefinisikan dalam ruang lingkup class turunan. Selain sebuah nama ClassDasar, sebuah ekspresi juga diperbolehkan. Contohnya pada saat menurunkan suatu class yang terdapat dalam suatu modul:

```
class NamaClassTurunan (namamodule.NamaClassDasar):
```

Menjalankan pendefinisian class turunan sama halnya dengan mendefinisikan class dasar. Pada saat suatu objek class dibangun, classDasar disertakan didalamnya. Hal ini digunakan sebagai referensi atribut; jika permintaan terhadap suatu atribut tidak ditemukan di dalam class, kemudian mencari atribut tersebut ke dalam classDasar. Aturan ini berkelanjutan jika classDasar yang ditentukan merupakan classTurunan lain.

Mari kita buat classTurunan dari class Mencetak :

```
>>> class printer (Mencetak):
...   def tambah_kata(self, kata):
...     self.kata = self.kata + kata
```

Pada contoh diatas kita membuat suatu classTurunan dengan nama printer yang diturunkan dari class Mencetak, dan ditambahkan satu metode yang berfungsi menambahkan suatu kata. Contoh penggunaannya :

```
>>> buku = printer()
>>> buku.kata = "Buku Python"
>>> buku.tambah_kata("Bisa dibeli di Amazon.com")
>>> buku.cetak()
Kata yang anda ingin cetak Buku Python Bisa dibeli di Amazon.com
```

Anda telah membuat suatu class turunan yang diturunkan dari class Mencetak!. Seperti yang kita lihat beberapa metode diatas tidak didefinisikan dalam class

printer, tetapi mengapa printer dapat menjalankan metode yang berada pada class Mencetak ? hal ini dikarenakan classTurunan akan mencari atribut yang dipanggil pada classDasar jika atribut tersebut tidak ditemukan dalam class itu sendiri. Anda juga dapat mendefinisikan classTurunan dari beberapa classDasar, seperti halnya satu classDasar, jika atribut yang diminta tidak ada pada class itu sendiri maka akan mencari ke classDasarsatu, kemudian jika tidak ada juga maka akan mencari ke classDasardua begitu selanjutnya.

7.3 Constructor

Dalam Python juga terdapat constructor seperti pada C++, tetapi pada python tidak terdapat Destructor . Constructor ini berfungsi untuk menginisialisasikan sesuatu metode atau variabel pada saat class tersebut dipanggil atau didefinisikan. Sintaksnya adalah sebagai berikut :

```
class NamaClass:

   def __init__ (self, <argumen-parameter>:

      <kalimat perintah ke -1>

      <kalimat perintah ke -n>
```

Contohnya mari kita buat suatu classTurunan dari class printer :

```
>>> class Kalimat(printer):

...   def __init__(self, kata):

...      self.kata = kata

...      self.cetak()

...      self.awal = self.kata + 'Ini variabel awal'
```

dari sini kita langsung dapat menentukan kata yang ingin dicetak pada pendefinisian class.

```
>>> ObjekCTK = Kalimat("Dunia ini")

Kata yang anda ingin cetak Dunia ini
```

seperti yang Anda lihat ketika pendefinisian suatu class, maka akan menset variabel kata dan menjalankan metode fungsi cetak. Bagaimana jika kita tidak memberikan argumen parameter pada saat pendefinisian class? Akan menimbulkan pesan kesalahan :

```
>>> ObjekCTKB = Kalimat()

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

TypeError: __init__() takes exactly 2 arguments (1 given)
```

Tetapi hal ini dapat diatasi dengan memberikan nilai awal pada argumen parameter dengan nilai string kosong ".Misalnya :

```
>>> class KalimatA(printer):

...   def __init__(self, kata=''):

...       self.kata = kata

...       self.cetak()

...       self.awal = self.kata + 'Ini variabel awal'
```

Dengan ini kita dapat mendefinisikan suatu class secara optional , dengan parameter maupun tanpa parameter.

```
>>> ObjekCTKB = KalimatA()

Kata yang anda ingin cetak
```

Bab 8 Pesan Kesalahan (errors and exceptions)

8.1 Penelusuran program

Pemograman komputer adalah sebuah proses yang sangat kompleks, dan karena hal ini dilakukan manusia maka seringkali terjadi kesalahan - kesalahan. Pada umumnya kesalahan - kesalahan dalam pemograman dikenal sebagai bugs dan proses penulusuran program kembali dan memperbaikinya dikenal dengan debugging.

Tiga jenis kesalahan dapat terjadi dalam sebuah program: Kesalahan sintaks, kesalahan pada saat menjalankan program, yang kemudian dikenal sebagai runtime errors dan kesalahan algoritma program, yang kemudian dikenal dengan semantic errors. Sangatlah berguna untuk membedakannya satu sama lain dengan tujuan untuk mendeteksi kesalahan dan memperbaikinya dengan segera.

8.1.1 Kesalahan sintaks

Python hanya dapat mengeksekusi program tersebut hanya jika program tersebut berisi baris - baris perintah dengan sintaks yang benar. Kalau dalam program - program tersebut terdapat kesalahan sintaks maka proses akan berhenti dan menampilkan pesan - pesan kesalahan, yang kemudian dikenal sebagai Syntax errors . Sintaks merujuk ke sebuah struktur program dan aturan - aturan yang berperan dalam struktur tersebut. Sebagai contohnya, dalam bahasa Indonesia, sebuah kalimat harus di awali dengan huruf kapital dan dan diakhiri dengan tanda titik (.), kalimat tersebut akan mempunyai kesalahan sintaks jika penulisan kalimat tidak sesuai dengan aturan yang berlaku. Hal ini juga berlaku di dalam bahasa pemograman komputer.

Pada kebanyakan pembaca, beberapa kesalahan sintaks bukanlah masalah yang serius, seperti penulisan puisi, sajak dan lainnya. Tetapi bahasa pemograman Python bukanlah pemaaf yang baik dalam hal tersebut, jika terdapat satu kesalahan sintaks, maka program langsung memberikan pesan kesalahan dan keluar dari program. Pada waktu Anda baru mulai memprogram, mungkin Anda akan banyak menemui kesalahan - kesalahan sintaks tersebut. Ketika Anda sudah terbiasa memprogram, Anda hanya akan menemui beberapa kesalahan dan menemukan kesalahan tersebut dengan cepat.

Kesalahan sintaks, dapat juga disebut dengan kesalahan dalam memparsing kode python yang salah, umumnya ditemui pada saat Anda baru memulai belajar bahasa pemograman python. Contohnya :

```
>>> while 1 print 'Hello world'
  File "<stdin>", line 1
    while 1 print 'Hello world'
                  ^
SyntaxError: invalid syntax
```

Pada contoh diatas, interpreter memberitahukan bahwa pada perintah terdapat kesalahan sintaks, interpreter akan menampilkan baris yang salah dan menunjukkan posisi kode yang salah dengan tanda panah kecil, contoh di atas pada penggunaan while seharusnya memberi tanda titik dua ":" setelah kondisi while.

8.1.2 Runtime errors

Jenis kesalahan yang kedua disebut dengan runtime errors, disebut begitu karena kesalahan tidak akan muncul sampai Anda menjalankan program tersebut. Kesalahan ini juga dikenal dengan exceptions atau pengecualian karena biasanya mengindikasikan sesuatu pengecualian yang buruk telah terjadi.

Runtime errors sangat jarang terjadi pada program - program yang sederhana pada contoh beberapa bab pertama.

8.1.3 Kesalahan Algoritma

Jenis kesalahan ketiga adalah kesalahan algoritma, yang kemudian dikenal dengan semantic errors. Jika terdapat kesalahan jenis ini dalam program Anda, program Anda akan berjalan dan tidak mengeluarkan pesan - pesan kesalahan, tetapi tidak akan sesuai dengan harapan Anda. Akan terjadi penyimpangan dari keinginan Anda.

Karena program tersebut tidak sesuai dengan harapan Anda dan akan meminta Anda untuk menelusuri kembali program tersebut dari awal untuk memperbaiki algoritmanya, kesalahan ini akan sering muncul pada saat Anda mulai berpengalaman dengan suatu bahasa pemograman.

8.1.4 Penulusuran kembali

Satu hal yang paling penting dan Anda harus punya adalah penulusuran kembali atau disebut kemudian sebagai debugging. Walaupun hal tersebut bisa membuat putus asa, debugging merupakan kekayaan intelektual seseorang yang paling tinggi, menantang dan bagian yang paling menarik dari pemograman.

Pada beberapa cara, debugging bekerja seperti halnya seorang detektif, Anda

dipertemukan dengan berbagai petunjuk dan menulusuri sebuah proses dan kejadian - kejadian yang akhirnya mendapatkan hasil yang Anda inginkan.

Debugging juga seperti bereksperimen. Ketika Anda mendapatkan ide kesalahan apa yang telah terjadi, Anda memodifikasi dan mencobanya lagi. Jika hipotesis Anda benar, Anda akan dapat menerka hasil dari modifikasi tersebut, dan selangkah lebih dekat dengan hasil akhir program tersebut.

Menurut pendapat beberapa orang, pemograman dan debugging adalah hal yang sama. Jadi pemograman adalah sebuah proses yang harus melalui proses beberapa kali debugging untuk mendapatkan hasil yang Anda inginkan. Pada contohnya, Linux adalah sebuah sistem operasi, yang berisikan beribu - ribu baris kode perintah, tetapi program tersebut diawali dengan program sederhana yang Linus Trovalds gunakan untuk mengeksplorasi chip Intel 80386. Berdasarkan keterangan Larry Greenfield, "Proyek pertama linus adalah sebuah program yang mencetak AAAA dan BBBB secara bergantian. Kemudian program ini berevolusi menjadi Linux." (Majalah Linux Users's Guide Beta Version 1)

8.1.5 Penulisan Komentar

Dalam proses debugging, suatu komentar instruksi program sangat berguna sekali dalam pembacaan suatu kode. Pada umumnya komentar berisi keterangan tentang kegunaan suatu fungsi itu. Sintaksnya adalah tanda kres atau tanda pagar "#". Setelah meletakkan tanda tersebut, kita dapat mengetikan kalimat apa saja yang berhubungan dengan suatu instruksi perintah, sebab apapun kalimat tersebut tidak akan di proses oleh interpreter. Contohnya :

```
print Hello World! #Mencetak string "Hello World!" ke layar.

print 4 + 5 #Menampilkan hasil dari bilangan 4 + 5.
```

Anda mungkin telah banyak melihat pesan - pesan kesalahan yang terjadi, terdapat dua jenis yang tidak dapat dipisahkan, yaitu : kesalahan sintaks dan pengecualian(exceptions).

8.2 Pengecualian (exceptions)

Jika terjadi kesalahan pada saat program dijalankan (run-time errors), program tersebut membuat sebuah pengecualian (exceptions). Biasanya program terhenti dan menampilkan pesan kesalahan. Contohnya pada pembagian bilangan dengan nol :

```
>>> 40 / 0

Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

Juga untuk pengaksesan yang bukan elemen anggota dari suatu list.

```
>>> a = []
>>> a[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

Atau mengakses sebuah key yang tidak ada pada suatu dictionary.

```
>>> c = {}
>>> print c['polo']
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: polo
```

Pada setiap kasus, pesan kesalahan dibagi menjadi dua bagian: Jenis kesalahan sebelum titik dua, dan menjelaskan secara spesifik tentang kesalahan tersebut dibagian setelah titik dua. Pada umumnya, interpreter Python juga menampilkan sebuah penulusuran kembali dimana kesalahan pada program tersebut, yang dapat kita lihat sebelumnya.

8.3 Menangani pengecualian (Handling exceptions)

Kadang - kadang kita ingin menjalankan sebuah operasi yang dapat menyebabkan kesalahan pada runtime, tetapi kita tidak mau program tersebut berhenti total. Kita dapat mengatasinya dengan perintah try dan except.

Contohnya, Kita memberikan prompt pada user untuk menentukan nama file yang akan dibuka. Jika file tersebut tidak ada, kita tidak menginginkan program tersebut berhenti total yang dikarenakan tidak adanya file tersebut, kita menginginkan untuk mengendalikan kesalahan tersebut, dengan :

```
namafile = raw_input("Masukan nama file: ")
```

```
try :
  f = open (namafile, "r")
except :
  print "Nama file tidak ditemukan!"
```

Perintah try menjalankan perintah pada blok pertama. Jika tidak ada kesalahan yang terjadi, perintah except akan diabaikan. Tetapi jika terjadi kesalahan apapun jenisnya, program akan menjalankan perintah - perintah pada ruang lingkup except dan kemudian program tersebut berlanjut.

Kita dapat menyatukannya dan menjadikannya suatu fungsi, yaitu fungsi exist. fungsi ini akan mengambil argumen parameter sebuah nama file dan mengembalikan nilai true, jika file tersebut ada dan mengembalikan nilai false, jika file tersebut tidak ada. Contohnya :

```
def exist(namafile):
  try :
    f = open (namafile)
    f.close()
    return 1
  except :
    return 0
```

Anda dapat menggunakan kalimat perintah beberapa blok except (lebih dari satu) untuk menangani beberapa jenis kesalahan, detailnya Anda dapat lihat di Python Refference Manual. Jika pada program Anda mendeteksi adanya kesalahan, Anda dapat membuatnya sebagai sebuah pengecualian (exception). Pada contoh ini kita akan membuat suatu fungsi yang mengecheck umur seseorang.

```
def check():
   umur = input("Masukan umur anda? ")
   if umur <= 17:
     raise 'DibawahUmur!', 'Anda harus 17 tahun keatas'
   return umur
```

```
check()
```

Maka program tersebut jika dijalankan dengan mengisi umur dibawah 17 tahun, akan menimbulkan suatu kesalahan exception.

```
Masukan umur anda? 3
Traceback (most recent call last):
  File "umur.py", line 6, in ?
    check()
  File "umur.py", line 4, in check
    raise 'DibawahUmur!', 'Anda harus 17 tahun keatas'
DibawahUmur!: Anda harus berumur 17 tahun keatas
```

Perintah raise akan mengambil dua argumen: jenis kesalahan dan informasi yang spesifik tentang kesalahan tersebut. DibawahUmur! adalah jenis kesalahan exception baru yang dibuat pada fungsi tersebut.

Latihan

Buatlah sebuah exception yang mengharuskan untuk memasukkan nilai minus (-1) dan menampilkan pesan kesalahan (exceptions) jika yang dimasukkan bukanlah tipe data bilangan integer atau float.

Jawaban

```
print "Ketikkan Control C atau -1 untuk"
number = 1
while number != -1:
   try:
     number = int(raw_input("Masukan angka: "))
     print "Anda memasukan: ",number
   except ValueError:
     print "Bukan angka bilangan integer atau float!"
```

Bab 9 Konektivitas Database

9.1 Instalasi PostgreSQL

Untuk menginstall PostgreSQL, anda dapat mengambil source codenya di http://www.postgresql.org , kemudian pilih mirror site terdekat dan di compile atau dengan bentuk RPM.

* Install PostgreSQL dari tar ball

Setelah anda download, ekstrasikan tar ballnya, kemudian lakukan perintah sebagai berikut pada console linux :

```
./configure

gmake

gmake install

adduser postgres

su - postgres

/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data

/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data
>logfile 2>&1 &

/usr/local/pgsql/bin/createdb test

/usr/local/pgsql/bin/psql test
```

* Install dari Paket RPM

Anda tinggal mengetikan perintah `rpm -ivh <namapacket.rpm>` pada console linux, dimana :
- i → Intall,
- v → Verbose,
- h → Hash.

9.2 Menambahkan user dan database

Untuk menambahkan user, anda harus login ke Superusernya PostgreSQL, yaitu postgres. Kemudian anda login ke database dengan menjalankan perintah psql pada console.

```
 psql
Welcome to psql, the PostgreSQL interactive terminal.


Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit
postgres=#
```

Untuk membuat user baru, anda jalankan perintah create user, dengan Sintaks sebagai berikut :

```
CREATE USER username
    [ WITH
     [ SYSID uid ]
     [ PASSWORD 'password' ] ]
    [ CREATEDB   | NOCREATEDB ] [ CREATEUSER | NOCREATEUSER ]
    [ IN GROUP     groupname [, ...] ]
    [ VALID UNTIL  'abstime' ]
```

9.3 PygreSQL

Untuk melakukan interfacing dengan Python, kita harus mempunyai modul PyGreSQL yang dapat menggunakan fitur - fitur unggulan pada PostgreSQL. PyGreSQL dikembangkan dan sudah diujicobakan pada platform NetBSD.

9.4 Penginstalasian PygreSQL

Jika anda menginstall PostgreSQL dari source codenya, anda tinggal menambahkan option --with-python pada saat menjalankan perintah configure. Tetapi jika mesin anda berjalan dengan arsitektur i386, anda bisa download RPM nya ftp://ftp.druid.net/pub/distrib/pygresql.i386.rpm .

Distribusi RPM tersebut terdiri atas :

```
README      - file
Keterangan

  Announce
- Pengumuman rilis

  ChangeLog    - Perubahan
yang terjadi pada paket

  pgmodule.c   - Modul
C untuk Python

  pg.py
- PyGreSQL DB class.

  pgdb.py
- DB-SIG DB-API 2.0 compliant API wrapper for PygreSQL

  tutorial/
- direktori demo
```

Terdiri dari : `basics.py, syscat.py, advanced.py, func.py and`

`pgtools.py`. yang berisi tentang demo menggunakan Python sebagai interface dari PostgreSQL.

9.4.1 Modul pg pada PygreSQL

Untuk mengkoneksi ke database, kita gunakan modul connect. Yang memerlukan beberapa argumen parameter. Sintaksnya sebagai berikut :

```
connect(dbname, host, port, opt, tty, user, passwd)
```

* dbname => Nama database (dapat berupa string atau tidak diisi (None))

* host => Nama dari host server (dapat berupa string atau tidak diisi (None))

* port => Port yang digunakan oleh Server Database (berupa tipe data integer / -1)

* opt => Option pada koneksi (dapat berupa string atau tidak diisi (None))

* tty => Terminal untuk mendebug (dapat berupa string atau tidak diisi (None))

* user => Nama user PostgreSQL (dapat berupa string atau tidak diisi (None))

* passwd => Password user (dapat berupa string atau tidak diisi (None))

Mengembalikan nilai berupa objek, yaitu pgobject yang berfungsi untuk menangani koneksi database.

Kesalahan - kesalahan yang mungkin terjadi :

* TypeError => kebanyakan argumen, atau kesalahan pada argumen.

* SyntaxError => pendefinisian argumen secara ganda

* pg.error => Beberapa kesalahan yang terjadi pada saat pendefinisian koneksi ke database.

9.4.2 pgobject

Objek ini menangani sebuah koneksi ke database PostgreSQL dan mempunyai beberapa metode:

query

metode query berfungsi untuk menjalankan perintah - perintah dasar SQL. Sintaksnya adalah sebagai berikut :

query(<perintah-perintah SQL>)
Yang memerlukan parameter berupa perintah - perintah SQL dengan tipe data string. Tidak mengembalikan nilai, kesalahan yang mungkin terjadi adalah :

* TypeError => kesalahan pada penulisan argumen

* ValueError => tidak adanya perintah SQL

* pg.error => kesalahan pada saat memproses suatu query,
atau koneksi yang salah.

getresult

Mengembalikan hasil dari suatu query.

sintaksnya :

```
getresult()
```

Mengembalikan nilai bertipe data LIST.

Kesalahan yang mungkin terjadi :

* SyntaxError => Terlalu banyak parameter([footnote] metode getresult, tidak memerlukan parameter
apapun)

* pg.error => Kesalahan pada hasil sebelumnya

dictresult

Sama seperti metode getresult, tetapi metode ini mengembalikan
nilai dengan tipe data dictionary.

listfields

Metode ini menampilkan seluruh daftar - daftar nama fields,
hasil dari query sebelumnya.

sintaksnya :

```
listfields()
```

Mengembalikan nilai bertipe data LIST yang berupa daftar
nama - nama fields.

fieldname

Metode ini berguna untuk mencari sebuah nama field dari urutan
angkanya.

Sintaksnya :

```
fieldname(<integer>)
```

Memerlukan parameter bertipe data integer yang berarti urutan nomor suatu field. Mengembalikan nilai berupa string yang menunjukkan nama dari field tersebut.

fieldnum

Metode ini untuk mengetahui urutan nomor dari suatu field yang memerlukan parameter bertipe data integer.

Sintaksnya :

```
fieldnum(<string>)
```

Mengembalikan nilai berupa urutan nomor field, bertipe data integer.

reset

Metode ini untuk mereset koneksi ke database.

Sintaksnya :

```
reset()
```

Tidak memerlukan argumen apapun, dan tidak mengembalikan nilai apapun.

close

Berfungsi untuk memputuskan koneksi ke database,

Sintaksnya :

```
close()
```

Berikut contoh sebuah aplikasi untuk koneksi ke server dan menjalankan perintah - perintah SQL.

```
#!/usr/bin/env python
```

```
import pg



TRY = 0
while TRY <= 3:
    admin = raw_input("Admin name: ")
    password = raw_input("Password: ")
    try :
        PG    =    pg.connect(user=admin,    passwd=password,
dbname='cacem')
        print "Congratulation! You're Log IN"
        PG.close()
        break
    except pg.error:
        if TRY == 3 :
            print "GoodBye !!!!"
        else :
            print "Wrong Password! You have", 3 - TRY, 'chances!'


TRY = TRY + 1
a = ''
while 1:
        a = raw_input("SQL==> ")
        if a in ('\q', 'quit') :
                print "Good Bye ...."
                break
```

```
        try :
                sql = PG.query("%s"%a)
                print sql
        except :
                print "Wrong SQL Syntax!"
```

Penutup

Akhir kata, penulis meminta kesediaan pembaca untuk ikut mengkontribusikan saran, komentar, pesan, permintaan untuk menuliskan dokumentasi yang berkaitan dengan buku ini, serta di tunggu kontribusi artikel yang berkaitan dengan Python.

Referensi

1. http://www.python.org
2. Modul Pelatihan RAB Linux Indonesia
3. Modul Pelatihan Linuxindo, Hendri
4. http://www.thinkpython.com
5. Core Python Programming by Wesley Chun
6. Python Pocket Reference by Mark Lutz