Ridwan Fadjar Septian

python™

Belajar Pemrograman Python Dasar

- Instalasi Python di Linux dan Windows
- Cara Menggunakan Python
- Mengenal Tipe Data dan Operator
- Membuat Pemilihan Kondisi
- Menyusun Looping
- Mengenal Data Struktur Python Tingkat Lanjut
- Membuat Function
- Mengenal Exception
- Membuat File
- Pengenalan Class
- Pengenalan Module
- Teknologi yang Menggunakan Python

POSS
Jl. Dr. Setiabudhi No. 229 Bandung 40154
Jawa Barat - Indonesia
Website : http://www.poss-upi.org
Twitter : @possupi

Hak Cipta

28 September 2013

Buku ini diperbaharui terus dan bisa didapatkan di http://www.poss-upi.org/download. Jika Anda mendapatkannya di tempat lain kemungkinan besar merupakan versi lama.

Buku Serial Open Source

Belajar Pemrograman Python Dasar

Penulis : Ridwan Fadjar Septian
Editor : Andri Priyanto
Desain Sampul : Herdi Agustina
Pengawas Proyek : Zia Ulhafiedz
Penanggung Jawab : Erik Romadona

Disusun oleh : POSS – UPI

Telp 0898 6964 714
e-mail : poss@upi.edu
website : http://www.poss-upi.org
twitter : @possupi

Versi ke – 1, 28 September 2013

Kritik dan saran dapat menghubungi POSS – UPI di Jl. Dr. Setiabudhi No. 229 Bandung 40154 Jawa Barat – Indonesia. Atau pada no kontakdan email yang terera diatas

Kata Pengantar

Puji dan syukur kita panjatkan atas kehadirat-Nya karena dengan rahmatnya buku ini dapat disusun hingga versi pertama ini. Python merupakan salah satu bahasa pemrograman dan penunjang pada perkembangan FOSS dan Linux. Tidak hanya kedua itu di perkembangan teknologi lainyya Python hadir sebagai salah satu teknologi mumpuni yang patut kita pelajari. Biasanya Python sudah tertanam di distro – distro tertentu seperti Ubuntu, Fedora, dan Slackware. Python sendiri merupakan bahasa pemrograman yang mendukung paradigma *object oriented programming* ataupun *scripting.* Python hadir di ranah sistem operasi, virtualisasi, jaringan komputer, grafika komputer, kecerdasan buatan, teknologi web, *game,* dan lain sebagainya.

Buku ini hadir bagi yang ingin dasar – dasar Python. Buku ini ditujukan bagi *programmer* pemula atau yang sudah ahli tapi ingin mencoba bahasa pemrograman Python. Buku ini juga merupakan penghantar untuk manju ke teknologi Python lainnya. Selain itu, dengan menggunakan bahasa Indonesia diharapkan dapat lebih menjaring *programmer* lokal untuk berlajar bahasa pemrograman Python dan menambah khazanah dunia pemrograman Python di Indonesia.

Terima kasih penulis ucapkan kepada Bramandityo Prabowo, kakak tingkat dan anggota POSS – UPI juga yang sudah mengajarkan penulis untuk belajar bahasa pemrograman Python hingga mengenal beberapa teknologi Python yang layak untuk dipelajari. Terima kasih juga kepada Ricko dan Zia sebagai anggota POSS – UPI yang telah menginspirasi penulis untuk menggunakan Linux. Terima kasih juga kepada Herdi Agustina karena desain sampul bukunya yang memukau, Andri Priyanto yang membantu *editing* buku ini, Zia sebagai pengawas pengerjaan buku ini, dan Erik Romadona sebagai pengarah hingga buku ini selesai ditulis.

Kritik dan saran kami tunggu demi kemajuan buku ini. Semoga Anda mendapatkan manfaat dalam mempelajari bahasa pemrograman Python dan ikut memajukan teknologi informasi dan komunikasi di Indonesia.

Penulis

Daftar Isi

1. Instalasi Python

Lingkungan Python

Dari http://www.tutorialspoint.com/python/python_environment.htm , terdapat beberapa lingkungan yang dapat dihuni Python. Berikut adalah beberapa lingkungan sistem operasi yang bisa dihuni Python:

- Win 9x/NT/2000
- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, dan lain - lain)
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion
- Python juga dirancang ulang di Java dan .NET Virtual Machine

Install Python di Linux

Unduh *installer* Python untuk Unix/Linux di http://www.python.org/download. Kemudian. Unduh *source code* Python yang dibungkus oleh *zip* dan ikuti alur pengunduhannya. Setelah diunduh kemudian ekstrak *installer* tersebut. Kemudian masuk kedirektori installer Python. Jika ada pengaturan khusus yang diinginkan, *edit file* Modules/Setup pada *installer*. Jika pengaturan sudah sesuai jalankan perintah **./configure**. Hal ini dilakukan untuk konfigurasi *installer* Python pada sistem operasi kita. Setelah konfigurasi beres, jalankan perintah **make** untuk meng-*compile installer* Python. Setelah itu baru jalankan perintah **make install** untuk memulai proses instalasi.

File ekskeusi Python akan diinstall di /usr/local/bin dan *library* nya diinstall di /usr/local/bin/pythonX.X. Kemudian jangan lupa untuk lakukan pengaturan *path* agar bisa

dieksekusi di *shell*. Berikut adalah beberapa pengaturan *path* pada beberapa *shell* :

- pengaturan *path di shell* bash : export PATH = "$PATH:/usr/local/bin/python" kemudian tekan enter
- pengaturan *path di shell* csh : setenv PATH = "$PATH:/usr/local/bin/python" kemudian tekan enter
- pengaturan *path di shell* ksh : PATH = "$PATH:/usr/local/bin/python" kemudian tekan enter

Selain menggunakan *source code* Python dan diinstall secara manual seperti diatas. Anda bisa menginstall Python melalui *packet manager* jika sistem operasi yang Anda gunakan mendukung fitur tersebut seperti Ubuntu, Slackware, dan Fedora.

Di beberapa sistem operasi kadang sudah tertanam di dalamnya dan langsung bisa digunakan seperti di Ubuntu dan Fedora.

Install Python di Windows

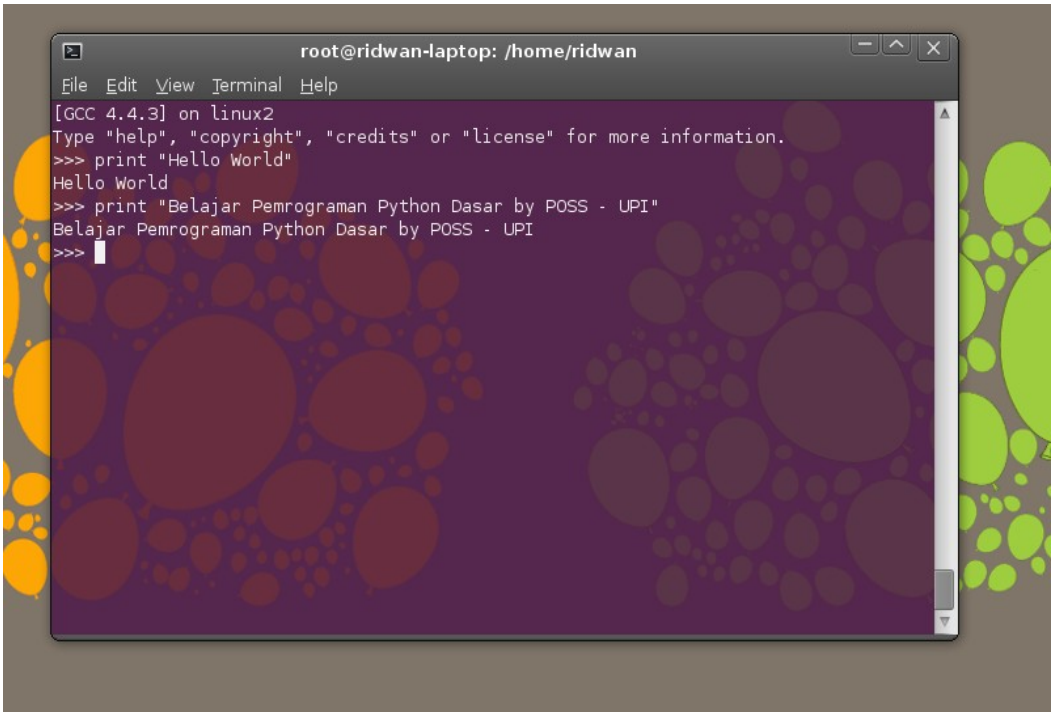
Unduh Python dari http://www.python.org/download. Kemudian unduh *installer* Python untuk Windows. Setelah berhasil mengunduh, klik dua kali *installer* tersebut. Ikuti alurnya sampai selesai. Setelah berhasil biasanya hasil instalasi Python disimpan di direktori C:/PythonX.X dimana X.X adalah versi dari Python yang digunakan.

Kemudian agar Python bisa dieksekusi di cmd, akses **Control Panel → System → Advanced → Environment Variables** → Klik variabel yang dinamakan PATH di bagian System Variables kemudian pilih dan *edit*, tambahkan ;C\PythonX.X tanpa tanda petik. Kemudian tekan OK, dan siap dijalankan di cmd

2. Cara Menggunakan Python

Menggunakan Python Interpreter Prompt dan Teks Editor

Untuk menggunakan Python, kita bisa memilih dua cara yang umum digunakan, yaitu lewat *Python Interpreter Prompt* dan mengeksekusi *file* Python dari *command line*. Anda bisa menggunakan *interpreter prompt* untuk sekedar menulis program kecil, atau menguji coba modul – modul yang dimiliki Python. Untuk menggunakannya jika sistem operasi Anda sudah di-*install* Python, coba panggil perintah **python** melalui *command line*. Dan jika pengaturan *path*-nya sudah sesuai, akan muncul gambar seperti berikut:

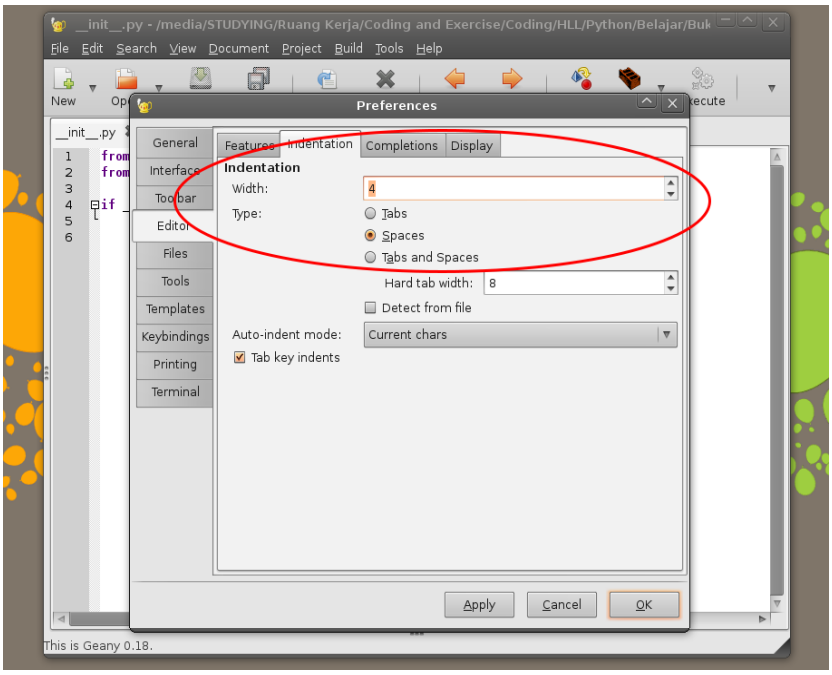


<< gambar 2.1 python interpreter prompt >>

Untuk keluar dari mode *interpreter prompt*, gunakan kombinasi ctrl+d atau ctrl+z kemudian tekan *enter*.

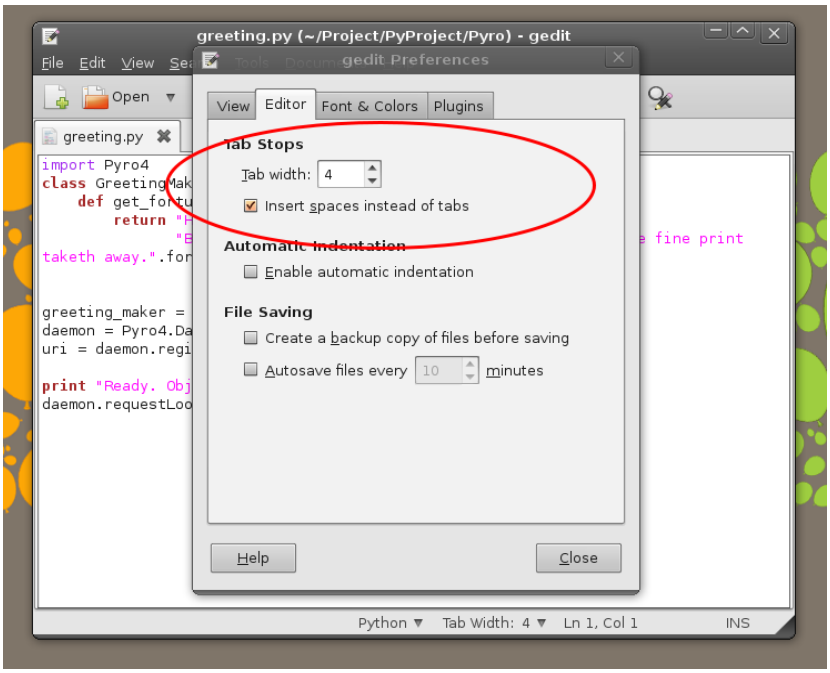
Lalu jika ingin menggunakan Python tanpa melalui *interpreter prompt*, Anda bisa menggunakan *text editor* yang Anda sering gunakan dan simpan *file* Python dengan ekstensi *file* *.py .Kemudian atur jarak tab pada *text editor* tersebut sebanyak empat satuan dan isi tab tersebut diganti dengan spasi atau memilih menu "*replace by space*". Berikut adalah beberapa contoh *text editor* yang bisa digunakan untuk membuat program Python beserta menu untuk pengaturan tab.

Tampilan antarmuka Geany dan tempat pengaturan indentasinya :



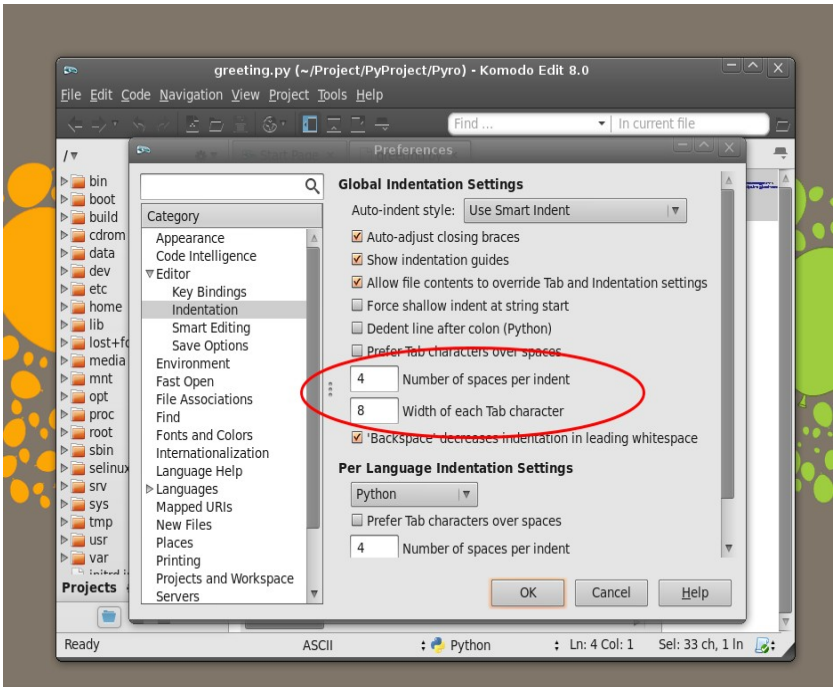
<< gambar 2.2 Geany dan pengaturan tab-nya>>

Tampilan antarmuka GEdit dan tempat pengaturan indentasinya :



<< gambar 2.3 GEdit dan pengaturan tab-nya>>

Tampilan antarmuka Komodo Edit dan tempat pengaturan indentasinya :



<< gambar 2.4 KomodoEdit dan pengaturan tab-nya >>

Jika Anda belum pernah belajar pemrograman Anda bisa gunakan GEdit atau Geany untuk membuat *file* Python. Jika sudah pernah belajar pemrograman, Anda bisa pilih *text editor* manapun sesuai selera.

Mencetak Informasi dengan Function “print”

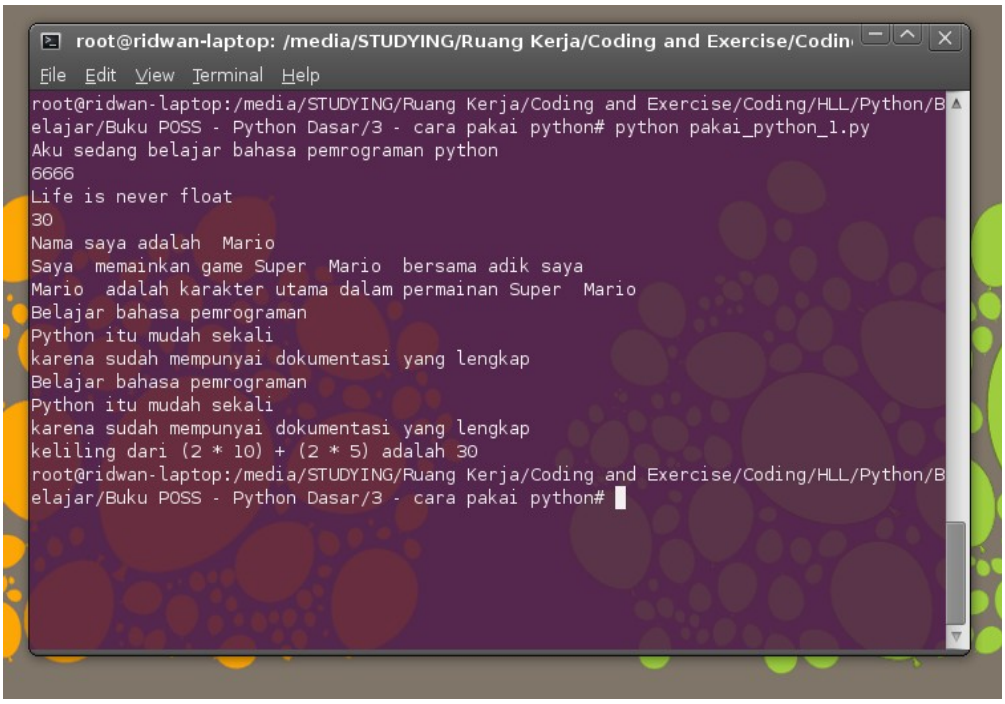
Output digunakan pada program untuk memberikan *feedback* dan keadaan sebuah program, misal hasil perhitungan, pertanyaan, daftar pengguna, dan grafik.. Tapi dalam sebuah program konsol, *output* biasanya berupa teks yang dicetak dengan menggunakan *function* tertentu pada sebuah bahasa pemrograman. Di Python untuk mencetak teks ke layar digunakanlah *function* **print**. *Function* **print** ini akan mencetak *string* yang diinginkan. Ada banyak cara dalam menggunakan **print**, berikut adalah contoh penggunaan **print** dalam sebuah program konsol :

listing : pakai_python_1.py

```
# mencetak sebuah kalimat
```

```
print "Aku sedang belajar bahasa pemrograman python"

# mencetak angka
print 6666

# mencetak variabel
sebuah_variabel = "Life is never float"
print sebuah_variabel

# mencetak langsung sebuah operasi bilangan
panjang = 10
lebar = 5
print (2 * panjang) + (2 * lebar)

# mencetak sebuah variabel dan string dengan menggunakan koma
nama = "Mario"

# dicetak diakhir
print "Nama saya adalah ", nama

# dicetak ditengah
print "Saya  memainkan game Super ", nama, " bersama adik saya"

# dicetak diawal
print nama, " adalah karakter utama dalam permainan Super ", nama

# mencetak banyak baris
print "Belajar bahasa pemrograman"
print "Python itu mudah sekali"
print "karena sudah mempunyai dokumentasi yang lengkap"

# mencetak banyak baris dalam satu kali print
print "Belajar bahasa pemrograman \nPython itu mudah sekali \nkarena sudah mempunyai dokumentasi yang lengkap"

# mencetak variabel pada string dengan format string
panjang = 10
lebar = 5
keliling = (2 * panjang) + (2 * lebar)
print "keliling dari (2 * %d) + (2 * %d) adalah %d " % (panjang, lebar, keliling)
```

Jika kode diatas dieksekusi, akan tampil *output* seperti berikut :



<< gambar 2.5 hasil eksekusi pakai_python_1.py >>

Menerima Masukan Data dengan Function “raw_input” dan “input”

Selain *output* tentu saja dalam membuat sebuah program Anda membutuhkan fitur untuk meminta *input* dari *user* . Fitur tersebut berguna untuk menciptakan interaksi antara *user* dan program yang Anda bangun. Di Python, untuk menerima *input* ada beberapa cara yang biasa digunakan :

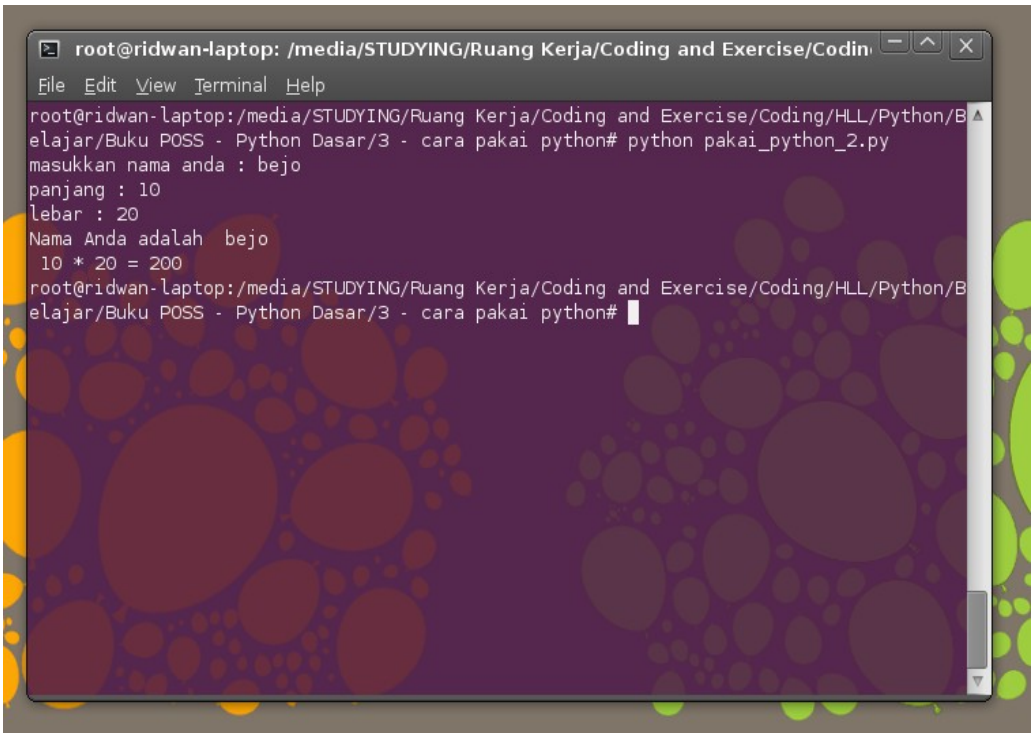
raw_input, *function* ini berguna untuk menerima *input* dari *user* yang akan selalu dikonversi kedalam *string*. Misal Anda memberikan *input* berupa “Belajar Python”. Maka data tersebut akan ditampung sebagai *string* utuh. Kemudian pada **raw_input,** terdapat satu parameter yang akan dijadikan pertanyaan atau perintah tertulis saat meminta *input*. Jika Anda ingin memberikan *input* berupa angka, saat memasukkan angka tersebut tidak boleh lebih dari satu angka. Hal ini disebabkan karena ketika menggunakan **raw_input**, sekalipun yang diberikan adalah angka tetap akan dianggap *string*. Apabila Anda memberikan *input* satu angka kepada **raw_input**, Anda harus mengkonversinya dengan *function* **int, float, long,** atau beberapa *function* konversi ke angka lainnya sesuai dengan kebutuhan yang diinginkan. Coba perhatikan kode dibawah ini :

listing : pakai_python_2.py

```
nama = raw_input('masukkan nama anda : ')
panjang = raw_input("panjang : ")
lebar = raw_input("y : ")
```

```
print "Nama Anda adalah ", nama
luas = int(panjang) * int(lebar)
print " %d * %d = %d" % (int(panjang), int(lebar), luas)
```

Jika kode diatas dieksekusi, maka akan muncul *output* seperti berikut :



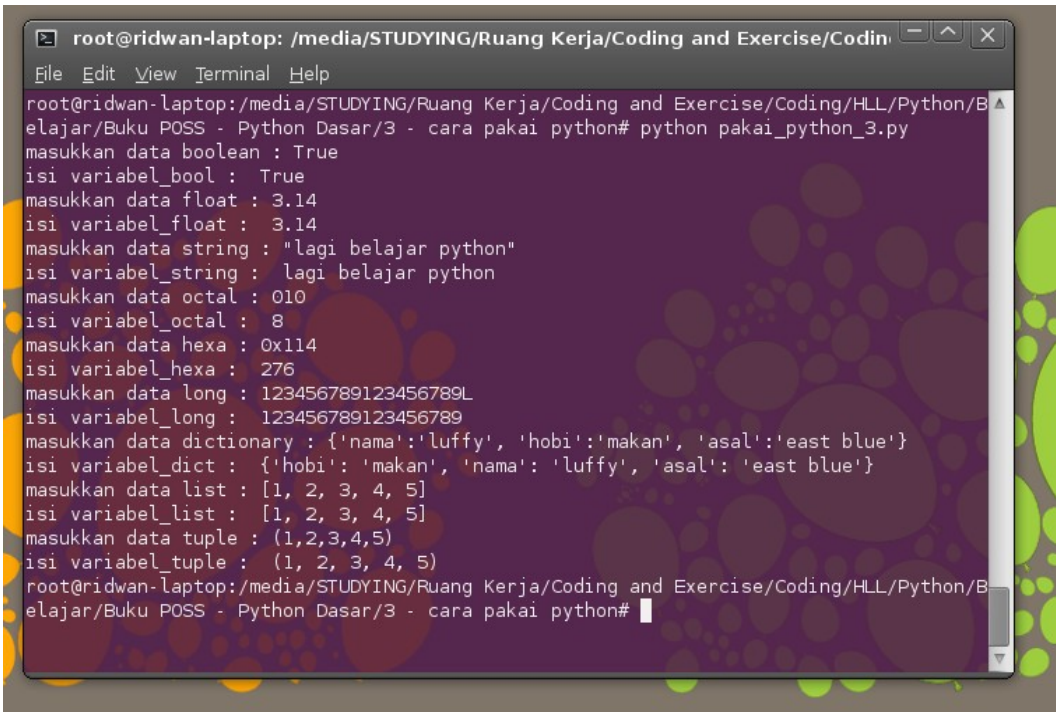
<< gambar 2.6 hasil eksekusi pakai_python_2.py >>

input, function ini digunakan untuk menerima *input* sesuai dengan data yang diberikan oleh *user*. Tidak seperti **raw_input** yang menerima *input* dan dianggap *string*. Saat memberikan *input* kepada **raw_input,** Anda tidak perlu menggunakan aturan penulisan untuk tipe data tertentu. Sedangkan di **input** Anda harus mengikuti aturan penulisan untuk memasukkan *input* dari tipe data tertentu. Sebagai contoh dibawah terdapat beberapa contoh aturan penulisan saat akan memberikan data dengan tipe data tertentu kepada **input**.

listing : pakai_python_3.py

```
# meminta input boolean : coba masukkan True
variabel_bool = input('masukkan data boolean : ')
print "isi variabel_bool : ", variabel_bool

# meminta input float : coba masukkan 3.14
variabel_float = input('masukkan data float : ')
print "isi variabel_float : ", variabel_float

# meminta input string : coba masukkan “lagi belajar python”
variabel_string = input('masukkan data string : ')
print "isi variabel_string : ", variabel_string

# meminta input octal : coba masukkan 010
variabel_octal = input('masukkan data octal : ')
print "isi variabel_octal : ", variabel_octal

# meminta input hexa : coba masukkan 0x114
variabel_hexa = input('masukkan data hexa : ')
print "isi variabel_hexa : ", variabel_hexa

# meminta input long : coba masukkan 123456789123456789L
variabel_long = input('masukkan data long : ')
print "isi variabel_long : ", variabel_long

# meminta input dictionary : coba masukkan {'nama':'luffy', 'hobi':'makan', 'asal':'east blue'}
variabel_dict = input('masukkan data dictionary : ')
print "isi variabel_dict : ", variabel_dict

# meminta input list : coba masukkan [1, 2, 3, 4, 5]
variabel_list = input('masukkan data list : ')
print "isi variabel_list : ", variabel_list


# meminta input tuple : coba masukkan (1, 2, 3, 4, 5)
variabel_tuple = input('masukkan data tuple : ')
print "isi variabel_tuple : ", variabel_tuple
```

Jika kode diatas dieksekusi, akan muncul *output* seperti berikut :



<< gambar 2.7 hasil eksekusi pakai_python_3.py >>

Contoh diatas memperlihatkan sebuah perbedaan penggunaan **raw_input** dengan **input**. Data yang didapat dari **raw_input** harus dikonversikan dengan *built-in function* untuk tipe data tertentu. Sedangkan data yang didapat dari **input** tidak perlu dikonversikan, tapi saat memasukkan data harus mengikuti aturan penulisan untuk tipe data tertentu**.**

Hal Lain yang Harus Diingat dalam Penggunaan Python

Selain itu terdapat beberapa karakter khusus yang dinamakan *escape character*. Berikut adalah daftar beberapa *escape character* yang terdapat di Python :

Escape Character	Heksadesimal	Keterangan
\a	0x07	bel
\b	0x08	*backspace*
\f	0x0c	*formfeed*
\e	0x1b	*escape*
\n	0x0a	*newline*
\t	0x09	*tab*
\v	0x0b	*Vertical tab*
\r	0x0d	*Carriage return*
\nnn		Notasi oktal, dimana n

Escape Character	Heksadesimal	Keterangan
\xnn		merupakan rentang angka dari 0 sampai 7 Notasi heksadesimal, dimana n merupakan rentang dari 0..9, a..f, atau A..F

Pada kode diatas *listing* pakai_python_1.py, terdapat sebuah simbol %d di dalam perintah **print**. Simbol tersebut dinamakan *string formatter* yang berfungsi untuk mencetak data sesuai dengan format yang diinginkan pada *string* yang disisipi simbol tersebut. Berikut adalah daftar beberapa *string formatter* yang disediakan Python:

Simbol	Keterangan
%c	Mencetak karakter
%s	Mencetak data dari jenis apapun menjadi string
%i, %d	Mencetak angka desimal bertanda
%u	Mencetak angka desimal tak bertanda
%o	Mencetak angka oktal
%x, %X	Mencetak angka heksa dengan huruf kecil, Mencetak angka heksa dengan huruf besar
%f	Mencetak angka real berkoma
%e, %E	Mencetak tanda eksponensial dengan huruf kecil, mencetak tanda eksponensial dengan huruf besar
%g, %G	Fungsi hampir sama dengan %f dan %e hanya saja pencetakan angka di belakang koma lebih pendek, pencetakan tanda eksponensial menggunakan huruf besar

Kemudian tak lupa kalau di Python sendiri saat sedang menggunakan *interpreter prompt,*Anda bisa menggunakan *function* **help()** untuk melihat struktur sebuah objek atau perintah – perintah di Python. Misal Anda bisa melihat bantuan tentang perintah **print** maka Anda harus mengetikkan**:**

```
>> help('print')
```

```
The ``print`` statement

**********************


print_stmt ::= "print" ([expression ("," expression)* [","]]
 | ">>" expression [("," expression)+ [","]])

``print`` evaluates each expression in turn and writes the resulting
object to standard output (see below).  If an object is not a string,
it is first converted to a string using the rules for string
conversions.  The (resulting or original) string is then written.  A
space is written before each object is (converted and) written, unless
the output system believes it is positioned at the beginning of a
line.  This is the case (1) when no characters have yet been written
to standard output, (2) when the last character written to standard
output is a whitespace character except ``' '``, or (3) when the last
write operation on standard output was not a ``print`` statement. (In
some cases it may be functional to write an empty string to standard
output for this reason.)

Note: Objects which act like file objects but which are not the built-in
  file objects often do not properly emulate this aspect of the file
  object's behavior, so it is best not to rely on this.

A ``'\n'`` character is written at the end, unless the ``print``
statement ends with a comma.  This is the only action if the statement
contains just the keyword ``print``.

Standard output is defined as the file object named ``stdout`` in the
built-in module ``sys``.  If no such object exists, or if it does not
have a ``write()`` method, a ``RuntimeError`` exception is raised.

``print`` also has an extended form, defined by the second portion of
the syntax described above. This form is sometimes referred to as
"``print`` chevron." In this form, the first expression after the
``>>`` must evaluate to a "file-like" object, specifically an object
that has a ``write()`` method as described above.  With this extended
form, the subsequent expressions are printed to this file object.  If
the first expression evaluates to ``None``, then ``sys.stdout`` is
used as the file for output.

(END)
```

Untuk keluar dari mode bantuan tersebut tekan tombol “q”. Sekarang kita coba lihat bantuan mengenai struktur data list:

```
>> help('list')

Help on class list in module __builtin__:



class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __delitem__(...)
 |      x.__delitem__(y) <==> del x[y]
 |
 |  __delslice__(...)
 |      x.__delslice__(i, j) <==> del x[i:j]
 |
 |      Use of negative indices is not supported.
 |
 |  __eq__(...)
:[]
```

Dengan demikian sekalipun tidak ada koneksi internet, Anda tetap bisa terus membuat program Python dengan dibantu **help()** yang sudah disediakan oleh Python.

Tipe data yang terdapat pada kode – kode diatas akan dijelaskan lebih lanjut pada bab berikutnya.

3. Mengenal Tipe Data dan Operator

Tipe Data di Python

Variabel menyimpan data yang dilakukan selama program dieksekusi dan isinya dapat diubah oleh operasi – operasi tertentu pada program yang menggunakan variabel tersebut.

Di dalam Python, terdapat beberapa tipe data yang cukup unik bila dibandingkan dengan bahasa pemrograman seperti C, Java, dan yang lainnya. Tipe data pada Python adalah sebagai berikut :

- Boolean, contoh True and False
- Complex, pasangan angka real dan imajiner, misalnya 1 + 5j
- Date, bilangan yang dapat dikonversi menjadi format tanggal, misalnya 26-09-2013
- Float, bilangan yang mempunyai koma, misalnya 3.14, 6.387
- Hexadecimal, bilangan dalam format heksa, misalnya 7b, 4d2
- Integer, bilangan bulat, misalnya 10, 20, 30, 15, 37
- Long, bilangan bulat yang panjang, misal 123456789123456789L
- None, data yang tidak terdefinisi tipe data apapun
- String, data yang berisi kalimat. Bisa dibentuk dengan diapit tanda ' dan ', atau diapit “ dan “, atau diapit “”” dan “”” untuk membentuk paragraf.
- List, sebuah data berupa untaian yang menyimpan berbagai tipe data dan isinya bisa diubah. Lebih lengkapnya akan dibahas di bab 6.
- Tuple, sebuah data berupa untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah. Lebih lengkapnya akan dibahas di bab 6.
- Dictionary, sebuah data berupa untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai. Lebih lengkapnya akan dibahas di bab 6.
- Objek, sebuah data yang berisi atribut dan method. Lebih lengkapnya akan dibahas di bab 10

Operator – Operator di Python

Selain variabel diatas, terdapat juga beberapa operator untuk pemrosesan data di Python. Agar lebih memahami seperti apa cara kerja operator dipython, bukalah console **python** dan coba contoh disamping penjelasan tiap operator. Berikut operator yang ada di Python :

1. Aritmatika (Arithmetic Operator)

Operator	Penjelasan	Contoh
+	Penjumlahan, menambahkan dua buah operan	a, b = 10, 5 hasil = a + b

Operator	Penjelasan	Contoh
		# hasil akan 15 print “hasil : “, hasil
-	Pengurangan, mengurangka operan disebelahkiri operator dengan operan di sebelah kanan operator	a, b = 10, 8 hasil = a – b # hasil akan 2 print “hasil : “, hasil
*	Perkalian, mengalikan operan di sebelah kiri dengan operan di sebelah kanan operator	a, b = 3, 5 hasil = a * b # hasil akan 15 print “hasil : “, hasil
/	Pembagian, membagi operan di sebelah kiri dengan operan disebelah kanan operator	a, b = 4, 2 hasil = a / b # hasil akan 2 print “hasil : “, hasil
%	Modulus, mendapatkan sisa pembagian dari operan di sebelah kiri operator ketika dibagi oleh operan di sebelah kanan	a, b = 11, 2 hasil = a % b # hasil akan 1 print “hasil : “, hasil
**	Pemangkatan, memangkatkan operan disebelah kiri operator dengan operan di sebelah kanan operator	a, b = 11, 2 hasil = a ** b # hasil akan 121 print “hasil : “, hasil
//	Pembagian bulat, prosesnya sama seperti pembagian. Hanya saja angka dibelakang koma dihilangkan	a, b = 11, 2 hasil = a // b # hasil akan 5 print “hasil : “, hasil

2. Perbandingan (Comparison Operator)

Operator	Penjelasan	Contoh
==	Memeriksa apakah kedua nilai (operan) yang dibandingkan sama atau tidak. Jika sama akan dikembalikan nilai **True** jika tidak sama akan dikembalikan nilai **False**.	a, b = 10, 10 # hasil akan True print “hasil : “, a == b

Operator	Penjelasan	Contoh
!=	Memeriksa apakah kedua nilai yang dibandingkan sama atau tidak. Jika tidak sama akan dikembalikan nilai **True** jika sama akan dikembalikan nilai **False**.	a, b,= 10, 8 # hasil akan True print “hasil : “, a != b c = 10 # hasil akan False print “hasil : “, a != c
<>	Fungsinya sama dengan operator !=	a, b,= 10, 8 # hasil akan True print “hasil : “, a <> b c = 10 # hasil akan False print “hasil : “, a <> c
>	Memeriksa apakah nilai di sebelah kiri operator lebih besar dari nilai di sebelah kanan operator	a, b = 4, 2 # hasil akan True print “hasil : “, a > b
<	Memeriksa apakah nilai di sebelah kiri operator lebih kecil dari nilai di sebelah kanan operator	a, b = 2, 4 # hasil akan True print “hasil : “, a < b
>=	Memeriksa apakah nilai di sebelah kiri operator lebih besar dari nilai di sebelah kanan operator atau memiliki nilai yang sama	a, b = 4, 2 c = 4 # hasil akan True print “hasil : “, a >= b # hasil akan True print “hasil : “, a >=c # hasil akan False print “hasil : “, b >= a
<=	Memeriksa apakah nilai di sebelah kiri operator lebih kecil dari nilai di sebelah kanan operator atau memiliki nilai yang sama	a, b = 4, 2 c = 4 # hasil akan False print “hasil : “, a <= b # hasil akan True print “hasil : “, a <= c # hasil akan True

Operator	Penjelasan	Contoh
		print “hasil : “, b <= a

3. Penugasan (Assignment Operator)

Operator	Penjelasan	Contoh
=	Mengisikan nilai di sebelah kanan operator ke nilai di sebelah kiri operator	a = 10 # hasil akan 10 print a b = 15 # hasil akan 15 print b
+=	Menambahkan operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 18 sma dgn a = a + b a += b print “hasil : “, a
-=	Mengurangi operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 2 sma dgn a = a - b a -= b print “hasil : “, a
*=	Mengalikan operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 80 sma dgn a = a * b a *= b print “hasil : “, a
/=	Membagi operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 5 # hasil akan 2 sma dgn a = a / b a /= b print “hasil : “, a
%=	Mengambil sisa bagi dari operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 4 # hasil akan 2 sma dgn a = a % b a %= b print “hasil : “, a
**=	Memangkatkan operan sebelah kiri	a, b,= 10, 2

Operator	Penjelasan	Contoh
	operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	# hasil akan 100 sma dgn a = a ** b a **= b print “hasil : “, a
//=	Membagi bulat operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 4 # hasil akan 2 sma dgn a = a // b a //= b print “hasil : “, a

4. Biner (Bitwiser Operator)

Operator	Penjelasan	Contoh
&	Operator biner AND, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika keduanya bernilai 1 maka bit hasil operasi akan bernilai 1	a, b = 13, 37 # a akan bernilai '0000 1101' # b akan bernilai '0010 0101' c = a & b # c akan bernilai 5 = '0000 0101' print c
|	Operator biner OR, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika salah satunya bernilai 1 maka bit hasil operasi akan bernilai 1	a, b = 13, 37 # a akan bernilai '0000 1101' # b akan bernilai '0010 0101' c = a | b # c akan bernilai 45 = '0010 1101' print c
^	Operator biner XOR, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika keduanya bernilai 1 maka bit hasil operasi akan bernilai 0	a, b = 13, 37 # a akan bernilai '0000 1101' # b akan bernilai '0010 0101' c = a ^ b # c akan bernilai 40 = '0010 1000' print c
~	Operator biner Negative, membalik nilai bit. Misal dari 1 menjadi 0, dari 0 menjadi 1	a, b = 13, 37 # a akan bernilai '0000 1101'

Operator	Penjelasan	Contoh
		# b akan bernilai '0010 0101'
<<	Operator penggeser biner ke kiri, deret bit akan digeser ke kiri sebanyak n kali	a, b = 13, 37 # a akan bernilai '0000 1101' # b akan bernilai '0010 0101' # hasil bernilai 52 = "0011 0100' print a << 2 # hasil bernilai 148 = '1001 0100' print b << 2
>>	Operator penggeser biner ke kanan, deret bit akan digeser ke kanan sebanyak satu kali	a, b = 13, 37 # a akan bernilai '0000 1101' # b akan bernilai '0010 0101' # hasil bernilai 3 = '0000 0011' print a >> 2 # hasil bernilai 9 = '0000 1001' print b >> 2

5. Logika (Logical Operator)

Operator	Penjelasan	Contoh
and	Jika kedua operan bernilai True, maka kondisi akan bernilai True. Selain kondisi tadi maka akan bernilai False	a, b = True, True # hasil akan True print a and b
or	Jika salah satu atau kedua operan bernilai True maka kondisi akan bernilai True. Jika keduanya False maka kondisi akan bernilai False	a, b = True, False # hasil akan True print a or b print b or a print a or a # hasil akan False print b or b
not	Membalikkan nilai kebeneran pada operan,	a, b = True, False

Operator	Penjelasan	Contoh
	misal jika asalnya True akan menjadi False dan begitupun sebaliknya	# hasil akan True print not a print not b

6. Keanggotaan (Membership Operator)

Operator	Penjelasan	Contoh
in	Memeriksa apakah nilai yang dicari berada pada list atau struktur data python lainnya. Jika nilai tersebut ada maka kondisi akan bernilai True	sebuah_list = [1, 2, 3,4 ,5] print 5 in sebuah_list
not in	Memeriksa apakah nilai yang dicari tidak ada pada list atau struktur data python lainnya. Jika nilai tersebut tidak ada maka kondisi akan bernilai True	sebuah_list = [1, 2, 3,4 ,5] print 10 not in sebuah_list

7. Identitas (Identity Operator)

Operator	Penjelasan	Contoh
is	Memeriksa apakah nilai di sebelah kiri operan memiliki identitas memori yang sama dengan nilai di sebelah kanan operan. Jika sama maka kondisi bernilai True	a, b = 10, 10 # hasil akan True print a is b
is not	Memeriksa apakah nilai di sebelah kiri operan memiliki identitas memori yang berbeda dengan nilai di sebelah kanan operan. Jika berbeda maka kondisi bernilai True	a, b = 10, 5 # hasil akan True print a is not b

Prioritas Eksekusi Operator di Python

Dari sekian banyaknya operator yang telah disebutkan, masing – masing mempunyai prioritas pemroresan yang dapat dilihat pada tabel berikut. Prioritas tersebut makin ke bawah makin akhir untuk dieksekusi. Paling atas adalah yang akan didahulukan daripada operator lain, sedangkan paling bawah adalah operator yang paling terakhir dieksekusi :

Operator	Keterangan
**	Aritmatika
~, +, -	Bitwise
*, /, %, //	Aritmatika
+, -	Aritmatika
>>, <<	Bitwise
&	Bitwise
^, |	Bitwise
<=, <, >, >=	Perbandingan
<>, ==, !=	Perbandingan
=, %=, /=, //=, -=, +=, *=, **=	Penugasan
is, is not	identitas
in, not in	membership
not, or, and	logika

4. Membuat Pemilihan Kondisi

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

Dalam kehidupan sehari – hari pasti Anda menentukan pilihan untuk memulai sebuah aksi di pagi hari. “Kalau hari ini ga hujan saya akan main tenis”, “Kalau ada ongkos nanti nonton *Man of Steel*”, “Kalau ga hujan dan ada ongkos nanti mau pergi makan ramen”. Disadari atau tidak, pengandaian atau kondisional sudah menjadi bagian dari hidup Anda secara otomatis saat sebelum melakukan sebuah tugas.

Dalam pemrograman pun demikian ada mekanisme dimana program akan menentukan aksi – aksi sesuai kondisi dari input atau nilai – nilai yang diproses selama program berjalan langsung. Pemilihan kondisi ini membutuhkan nilai “True” jika aksi yang diinginkan dibawah kondisi tersebut dieksekusi. Jika nilainya “False”, maka akan diperiksa kondisi lain yang sesuai atau akan langsung ke bagian program yang tidak memeriksa kondisi.

Di Python, terdapat beberapa *keyword* untuk membuat sebuah pemilihan kondisi. Ada **if**, **elif**, **else.** Tidak memerlukan kurawal atau penutup pada blok **if** tersebut. Sebuah *statement* akan dianggap blok **if** jika indentasinya lebih satu tab dari jumlah tab **if** diatasnya. Sebuah **if** akan diawali tanda titik dua baru dibawahnya terdapat kode program yang akan dieksekusi jika kondisi terpenuhi.

Dalam membuat pemilihan kondisi Anda juga membutuhkan operator logika (and, not, or) dan perbandingan (==, <=, >=, >, <, <>, !=) untuk menyusun kondisi yang Anda butuhkan.

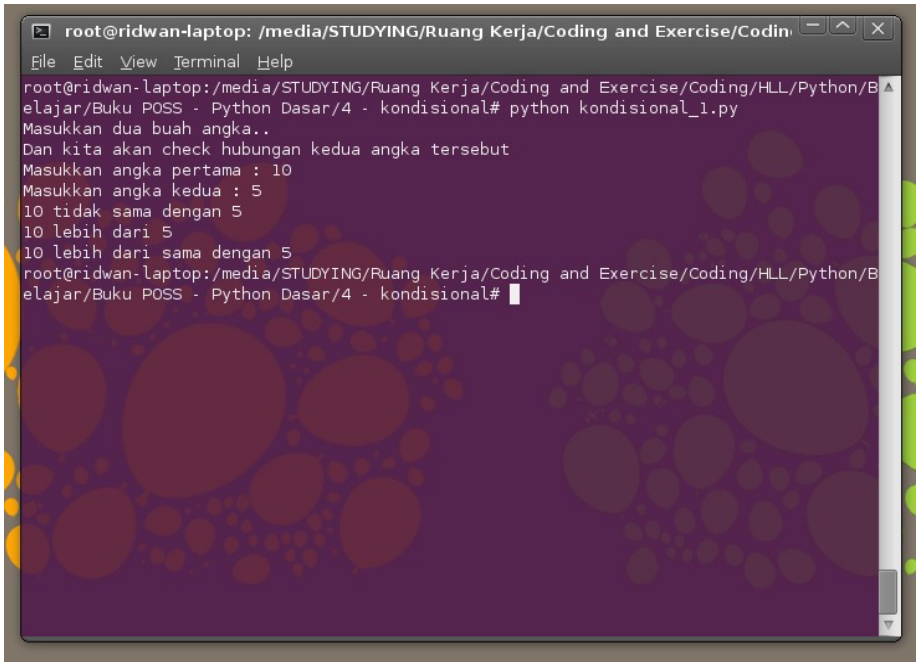
Berikut adalah contoh penggunaan **if** di Python. Contoh berikut menggunakan beberapa operator perbandingan untuk melihat hasil perbandingan dua buah angka. Dalam program berikut beberapa kondisi yang terpenuhi akan dieksekusi.

listing : kondisional_1.py

```
print "Masukkan dua buah angka.."
print "Dan Anda akan check hubungan kedua angka tersebut"


angka1 = raw_input("Masukkan angka pertama : ")
angka1 = int(angka1)

angka2 = raw_input("Masukkan angka kedua : ")
angka2 = int(angka2)


if angka1 == angka2 :
    print "%d sama dengan %d" % (angka1, angka2)
if angka1 != angka2 :
    print "%d tidak sama dengan %d" % (angka1, angka2)
```

```
if angka1 < angka2 :
    print "%d kurang dari %d" % (angka1, angka2)
if angka1 > angka2 :
    print "%d lebih dari %d" % (angka1, angka2)
if angka1 <= angka2 :
    print "%d kurang dari sama dengan %d" % (angka1, angka2)
if angka1 >= angka2 :
    print "%d lebih dari sama dengan %d" % (angka1, angka2)
```

Cobalah berbagai angka sebagai test case dan amati hasilnya. Misal Anda masukkan angka 10 dan 5 Maka hasilnya akan terdapat beberapa kondisi yang dieksekusi :



<< gambar 4.1 hasil eksekusi kondisional_1.py >>

Penggunaan “else” pada “if”

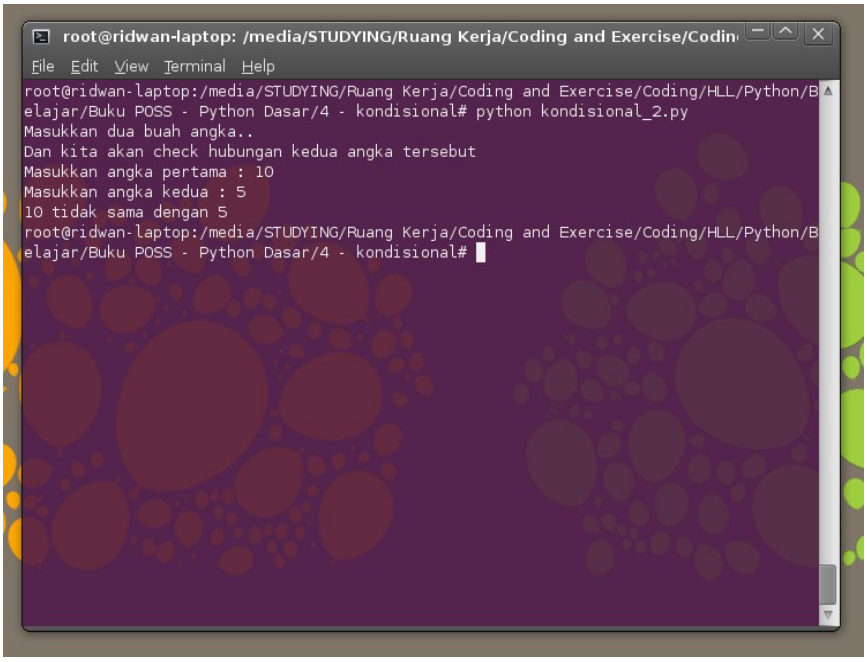
Keyword **else** digunakan dalam blok **if** untuk menampung berbagai kondisi yang berlawanan dengan kondisi pada **if** sebelumnya. *Keyword* **else** ini membutuhkan blok **if** atau **elif** diatasnya. Tanpa kedua *keyword* tadi, **else** tidak dapat digunakan. Berikut ini terdapat contoh penggunaan **else**, mari kita coba.

listing : kondisional_2.py

```
```

```
print "Masukkan dua buah angka.."
print "Dan Anda akan check hubungan kedua angka tersebut"


angka1 = raw_input("Masukkan angka pertama : ")
angka1 = int(angka1)

angka2 = raw_input("Masukkan angka kedua : ")
angka2 = int(angka2)


if angka1 == angka2 :
        print "%d sama dengan %d" % (angka1, angka2)
else:
        print "%d tidak sama dengan %d" % (angka1, angka2)
```

cobalah masukkan dua angka berbeda dan amati hasilnya. Misalkan Anda memasukkan angka 10 dan 5 maka akan tampil hasil seperti berikut :



<< gambar 4.2 hasil eksekusi kondisional_2.py >>

Penggunaan “elif” pada “if”

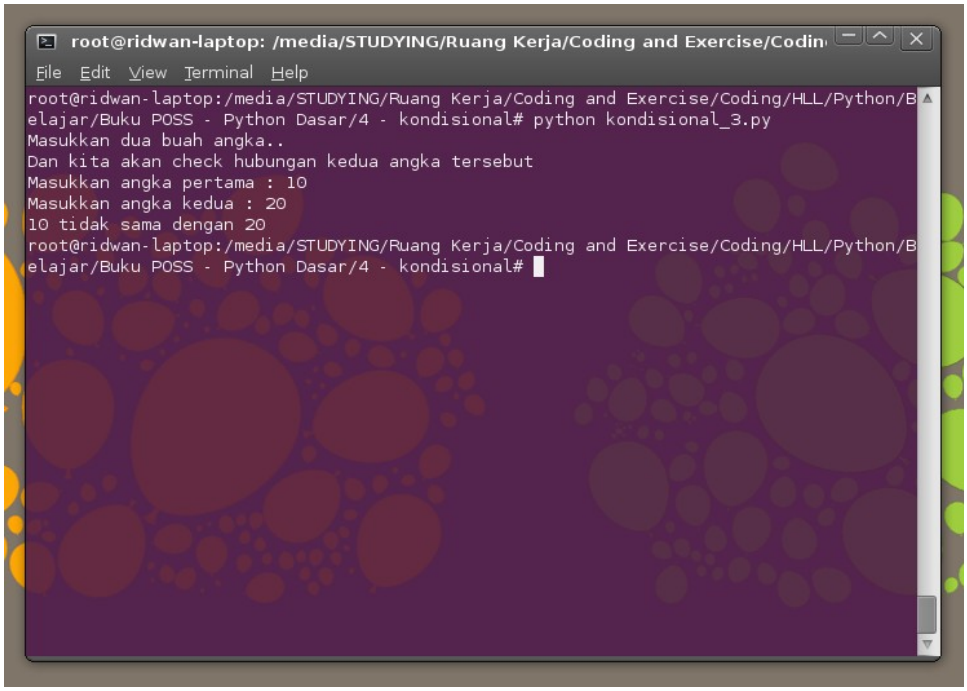
Jika pada kondisional_1.py beberapa blok **if** akan dieksekusi, karena tidak ada pilihan lain pada masing – masing blok **if**. Pada contoh berikutnya beberapa **if** akan digabung dan membentuk sebuah blok **if** yang lebih besar karena adanya **elif**. *Keyword* **elif** ini berfungsi untuk membuat multi kondisional. Jadi jika kondisi di if paling atas tidak sesuai maka kondisi yang ada dibawahnya akan diperiksa dan jika cocok akan dieksekusi. Pada contoh berikutnya jika kondisi sudah sesuai

pada blok teratas maka blok tersebutlah yang akan dieksekusi, berbeda dengan contoh pada kondisional_1.py karena terdiri dari beberapa blok **if** yang dianggap berbeda oleh Python. Untuk lebih jelasnya mari coba kode berikut

listing : kondisional_3.py

```
print "Masukkan dua buah angka.."
print "Dan Anda akan check hubungan kedua angka tersebut"


angka1 = raw_input("Masukkan angka pertama : ")
angka1 = int(angka1)

angka2 = raw_input("Masukkan angka kedua : ")
angka2 = int(angka2)


if angka1 == angka2 :
        print "%d sama dengan %d" % (angka1, angka2)
elif angka1 != angka2 :
        print "%d tidak sama dengan %d" % (angka1, angka2)
elif angka1 < angka2 :
        print "%d kurang dari %d" % (angka1, angka2)
elif angka1 > angka2 :
        print "%d lebih dari %d" % (angka1, angka2)
elif angka1 <= angka2 :
        print "%d kurang dari sama dengan %d" % (angka1, angka2)
elif angka1 >= angka2 :
        print "%d lebih dari sama dengan %d" % (angka1, angka2)
```

Coba masukkan dengan angka 10 dan 20, maka blok if yang dieksekusi hanya blok kedua yang berisi kondisi angka1 tidak sama dengan angka 2. Jelas berbeda dengan kode yang ada di kondisional_1.py. Untuk lebih jelasnya coba perhatikan gambar berikut



<< gambar 4.3 hasil eksekusi kondisional_3.py >>

Penggunaan “else” pada “if”

Misal ada sebuah kondisi seperti berikut, “Kalau saya punya uang saya akan pergi ke taman bermain, Lalu kalau uangnya cuma 10.000 cuma bakal naik komedi putar, kalau uangnya 20.000 bakal naik komedi putar dan bom bom car”. Jika Anda perhatikan setelah kondisi pertama ada kondisi lagi yang masih berada dibawah kondisi pertama. Kondisi semacam ini dapat disebut dengan kondisi bersarang (nested if).

Di Python, untuk membuat sebuah blok **if** di dalam **if,** maka blok **if** yang ingin disimpan di dalam sebuah **if** harus mempunyai satu tab lebih dibanding **if** sebelumnya. Anda dapat membuat if bersarang di dalam if bersarang hingga tingkat sedalam yang Anda inginkan.

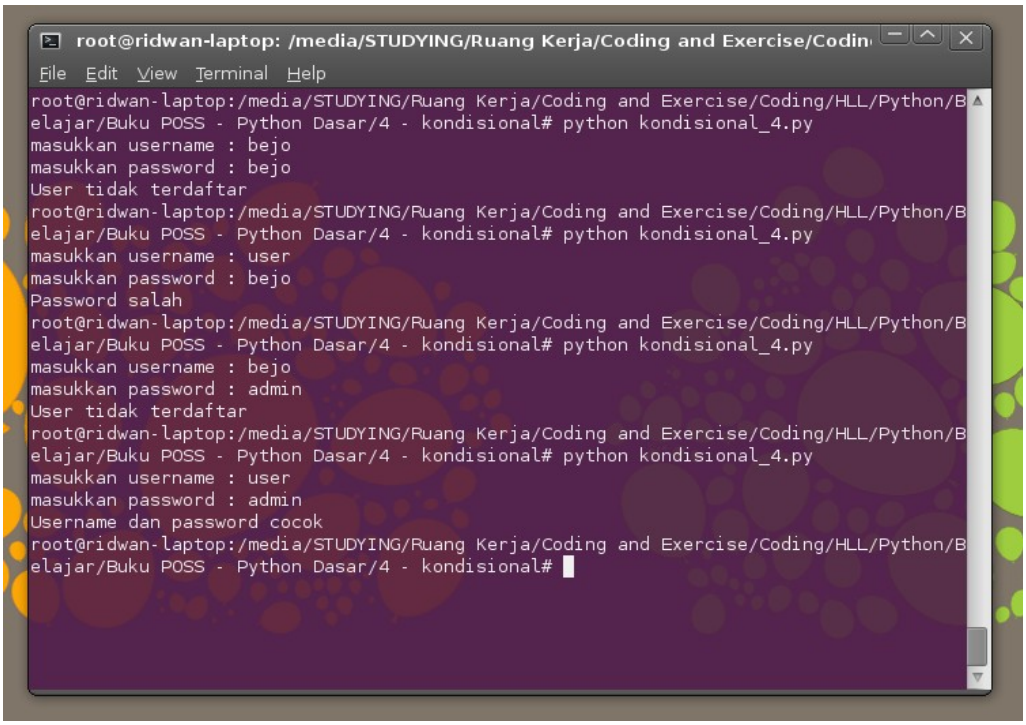
Agar lebih paham mari Anda coba kode berikut :

listing : kondisional_4.py

```
username = raw_input("masukkan username : ")
password = raw_input("masukkan password : ")

username_from_db = "user"
password_from_db = "admin"

if username == username_from_db :
    if password == password_from_db :
```

```
      print "Username dan password cocok "
   else:
      print "Password salah "
else:
   print "User tidak terdaftar"
```

Pada contoh diatas, Anda diminta masukan berupa “username” dan “password”. Kemudian ada sebuah variabel yang diasumsikan mengambil data “username” dan “password” dari *database*. Blok i**f** akan memeriksa apakah user sudah sesuai atau belum, jika tidak sesuai maka akan ditampilkan “User tidak terdaftar”. Jika “username” sesuai maka kondisi selanjutnya adalah memeriksa “password” jika sesuai maka akan muncul notifikasi “Username dan password cocok”, jika tidak sesuai maka akan muncul notifikasi “Password salah”. Lebih jelasnya perhatikan gambar berikut :



<< gambar 4.4 hasil eksekusi kondisional_4.py >>

5. Menyusun Pengulangan

Mengenal Pengulangan “for” dan “while”

Seperti pada bahasa pemrograman lainnya, Python memiliki mekanisme pengulangan untuk menjalankan pekerjaan – pekerjaan yang berulang. Pada umumnya pengulangan terdiri dua jenis. Ada pengulangan yang terbatas dan tidak terbatas. Pengulangan terbatas biasanya dilakukan pengulangan dari 1 hingga kesekian kali (n kali). Perulangan terbatas menggunakan **for.** Sedangkan pengulangan tidak terbatas menggunakan **while**. Dalam pengulangan menggunakan **while** pengulangan akan terus dilakukan selama kondisional dalam perulangan while tetap dalam keadaan **true** jika dalam keadaan **false** maka pengulangan **while** akan berhenti.

Menyusun Pengulangan dengan “for”

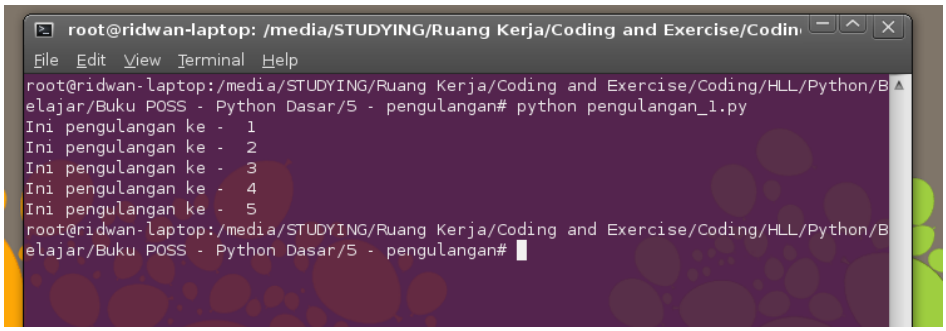
Kita melangkah ke pengulangan yang terbatas dulu yah. Dalam pengulangan **for**, tidak seperti di bahasa pemorgraman C atau Java yang menggunakan nilai *incremental* untuk melakukan pengulangan. Di Python, **for** melakukan pengulangan dengan meng-iterasi elemen dari sebuah *list*. *List* ini dapat berisi kumpulan karakter, kumpulan *string*, kumpulan angka, atau kumpulan data jenis lainnya yang disediakan Python. (Untuk lebih lengkapnya di bab berikutnya akan dibahas lebih jauh tentang *List* di Python).

Misal disini ada sebuah *list* yang berisi [1, 2, 3, 4, 5], (sebuah list diawali oleh tanda '[' dan ditutup oleh tanda ']'). Banyaknya elemen pada *list* tersebut menentukan banyaknya pengulangan yang akan dilakukan saat melakukan pengulangan. Mari kita lihat implementasinya pada kode dibawah ini :

listing : kode pengulangan_1.py

```
# pengulangan sebanyak 5 kali
for i in [1, 2, 3, 4, 5]:
   print "Ini pengulangan ke - ", i
```

Pada contoh diatas, akan dicetak teks “ini pengulangan ke - “ sebanyak 5 kali. Nilai 'i' pada pengulangan tersebut akan selalu berganti nilainya setiap tahap pengulangan dilakukan. Misal ketika pengulangan pertama, nilai 'i' akan berisi 1, ketika pengulangan kedua, nilai 'i' akan berisi 2, begitu seterusnya sampai elemen terakhir. Jika kode diatas dieksekusi akan menampilkan hasil seperti pada gambar dibawah ini :



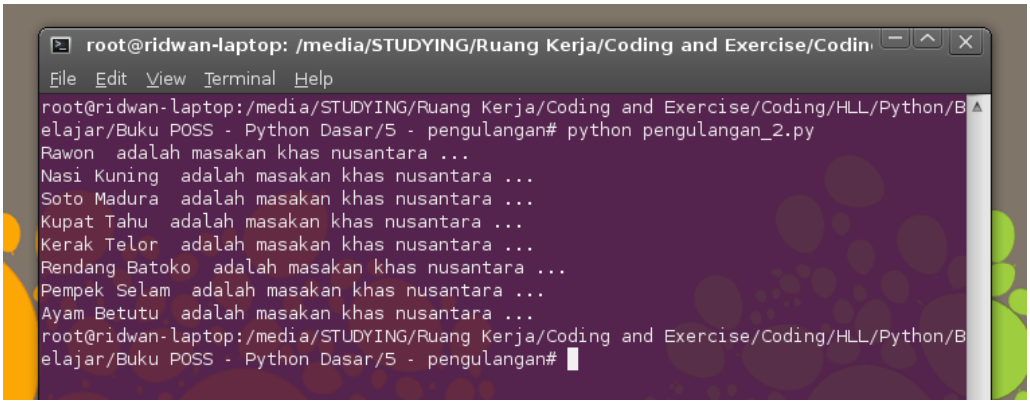
<< gambar 5.1 hasil eksekusi pengulangan_1.py >>

Selain menggunakan *list* yang berisi angka, *List* yang berisi *string* dapat digunakan juga untuk melakukan pengulangan **for** di Python. Misal terdapat *list* yang berisi seperti berikut [“Rawon”, “Nasi Kuning”, “Soto Madura”, “Kupat Tahu”, “Kerak Telor”, “Rendang Batoko”, “Pempek Selam”, “Ayam Betutu”], dalam *list* tersebut terdapat elemen sebanyak delapan jenis masakan nusantara. Dengan demikian ketika pengulangan **for** menggunakan *list* masakan tadi, pengulangan akan dijalankan sebanyak delapan kali. Mari Anda lihat implementasinya pada kode dibawah ini :

listing : kode pengulangan_2.py

```
# pengulangan sebanyak 8 kali
for i in ["Rawon", "Nasi Kuning", "Soto Madura", "Kupat Tahu", "Kerak Telor", "Rendang Batoko", "Pempek Selam", "Ayam Betutu"] :
    print i, " adalah masakan khas nusantara …"
```

Kode diatas jika dieksekusi akan terlihat seperti gambar dibawah ini :



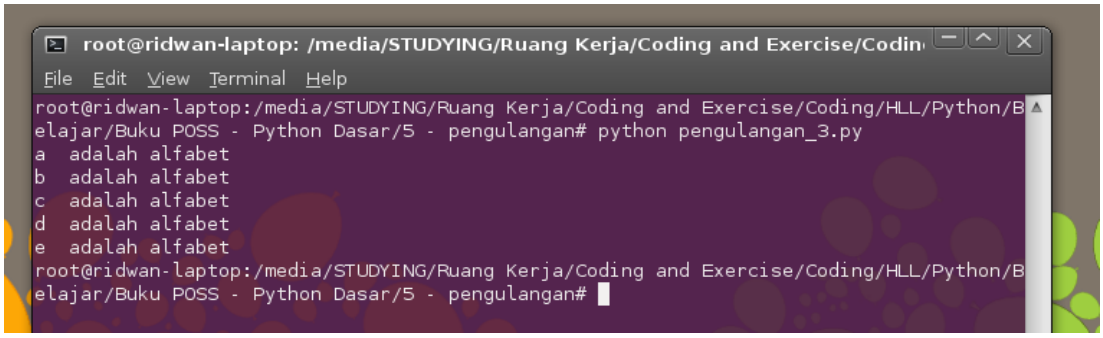
<< gambar 5.2 hasil eksekusi pengulangan_2.py >>

Sekarang Anda coba contoh terakhir dengan menggunakan *string. String* pada dasarnya merupakan *list* karakter. Misal terdapat *string* seperti berikut "abcde". Jika *string* tersebut digunakan pada pengulangan **for**, maka akan terjadi pengulangan sebanyak lima kali. Coba lihat kode dibawah ini :

listing : kode pengulangan_3.py

```
# pengulangan sebanyak 5 kali
for i in "abcde":
    print i, " adalah alfabet"
```

Kode diatas jika dieksekusi akan terlihat seperti gambar dibawah ini :



<< gambar 5.3 hasil eksekusi pengulangan_3.py >>

Memahami Function "range"

Kalau teman – teman memperhatikan *list* yang dipakai pada pengulangan_1.py, pengulangan tersebut menggunakan *list* angka yang sudah jadi atau di-*hardcode*. Nah bagaimana nih kalau ingin membentuk suatu pola atau ingin membuat list *incremental* agar pengulangan **for** di Python ini mirip di Java atau C. Di Python terdapat fungsi yang bernama **range. Range** ini menghasilkan deret angka dengan parameter (*start*, *stop, step*). *Start* adalah batasawal dari *list, stop* adalah batas akhir dari *list, step* adalah jarak antar angka yang dihasilkan oleh **range.** Ada beberapa kasus penting yang perlu diperhatikan saat menggunakan **range**. Coba perhatikan kode dibawah ini :

listing : kode pengulangan_4.py

```
# pengulangan_4.py

# kasus - 1 : jika step tidak disertakan maka step akan diisi 1 secara default
print range(1, 10)

# kasus - 2 : jika step disertakan maka step akan sesuai dengan angka yang diisikan
```

```
print range(1, 10, 2)
print range(1, 10, 3)
print range(1, 10, 4)
print range(1, 10, 5)

# kasus - 3 : jika step melebihi stop maka list hanya akan berisi start
print range(1, 10, 11)

# kasus - 4 : jika start lebih besar nilainya daripada stop maka list akan kosong
print range(10, 1)

# kasus - 5 : jika start lebih besar nilainya daripada stop dan
# jika step melebihi stop maka list akan kosong
print range(10, 1, 2)
print range(10, 1, 11)

# kasus - 6 : jika start lebih besar daripada stop dan step bernilai minus
# dan jika start dikurangi step menghasilkan angka positif
# maka list akan berisi deret angka menurun
print range(10, 1, -1)

# kasus - 7 : jika start lebih besar daripada stop dan step bernilai minus
# dan jika start dikurangi step bernilai minus maka list hanya akan berisi start
print range(10, 1, -11)

# kasus - 8 : jika step bernilai 0 maka akan terjadi error
print range(1, 10, 0)

# kasus - 9 : jika start lebih besar daripada stop dan step bernilai 0 maka akan terjadi error
print range(10, 1, 0)
```

Kode diatas jika dieksekusi akan terlihat seperti gambar dibawah ini :

<< gambar 5.4 hasil eksekusi pengulangan_4.py >>

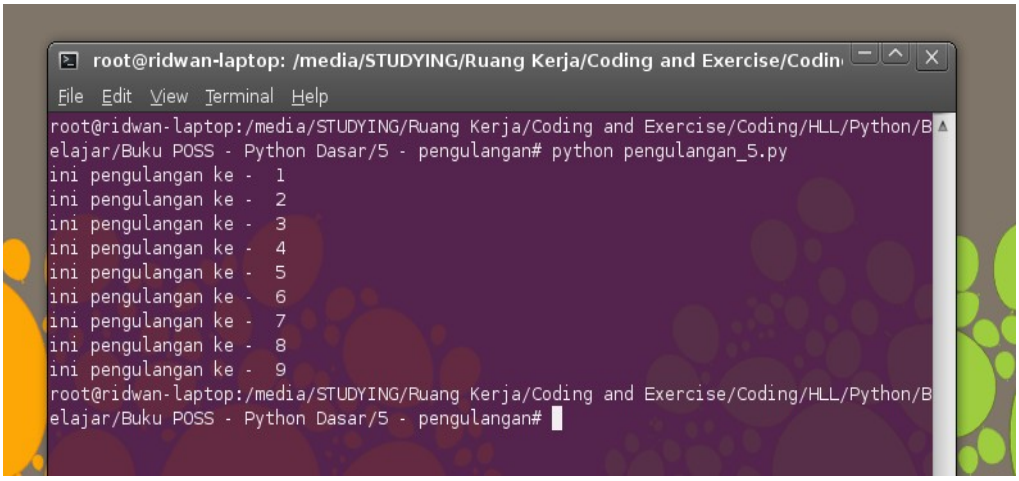
Menggunakan Function “range” pada Pengulangan “for”

Bagaimana dengan pengenalan **range** diatas ? Mudah kan ? Nah sekarang Anda akan coba menggunakan range dalam pengulangan **for.** Coba perhatikan contoh berikut :

listing : kode pengulangan_5.py

```
for i in range(1, 10):
   print "ini pengulangan ke - ", I
```

Pada contoh diatas akan terjadi pengulangan sebanyak 10 kali terhadap *statement* dibawah **for.** Dengan menggunakan **range,** Anda tidak perlu repot untuk membuat **list** terlebih dahulu untuk menentukan banyaknya pengulangan yang akan Anda lakukan terhadap *statement.*Jika kode diatas dieksekusi akan terlihat seperti gambar dibawah ini :



<< gambar 5.5 hasil eksekusi pengulangan_5.py >>

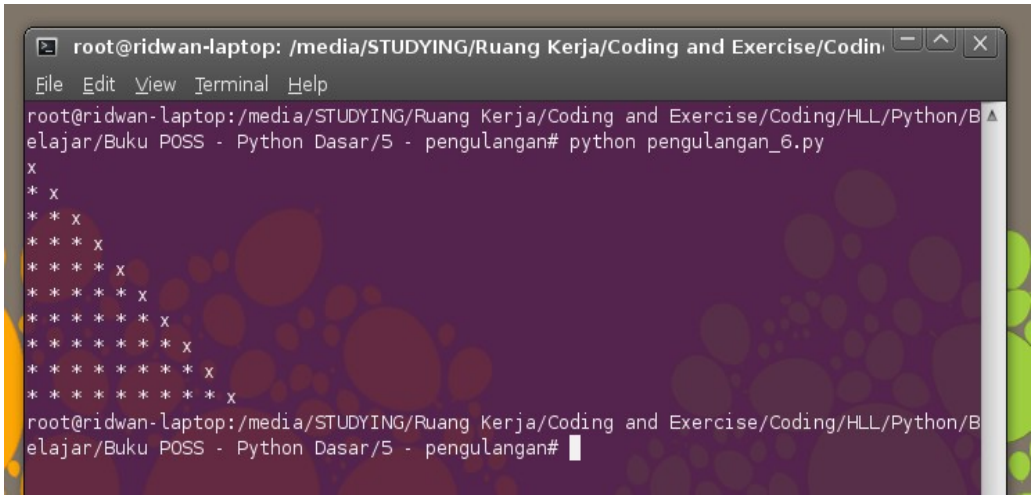
Agar lebih memahami lagi pengulangan **for,** kita coba lagi pelajari dua contoh berikut. Berikut ada kasus membuat sebuah segitiga yang dibuat dari kumpulan bintang dan membuat baris bilangan prima. Untuk lebih jelasnya coba perhatikan dua kasus berikut :

listing : kode pengulangan_6.py

```
# pengulangan_6.py
for i in range(0, 10):
   for j in range (0, i+1):
      if j == i:
         print "x"
```

```
    else:
        print "*",
print ""
```

Kode diatas jika dieksekusi akan tampil seperti berikut :



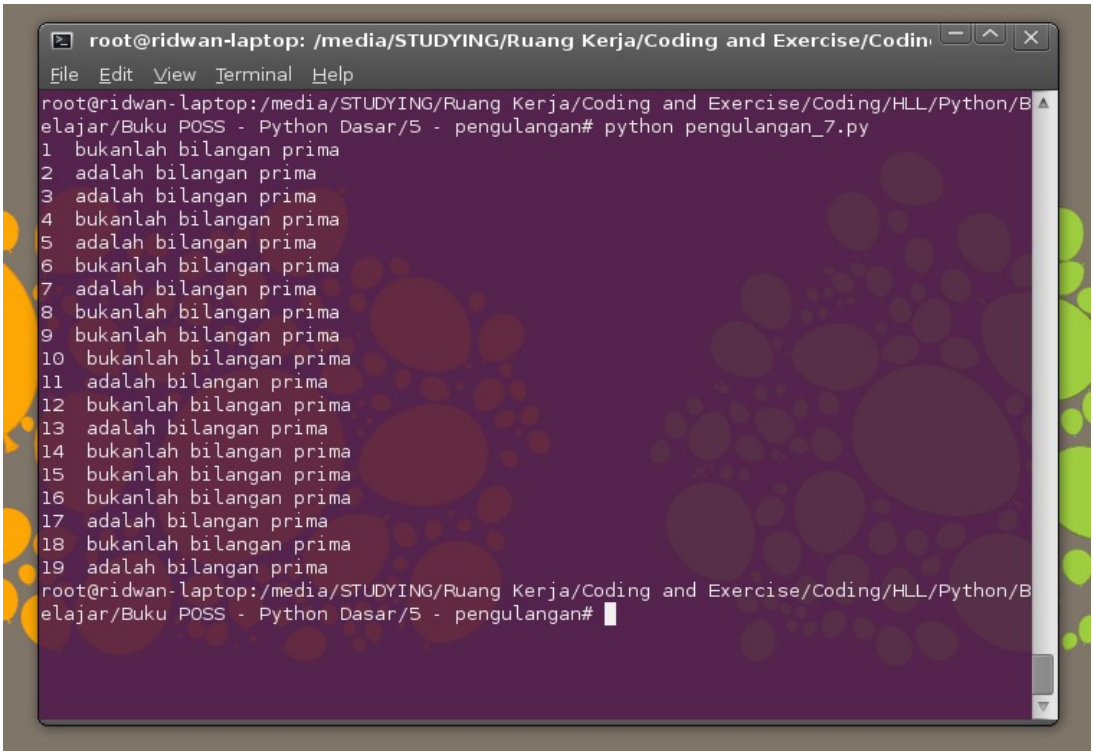
<< gambar 5.6 hasil eksekusi pengulangan_6.py >>

Kemudian dibawah ini adalah kode program untuk mencari bilangan prima. Bilangan prima adalah bilangan yang hanya bisa dibagi 1 dan bilangan itu sendiri

listing : kode pengulangan_7.py

```
# pengulangan_7.py
for i in range(1, 20):
    count_zero_remainder = 0
    for j in range(1, i+1):
        num_remainder = i % j
        #print num_remainder,
        if num_remainder == 0:
            count_zero_remainder = count_zero_remainder + 1

    if count_zero_remainder == 2:
        print i, " adalah bilangan prima"
    else:
        print i, " bukanlah bilangan prima"
```

Kode diatas jika dieksekusi akan tampil seperti berikut :



<< gambar 5.7 hasil eksekusi pengulangan_7.py >>

Menyusun Pengulangan dengan “while”

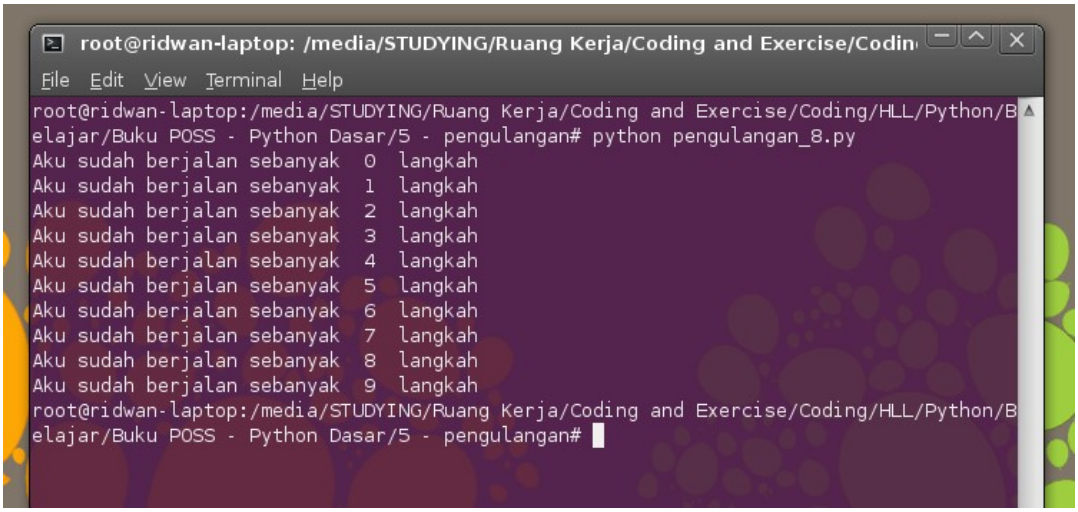
Sekarang kita akan bahas pengulangan yang menggunakan **while.** Pengulangan **while** memiliki cara kerja selama kondisi tertentu bernilai **true** maka pengulangan akan diteruskan sampai kondisi bernilai **false**. Tentunya dalam kondisi yang dipakai untuk eksekusi **while** memerlukan operator logika dan perbandingan seperti yang sudah di jelaskan di bab 3.

sebagai contoh coba lihat kode berikut. Kode berikut dieksekusi apabila variabel “angka” masih dibawah 10.

listing : kode pengulangan_8.py

```
angka =  0
while (angka < 10):
   print "Aku sudah berjalan sebanyak ", angka, " langkah "
   angka += 1
```

Kode diatas jika dieksekusi akan tampil seperti pada gambar berikut ini :

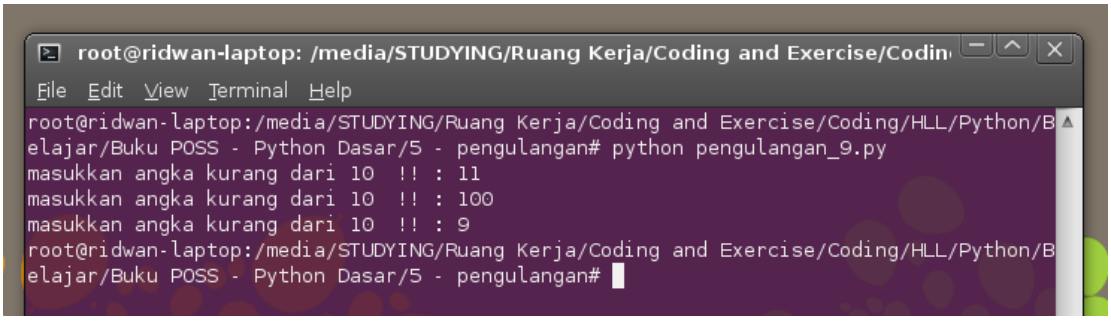


<< gambar 5.8 hasil eksekusi pengulangan_8.py >>

Pada contoh diatas kondisi untuk melakukan pengulangan ditaruh di **while**. Sekarang Anda coba taruh kondisi pengulangan di dalam pengulangannya. Coba lihat contoh berikut :

listing : kode pengulangan_9.py

```
terus_tanya = True
while terus_tanya :
    temp = raw_input('masukkan angka kurang dari 10  !! : ')
    angka = int(temp)
    if angka < 10:
        terus_tanya = False
    else:
        terus_tanya = True
```



<< gambar 5.9 hasil eksekusi pengulangan_9.py >>

Untuk memahami pengulangan **while** lebih lanjut, berikut terdapat contoh penjumlahan angka dari 1 sampai 10. Dalam pengulangan ini terdapat sebuah variabel *jml_angka* yang berfungsi untuk menampung angka – angka yang akan ditambahkan dengan angka berikutnya di setiap

pengulangan. Coba perhatikan kode dibawah ini :

listing : kode pengulangan_10.py

```
i = 1

jml_angka = 0

while i <= 10:
    print 'loop ke - %d : %d + %d' % (i, jml_angka, i)
    jml_angka = jml_angka + i
    i += 1

print 'total angka yang dijumlahkan : ', jml_angka
```

kode diatas jika dieksekusi akan tampi seperti pada gambar berikut ini :

<< gambar 5.10 hasil eksekusi pengulangan_10.py >>

6. Mengenal Data Struktur Python Tingkat Lanjut

Mengenal List, Dictionary dan Tuple

Data yang kompleks biasanya dapat menampung objek atau entitas yang banyak. Misal ada sebuah toples yang berisi banyak permen dengan ragam jenisnya. Dalam sebuah toples dapat saja ada permen rasa asem, permen rasa stroberi, permen rasa nanas, permen rasa asem lagi, dan rasa lainnya. Tapi bukan hanya menyimpan entitas yang banyak saja, struktur data pun ada yang menyimpan jaringan sebuah graf, antrian, tumpukan, dan banyak lagi tipe lainnya.

Di Python sendiri, terdapat beberapa tipe struktur data yang dapat digunakan oleh penggunanya. Tapi tiga tipe data yang umum digunakan, diantaranya **list**, **tuple**, dan **dictionary**. Ketiga tipe data struktur ini sangat membantu sekali bagi programmer Python yang membutuhkan tipe data banyak. Ketiga tipe data ini adalah objek Python yang berarti jika Anda mendefinisikan struktur data tersebut, Anda dapat menggunakan *method – method* yang berhubungan dengan pengolahan struktur data tersebut. Selain itu terdapat pula function untuk mengolah tiga struktur data tadi seperti mencari nilai **max**, **min**, hitung panjang, dan perbandingan isi.

Untuk mendefinisikan sebuah list Anda cukup buat sebuah variabel dengan nama yang diinginkan, kemudian isi variabel tersebut dengan list yang akan dibuat. List diawali dengan tanda '[' dan diakhiri dengan tanda ']'. Isinya dapat beragam, dapat string, number, object, bahkan list lagi. Pemisah antara data yang satu dengan yang lainnya digunakan tanda koma. List dapat ditambah isinya, dirubah data pada elemennya, hapus elemen, dan hapus seluruh list.

Hampir sama dengan list, tuple diawali dengan tanda '(' dan ')'. Elemen pada tuple tidak dapat dihapus dan diubah. Maka dari itu tuple disebut immutable sequence. Lebih jelasnya nanti Anda akan lihat perbedaan tuple dengan list.

Beda halnya antara dengan list dan tuple, pada dictionary setiap data akan memiliki pengenalnya masing – masing. Pengenal tersebut dinamakan dengan key dan datanya dinamakan dengan value. Dictionary diawali dengan tanda '{' dan diakhiri dengan tanda '}'. Khusus untuk key pada dictionary, nilainya harus berupa tipe data yang tidak dapat diganti seperti tuple, string dan number. Tapi umumnya key berisi number dan string. Karena jika Anda mencoba memasukkan tipe data yang mutable, akan keluar peringatan 'unhashable type' saat mendefinisikan dictionary yang key-nya berupa tipe data mutable.

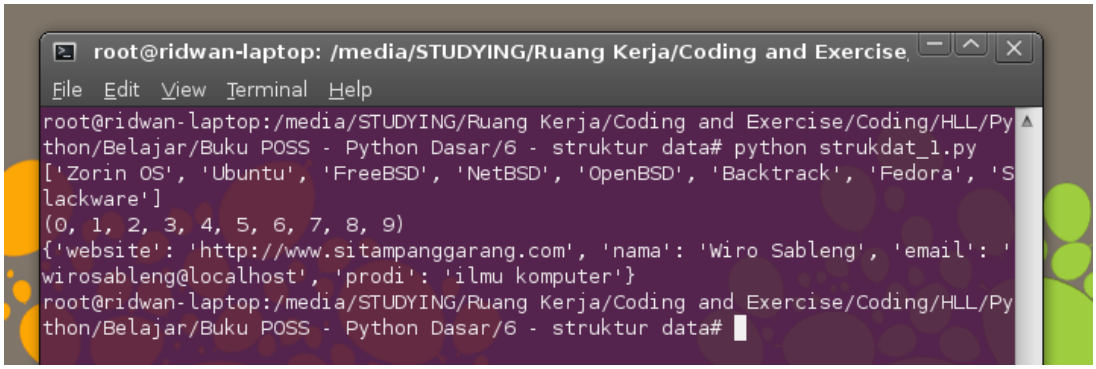
Agar lebih paham mari Anda coba mendefinisikan list, tuple, dan dictionary kedalam sebuah variabel.

listing : strukdat_1.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
```

```
            'Ubuntu',
            'FreeBSD',
            'NetBSD',
            'OpenBSD',
            'Backtrack',
            'Fedora',
            'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                'prodi':'ilmu komputer',
                'email':'wirosableng@localhost',
                'website':'http://www.sitampanggarang.com'
                }


print sebuah_list
print sebuah_tuple
print sebuah_dictionary
```

Dengan menggunakan perintah **print** maka Anda dapat mencetak isi **list**, **tuple**, atau **dictionary** yang hasil keluarannya berupa string.



<< gambar 6.1 hasil eksekusi strukdat_1.py >>

Cara Akses List, Tuple, dan Dictionary

Setelah Anda mengenal cara membuat ketiga struktur data tersebut, sekarang Anda coba cara mengakses elemen – elemen pada struktur data tersebut. Ada beberapa cara mengakses elemen pada struktur data tersebut yang biasa dilakukan misalnya mengakses salah satu elemen dengan

menggunakan indeks, *slicing* indeks, dan mengakses elemen lewat pengulangan.

Untuk akses elemen lewat indeks elemen Anda panggil nama list kemudian disusul indeks elemen yang Anda inginkan dengan diapit tanda baca "[" dan "]", misal ada sebuah list dengan nama **daftar_barang** kemudian ingin mengakses indeks ke – 10, maka pemanggilan indeks pada list tersebut adalah **daftar_barang[9].** Kenapa di pemanggilan indeks nya dari 9 ? karena indeks tersebut diawali dari 0 sehingga indeks yang diinginkan akan dikurangi 1. Begitupun dengan tuple cara akses salah satu elemenya sama dengan cara akses salah satu elemen di list. Pada dictionary, untuk mengakses salah satu elemennya Anda panggil salah satu key-nya untuk mendapatkan data yang ditunjuk key tersebut.

Slicing indeks merupakan cara untuk mengakses beberapa elemen pada list dan tuple. Cara ini tidak dapat dilakukan di dictionary. Slicing indeks dilakukan dengan memanggil list atau tuple kemudian tentukan indeks awal slicing dan batas akhirnya. Kemudian indeks tersebut dipisahkan dengan tanda ":" dan diapit oleh tanda "[" dan "]". Misal ada sebuah list **daftar_barang** kemudian ingin mengambil 10 datanya dari indeks ke – 2 maka pemanggilannya adalah **daftar_barang[1:11]**.

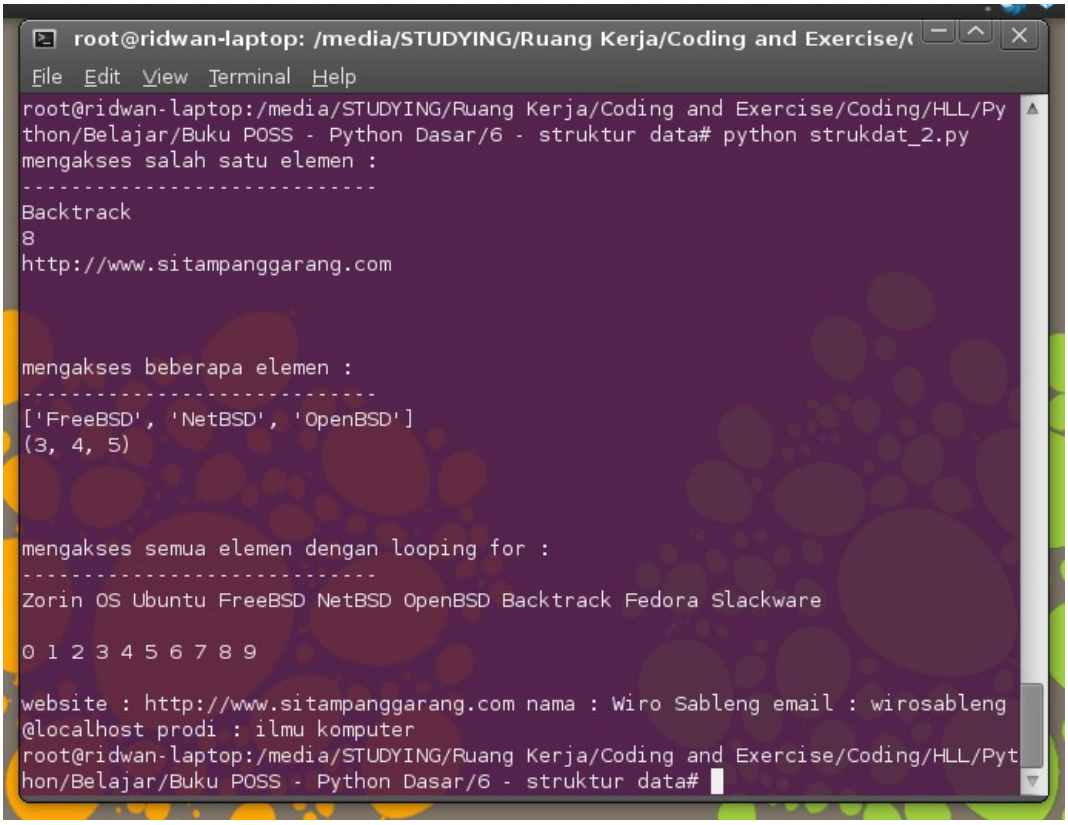
Berikutnya cara terakhir yang biasa dilakukan oleh untuk mengakses elemen secara keseleruhan adalah dengan melalui pengulangan **for**. Melalui cara tersebut, isi dari list, tuple, dan dictionary dapat diambil elemennya selama iterasi pada pengulangan **for**. Pada list dan tuple jika datanya diambil lewat pengulangan **for,** setiap elemen akan langsung diekstrak di setiap iterasi dan dapat digunakan untuk keperluan pemrosesan pada proses di setiap iterasi. Kalau pada dictionary di setiap iterasi pengulangan bukan elemen yang diektrak tapi key-nya. Jadi saat ingin mengambil datanya Anda harus memanggil elemen dictionary tersebut dengan key yang didapatkan disetiap iterasi. Dibawah ini adalah contoh mengakses elemen dari list, tuple dan dictionary dengan tiga cara yang biasa dipakai programmer python.

listing : strukdat_2.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                     'prodi':'ilmu komputer',
                     'email':'wirosableng@localhost',
                     'website':'http://www.sitampanggarang.com'
                     }
```

```
# mengakses elemennya
print "mengakses salah satu elemen : "
print "-----------------------------"
print sebuah_list[5]
print sebuah_tuple[8]
print sebuah_dictionary['website']

print "\n\n"

# mengakses beberapa elemen
print "mengakses beberapa elemen : "
print "-----------------------------"
print sebuah_list[2:5]
print sebuah_tuple[3:6]


print "\n\n"


# mengakses elemennya dengan looping
print "mengakses semua elemen dengan looping for : "
print "-----------------------------"

for sebuah in sebuah_list:
    print sebuah,
print "\n"

for sebuah in sebuah_tuple:
    print sebuah,
print "\n"

for sebuah in sebuah_dictionary:
    print sebuah, ':', sebuah_dictionary[sebuah],
```

Apabila Anda jalankan kode program diatas maka akan muncul output seperti pada gambar berikut ini :



<< gambar 6.2 hasil eksekusi strukdat_2.py >>

Mengubah Isi List, Tuple, dan Dictionary

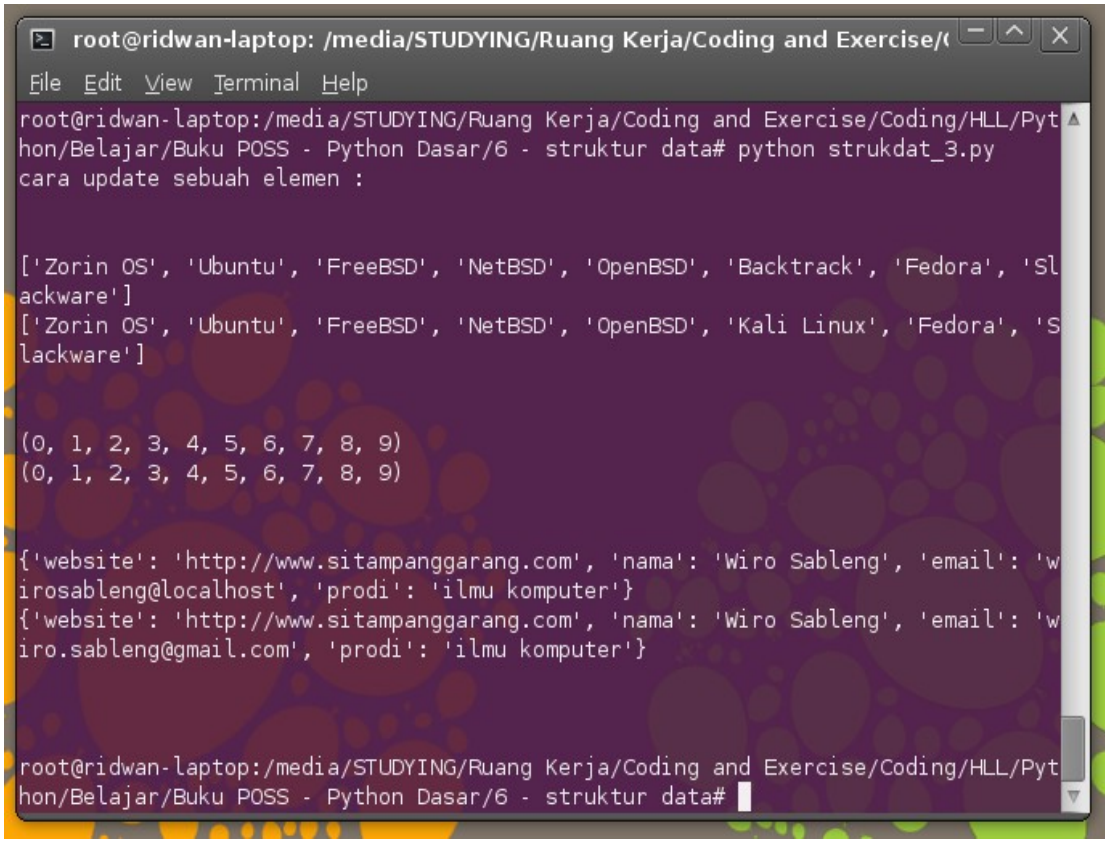
Ada waktunya Anda ingin mengubah salah satu elemen setelah mendefinisikan stuktur data. Misal ada sebuah list **daftar_barang** dan Anda ingin mengubah elemen ke-7 dengan data baru yang asalnya "kursi" menjadi "meja". Atau ada sebuah informasi dalam bentuk dictionary dengan key "nama" yang value asalnya "Son Go Ku" menjadi "Vash De Stampede". Cara mengubah data salah satu elemen di struktur data python mudah sekali. Tinggal tentukan indeks mana yang akan diubah, kemudian masukkan nilai baru kedalam indeks tersebut. Lebih jelasnya coba lihat contoh berikut :

listing : strukdat_3.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
```

```
            'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                'prodi':'ilmu komputer',
                'email':'wirosableng@localhost',
                'website':'http://www.sitampanggarang.com'
                }


# cara update sebuah elemen :
print "cara update sebuah elemen : "
print "\n"

print sebuah_list
sebuah_list[5] = 'Kali Linux'
print sebuah_list
print "\n"

print sebuah_tuple
# tuple tidak dapat melakukan operasi perubahan elemen :D
#sebuah_tuple[5] = 100
print sebuah_tuple
print "\n"

print sebuah_dictionary
sebuah_dictionary['email'] = 'wiro.sableng@gmail.com'
print sebuah_dictionary
print "\n\n"
```

Mudah sekali kan ? Dengan mengakses indeks tertentu pada list dan tuple serta mengakses keys tetentu pada dictionary, Anda dapat mengubah nilai pada indeks atau key tersebut dengan nilai yang baru.



<< gambar 6.3 hasil eksekusi strukdat_3.py >>

Menambahkan Data pada List, Tuple, dan Dictionary

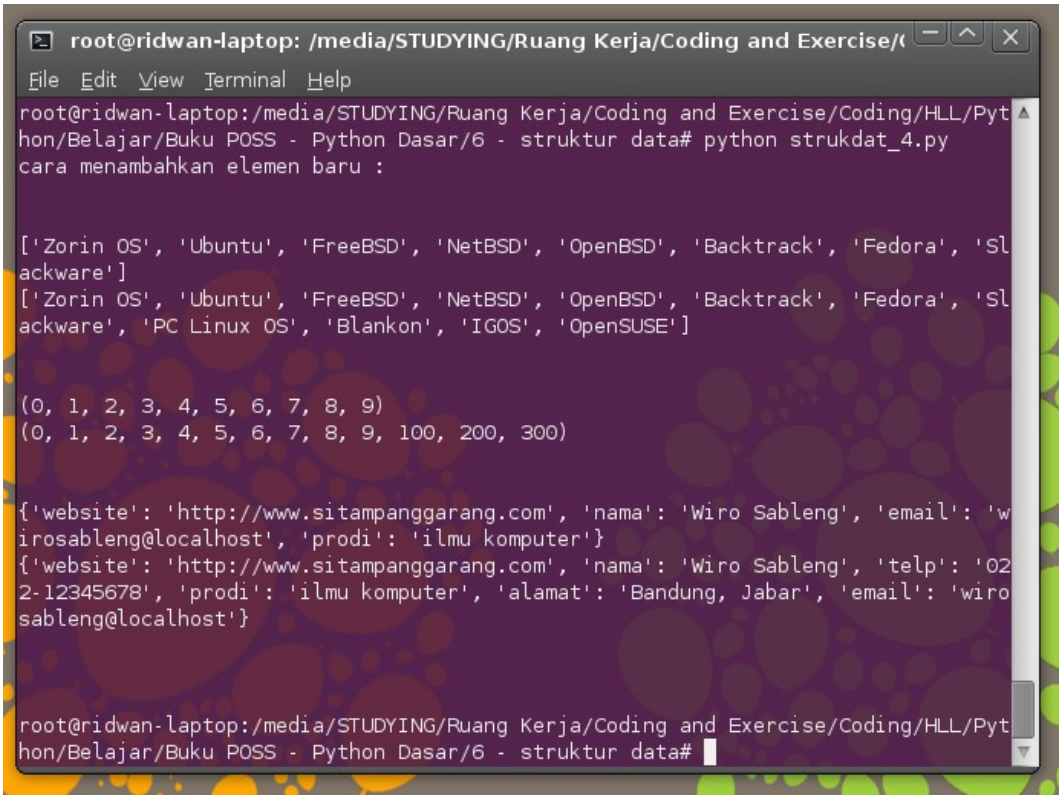
Ketiga struktur data inipun dapat ditambahkan data baru dari data semula. Pada list, digunakan tanda “+” untuk menambahkan data dari list baru ke list lama. Begitupun dengan tuple, tanda “+” digunakan untuk menambahkan data dari tuple baru ke tuple lama. Sedangkan pada dictionary digunakan method update dari dictionary yang ingin ditambahkan data baru. Kemudian dictionary semula akan memiliki data yang ditambahkan melalui method tersebut. Berikut adalah contoh menambahkan data baru pada ketiga struktur data tersebut.

listing : strukdat_4.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
```

```
            'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                     'prodi':'ilmu komputer',
                     'email':'wirosableng@localhost',
                     'website':'http://www.sitampanggarang.com'
                     }


# cara menambahkan data baru
print "cara menambahkan data baru : "
print "\n"

print sebuah_list
list_baru = sebuah_list + ['PC Linux OS', 'Blankon', 'IGOS', 'OpenSUSE']
print list_baru
print "\n"

print sebuah_tuple
tuple_baru = sebuah_tuple + (100, 200, 300)
print tuple_baru
print "\n"

print sebuah_dictionary
dictionary_baru = {'telp':'022-12345678', 'alamat':'Bandung, Jabar'}
sebuah_dictionary.update(dictionary_baru)
print sebuah_dictionary
print "\n\n"
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut ini :



<< gambar 6.4 hasil eksekusi strukdat_4.py >>

Menghapus Isi List, Tuple, dan Dictionary

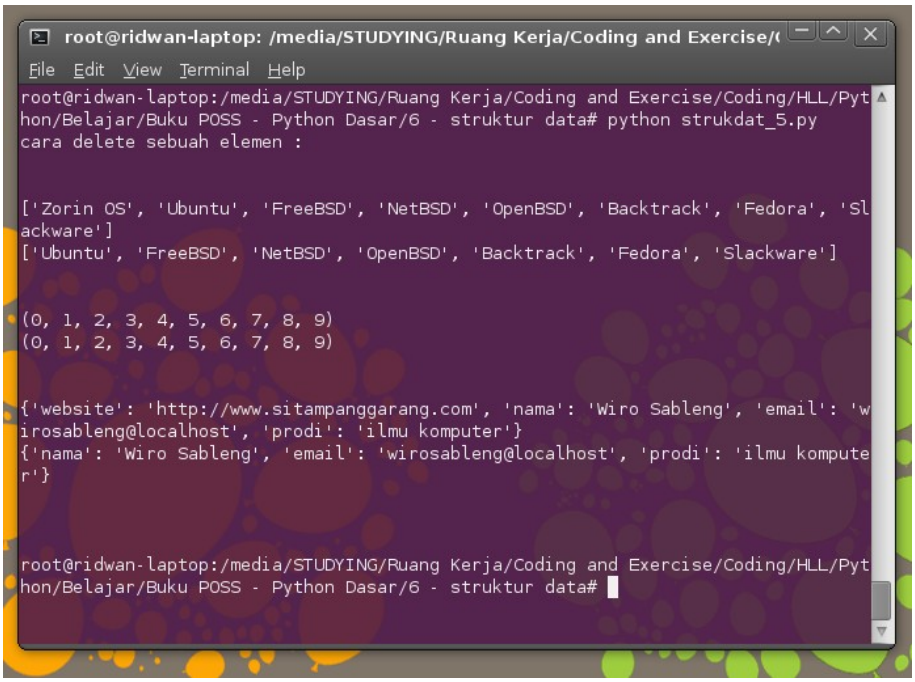
Tidak lengkap rasanya bila dalam sebuah informasi ada yang tidak dapat dihapus. Terkadang ada sebuah data yang tidak Anda butuhkan dan ingin Anda hilangkan dari kumpulan data yang dimiliki. Misal dalam sebuah list Anda ingin menghapus salah satu elemen. Atau di dictionary, Anda ingin menghilangkan salah satu key dari dictionary.tersebut. Di python sendiri penghapusan salah satu elemen dapat dilakukan di list dan dictionary. Sedangkan tuple tidak mendukung penghapusan elemen. Jika kita lakukan penghapusan pada salah satu elemen di tuple, maka akan muncul pesan error : “TypeError: 'tuple' object doesn't support item deletion”. Pada list Anda tinggal menunjuk salah satu elemennya dengan sebuah angka dari 0 sampai panjang list tersebut dikurangi satu dengan diapit tanda “[“ dan “]”. Sedangkan pada dictionary Anda tunjuk salah satu key yang akan dihapus dari dictionary. Berikut adalah contoh penghapusan salah satu elemen pada list dan dictionary.

listing : strukdat_5.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
```

```
            'Ubuntu',
            'FreeBSD',
            'NetBSD',
            'OpenBSD',
            'Backtrack',
            'Fedora',
            'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                'prodi':'ilmu komputer',
                'email':'wirosableng@localhost',
                'website':'http://www.sitampanggarang.com'
                }


# cara delete sebuah elemen :
print "cara delete sebuah elemen : "
print "\n"

print sebuah_list
del sebuah_list[0]
print sebuah_list
print "\n"

print sebuah_tuple
# tuple tidak mendukung proses penghapusan elemen :D.(coba hilangkan tanda '#' disampingnya)
#del sebuah_tuple[8]
print sebuah_tuple
print "\n"

print sebuah_dictionary
del sebuah_dictionary['website']
print sebuah_dictionary
print "\n\n"
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut :



<< gambar 6.5 hasil eksekusi strukdat_5.py >>

Menghapus List, Tuple, dan Dictionary

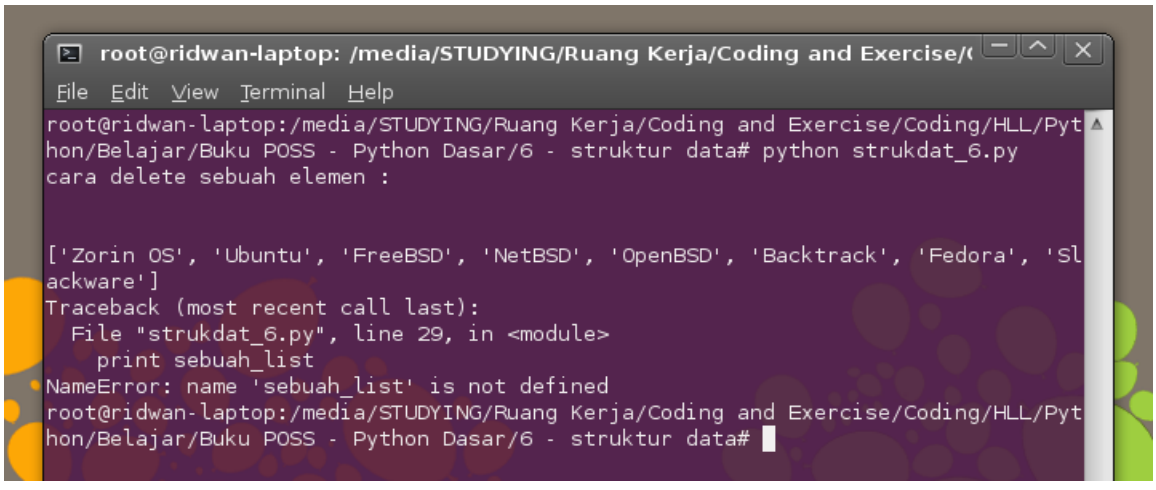
Pada contoh sebelumnya Anda hanya menghapus salah satu elemen. Lalu bagaimanakah jika ingin menghapus keseluruhan struktur data sehingga struktur data tersebut terhapus dari memory seluruhnya ?. Di Python dengan menggunakan perintah **del** pada sebuah struktur data maka struktur data tersebut akan dihapus sepenuhnya dari memory. Hal ini berlaku juga bagi variabel dan objek yang didefinisikan oleh programmer. Dengan hilangnya dari memory maka struktur data yang telah dihapus tidak dapat digunakan lagi oleh program yang Anda bangun.

list : strukdat_6.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']


# cara mendefinisikan tuple
```

```
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                     'prodi':'ilmu komputer',
                     'email':'wirosableng@localhost',
                     'website':'http://www.sitampanggarang.com'
                     }


# cara update sebuah elemen :
print "cara delete sebuah elemen : "
print "\n"

print sebuah_list
del sebuah_list
#print sebuah_list
print "\n"

print sebuah_tuple
del sebuah_tuple
#print sebuah_tuple
print "\n"

print sebuah_dictionary
del sebuah_dictionary
#print sebuah_dictionary
print "\n\n"
```

Cobalah hapus tanda “#” pada baris perintah print di bawah perintah del. Cobalah satu persatu dan amatilah apa yang terjadi. Jika kita coba pada print sebuah_list yang berada dibawah del sebuah_list. Maka akan muncul pesan error : “NameError : name 'sebuah_list' is not defined”



<< gambar 6.6 hasil eksekusi strukdat_6.py >>

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

Apakah fitur – fitur manipulasi list, tuple, dan dictionary hanya terbatas pada hapus, tambah dan ubah ?. Python menyediakan beberapa fitur dasar lainnya yang dapat digunakan untuk proses mencari nilai maksimum dan minimum, menghitung panjang, membandingkan dua buah struktur data yang sejenis, bahkan mengubah struktur data dari list ke tuple atau sebaliknya.

Untuk mencari nilai maksimum pada list, tuple, atau dictionary digunakan function **max()**, sedangkan untuk mencari nilai minimum digunakan function **min()**. Untuk perbandingan dua buah struktur data sejenis, misal list dengan list, digunakanlah function **cmp()**. Function **cmp()** ini akan menghasilkan tiga nilai yaitu -1 jika list pertama kurang dari list kedua, 0 jika kedua list sama, dan 1 jika list pertama lebih besar dari list kedua. Kemudian untuk mencari jumlah elemen yang berada pada struktur data tersebut digunakan function **len()**. Dan terdapat juga untuk konversi tipe struktur data. Tapi fitur ini hanya dapat digunakan pada list dan tuple. Dictionary tidak mendukung proses konversi. Jadi hanya pengubahan dari list ke tuple dan sebaliknya. Untuk pengubahan dari list ke tuple digunakan function **tuple()** sedangkan untuk pengubahan dari tuple ke list digunakan function **list().**

Agar lebih paham cobalah sedikit source code tentang penggunaan built-in function yang digunakan untuk manipulasi list, tuple, dan dictionary.

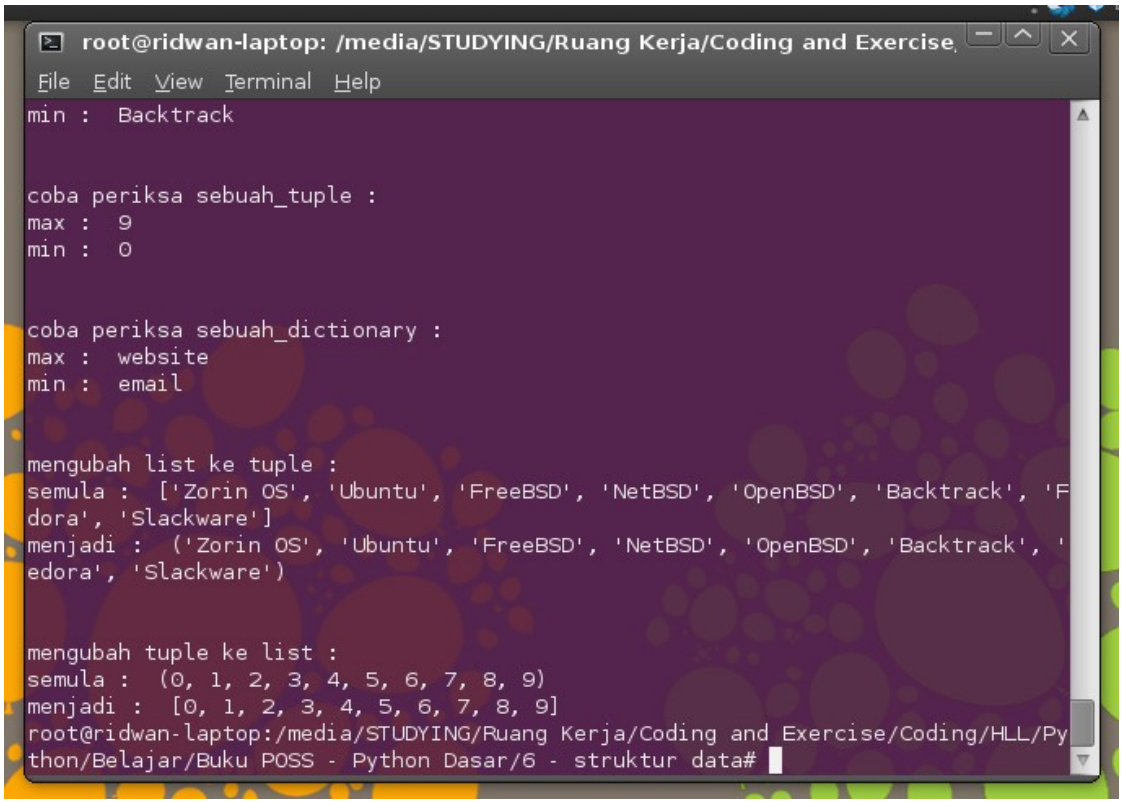
listing : strukdat_7.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']


# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama':'Wiro Sableng',
                     'prodi':'ilmu komputer',
                     'email':'wirosableng@localhost',
                     'website':'http://www.sitampanggarang.com'
                     }
```

```
# menambahkan elemen baru
print "menambahkan elemen baru : \n"
print sebuah_list
list_baru = sebuah_list + ['PC Linux OS', 'Blankon', 'IGOS', 'OpenSUSE']
print list_baru
print "\n"

print sebuah_tuple
tuple_baru = sebuah_tuple
print tuple_baru
print "\n"

print sebuah_dictionary
dictionary_baru = {'telp':'022-12345678', 'alamat':'Bandung, Jabar'}
print sebuah_dictionary
print "\n\n"

# membandingkan yang lama dengan yang baru
print "membandingkan yang lama dengan yang baru : \n"
print "sebuah_list banding list_baru : ", cmp(sebuah_list, list_baru)
print "sebuah_tuple banding tuple_baru : ", cmp(sebuah_tuple, tuple_baru)
print "sebuah_dictionary banding dictionary_baru : ", cmp(sebuah_dictionary, dictionary_baru)

print "\n\n"

# mengetahui panjang list, tuple, dan dictionary
print "mengetahui panjang list, tuple, dan dictionary : \n"
print "panjang sebuah_list : ", len(sebuah_list)
print "panjang sebuah_tuple : ", len(sebuah_tuple)
print "panjang sebuah_dictionary : ", len(sebuah_dictionary)

print "\n\n"

# mengubah list, tuple, dictionary menjadi string
print "mengubah list, tuple, dictionary menjadi string : \n"
print str(sebuah_list), ' memiliki panjang karakter : ', len(str(sebuah_list))
print str(sebuah_tuple), ' memiliki panjang karakter : ', len(str(sebuah_tuple))
print str(sebuah_dictionary), ' memiliki panjang karakter : ', len(str(sebuah_dictionary))

# mencari nilai max dan min
print "mencari nilai max dan min : \n"
print "coba periksa sebuah_list :"
print "max : ", max(sebuah_list)
print "min : ", min(sebuah_list)
print "\n"
print "coba periksa sebuah_tuple :"
print "max : ", max(sebuah_tuple)
print "min : ", min(sebuah_tuple)
print "\n"
print "coba periksa sebuah_dictionary :"
```

```
print "max : ", max(sebuah_dictionary)
print "min : ", min(sebuah_dictionary)
print "\n"

# mengubah list ke tuple dan sebaliknya
print "mengubah list ke tuple : "
print "semula : ", sebuah_list
print "menjadi : ", tuple(sebuah_list)
print "\n"
print "mengubah tuple ke list : "
print "semula : ", sebuah_tuple
print "menjadi : ", list(sebuah_tuple)
```

Dengan adanya *built-in function* tersebut pekerjaan Anda sudah dimudahkan oleh Python dalam memanipulasi struktur data yang telah Anda definisikan sebelumnya. Berikut adalah salah satu contoh hasil operasi dengan *built function* dari kode diatas.



<< gambar 6.7 hasil eksekusi strukdat_7.py >>

7. Membuat Function

Pengenalan Function Tanpa “return”

Pada saat membuat program terkadang kita menyalin blok kode yang sama di baris berikutnya, misal untuk mencetak sebuah biodata kita salin kembali kemudian ganti nilai – nilainya untuk menampilkan biodata tersebut. Apakah harus menyalin blok kode di setiap bagian kode yang membutuhkan kode tersebut ???. Waktu dulu kuliah Algoritma dan Pemrograman I, ketika mempelajari pemrograman pertama kalinya, baris kode yang sama dipakai berulang – ulang untuk mengeluarkan hasil tertentu. Tapi jadi tidak efisien, karena ukuran file program yang ditulis cukup besar. Oleh karena itu hampir di setiap bahasa pemrograman terdapat fitur yang dinamakan *function*. Nama lainnya *method*, *sub routine*, atau fungsi. *Function* ini berguna untuk penggunaan kembali blok kode yang akan digunakan di baris kode lain. Sekali tulis tinggal panggil.

Function ini jelas beda dengan *built-in function* yang ada di Python. *Built-in function* sendiri adalah *function* yang telah dibuatkan oleh pengembang bahasa pemrograman Python. Sedangkan *function* dibuat oleh programmer yang menggunakan bahasa pemrograman Python, atau istilah lainnya *user-defined function*.

Di Python untuk membuat *function* digunakan *keyword* **def**. Kemudian diikuti nama *function* yang diinginkan lalu parameter yang dibutuhkan dengan diawali tanda “(“ dan “)”.Untuk membuka *function* dimulai dengan tanda “:”. Tidak seperti di C, Java, atau PHP yang diawali dengan tanda “{“ dan diakhiri “}” untuk membuka sebuah *function*. Lalu tutupnya ? Seperti dijelaskan diawal di Python sendiri digunakan indentasi untuk menentukan apakah baris kode tersebut milik sebuah *function*, **if**, atau pengulangan. Jadi jika Anda ingin menuliskan kode untuk *function* yang Anda buat. Harus ada jarak satu indentasi agar kode tersebut dianggap sebagai kode dari *function* yang Anda tulis. Kemudian Anda dapat menambahkan *keyword* **return** jika *function* yang Anda tulis ingin mengembalikan nilai keluaran.

Berikut ada contoh tentang pembuatan *function* yang memiliki parameter dan tidak memiliki parameter. Penggunaan **return** digunakan pada contoh berikutnya :

listing : fungsi_1.py

```
def fungsi_tanpa_parameter():
    for i in range(1, 5):
        print "looping ke - ", i

def fungsi_berparameter(batas_akhir):
    for i in range(1, batas_akhir):
        print "looping ke - ", i
```

```
print " contoh penggunaan fungsi tanpa parameter : "
print "hasil : ", fungsi_tanpa_parameter()

print "\n\n"

print " contoh penggunaan fungsi berparameter : "
print "hasil : ", fungsi_berparameter(10)
```

Sebuah *function* jika dipanggil langsung nilai keluarannya tidak akan dicetak. Tetapi jika dipanggil melalui sebuah *function* seperti **print** nilai keluarannya akan ditampilkan. Kenapa nilainya "None" ? Karena di *function* yang tadi ditulis tidak disertakan *keyword* **return**. Jika sebuah *function* tidak diberikan **return** maka dapat dibilang *function* tersebut dianggap *procedure*. Sebuah *function* yang tidak memiliki nilai keluaran.

<< gambar 7.1 hasil eksekusi fungsi_1.py >>

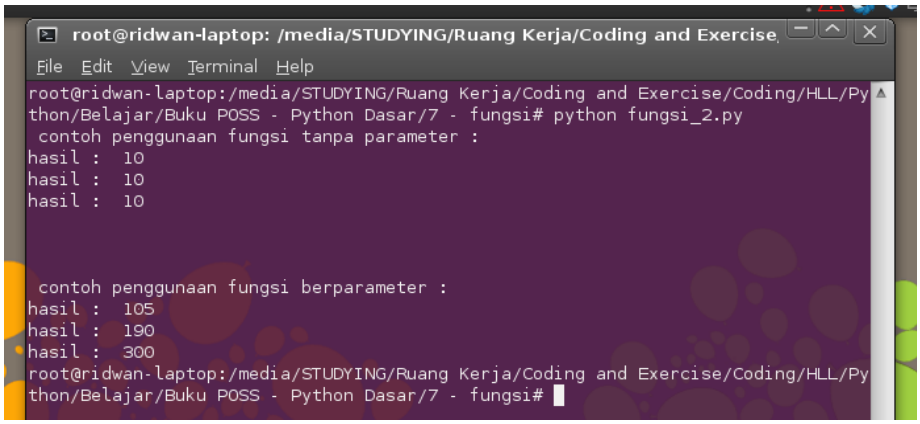
Function yang Menggunakan "return"

Bagaimana kalau ditambahkan **return** ? Jika ditambahkan **return,** *function* yang Anda tulis akan menghasilkan nilai keluaran. Jika dipanggil langsung maka program tidak akan menampilkan nilai keluaran dari *function* tersebut. Jika *function* tersebut dipanggil melalui *function* atau *keyword* misalnya **print,** maka nilai keluarannya akan ditampilkan. Berikut terdapat *function* yang menghasilkan nilai keluaran yang memiliki parameter dan tidak berparameter :

listing : fungsi_2.py

```
def fungsi_tanpa_parameter():
    temp = 0
    for i in range(1, 5):
        temp = temp + i
    return temp


def fungsi_berparameter(batas_akhir):
    temp = 0
    for i in range(1, batas_akhir):
        temp = temp + i
    return temp


print " contoh penggunaan fungsi tanpa parameter : "
print "hasil : ", fungsi_tanpa_parameter()
print "hasil : ", fungsi_tanpa_parameter()
print "hasil : ", fungsi_tanpa_parameter()

print "\n\n"

print " contoh penggunaan fungsi berparameter : "
print "hasil : ", fungsi_berparameter(15)
print "hasil : ", fungsi_berparameter(20)
print "hasil : ", fungsi_berparameter(25)
```

Anda sendiri dapat melihat perbedannya antara *function* yang berparameter dengan tidak berparameter. Pada *function* yang tidak berparameter. Ketika dipanggil berulang – ulang nilai keluarannya tetap sama. Berbeda dengan *function* yang memiliki parameter, nilai keluaranya berbeda – beda ketika dipanggil. Tergantung nilai masukan yang diberikan.



<< gambar 7.2 hasil eksekusi fungsi_2.py >>

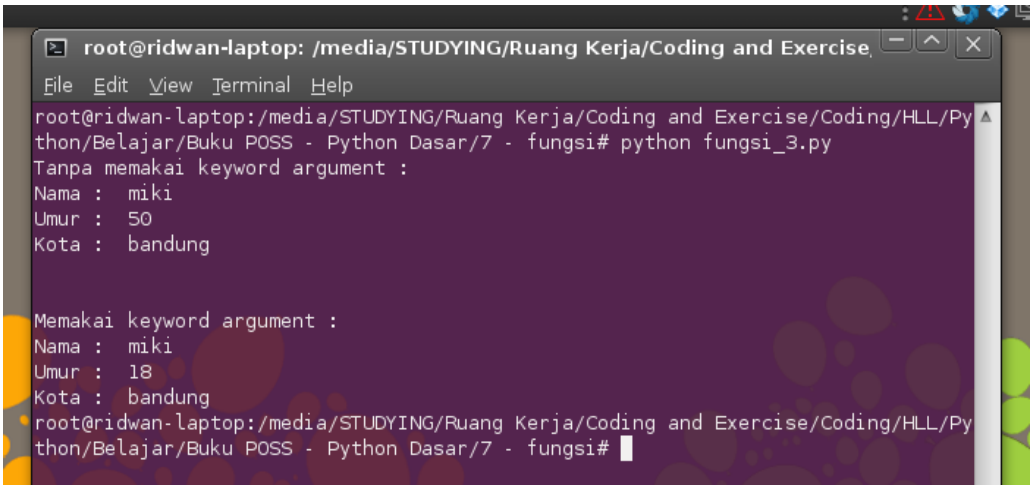
Default Argument pada Python

Sekarang Anda sudah mengenal *function* yang berparameter dan tidak berparameter. Umumnya saat akan memberikan nilai pada sebuah *function*, nilai tersebut akan diberikan saat *function* tersebut dipanggil. Apakah saat memasukkan nilai boleh tidak diisi atau dilewat ? Lalu apakah akan ada nilainya ?. Di Python terdapat sebuah fitur yang dinamakan *default argument* saat menulis sebuah *function*. *Default argument* sendiri adalah sebuah argumen yang sudah diisi nilai terlebih dahulu jika argumen tersebut tidak diberikan saat memanggil *function*. Jadi sekalipun dilewat nilai dari argument tersebut akan dipenuhi dengan nilai *default* nya. Berikut dibawah ini terdapat contoh pemanggilan *function* yang melewatkan semua argumen yang dibutuhkan *function*, dan yang tidak melewatkan semua argumen yang akan ditangani oleh *default argument* :

listing : fungsi_3.py

```
def cetak_biodata( nama, kota, umur=18):
   print "Nama : ", nama;
   print "Umur : ", umur;
   print "Kota : ", kota;
   return;

# kalau parameter diisi semua
print "Tanpa memakai default argument : "
cetak_biodata( nama="miki", umur=50, kota="bandung" )

print "\n"

# kalau parameter tidak diisi semua
print "Memakai default argument : "
cetak_biodata(kota="bandung", nama="miki")
```

Kode diatas jika dieksekusi akan tampil seperti berikut :



<< gambar 7.3 hasil eksekusi fungsi_3.py >>

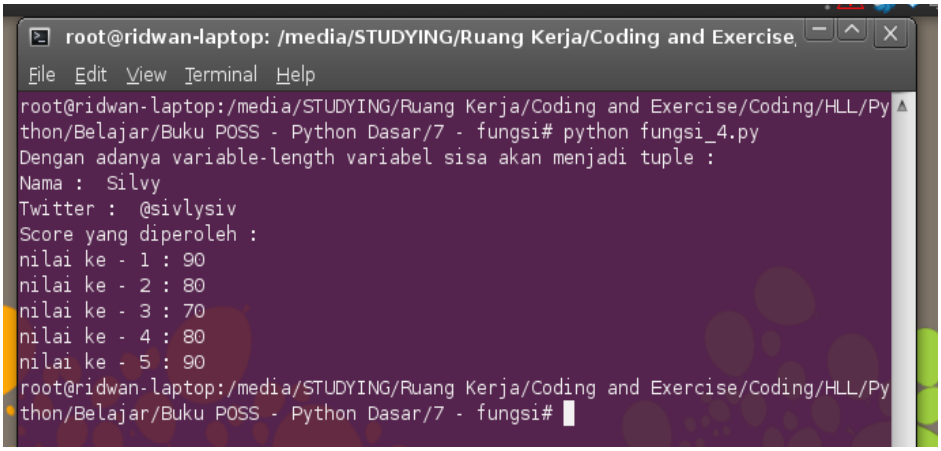
Variable-length Argument pada Python

Sekarang kita akan mengenal fitur yang dinamakan dengan *variable-length argument*. Fitur ini digunakan ketika ingin membuat sebuah *function* yang memiliki argumen yang dinamis. Argumen ini dapat disebut sebagai argumen yang tidak utama. Misal dalam sebuah fungsi dibutuhkan tiga argumen, maka argumen ke – 4 sampai ke – n argumen, tidak akan ditampung oleh argumen utama. Tapi ditampung oleh argumen terakhir yang menampung seluruh argumen yang diberikan setelah argumen utama. Di Python untuk menandakan bahwa argumen tersebut *variable-length argument*, diberikan tanda "*" pada argumen terakhir. *Variable-length argument* ini harus disimpan di akhir setelah argumen biasa dan *default argument*. Apabila disimpan di urutan awal, maka Python akan mengeluarkan *error* : **"SyntaxError: invalid syntax"**. Sebagai contoh Anda dapat perhatikan kode berikut ini :

listing : fungsi_4.py

```
def cetak_perolehan_nilai( nama, twitter, *scores):
   print "Nama : ", nama;
   print "Twitter : ", twitter;
   print "Score yang diperoleh : "
   i = 1
   for score in scores:
      print "nilai ke - %d : %d" % (i, score)
      i= i + 1

   return;

# kalau parameter diisi semua
print "Dengan adanya variable-length variabel sisa akan menjadi tuple : "
cetak_perolehan_nilai("Silvy", "@sivlysiv", 90, 80, 70, 80, 90)
```

Seperti yang Anda lihat pada contoh diatas, argumen utama adalah nama dan twitter. Apabila kita memasukkan argumen setelahnya, maka argumen tersebut akan dikumpulkan dalam satu wadah yaitu *scores. Berapapun kita masukkan argumen, akan ditampung menjadi sebuah **list** yang berisi argumen – argumen yang dimasukkan setelah nama dan twitter.



<< gambar 7.4 hasil eksekusi fungsi_4.py >>

Keyword Argument pada Function

Dalam penggunaan *function* Anda mesti melewatkan argumen sesuai urutan yang ditulis pada parameter yang dibutuhkan oleh *function* yang Anda tulis. Apakah mungkin jika ditulis tanpa urutan argumen sudah baku pada *function* tersebut. Dalam *function* terdapat sebuah fitur yang dinamakan *keyword argument*. *Keyword argument* ini dapat melewatkan argumen tanpa harus sesuai urutan. *Keyword argument* diberikan saat memanggil sebuah *function* dengan mengambil nama argumen yang terdapat di *function* disambung dengan tanda "=" dan nilai dari argumen tersebut. Jika kita memberikan argumen yang tidak sesuai urutan tanpa menggunakan *keyword argument*, maka argumen yang diterima *function* tersebut tidak akan sesuai.

listing : fungsi_5.py

```
def cetak_biodata( nama, umur, kota):
   print "Nama : ", nama;
   print "Umur : ", umur;
   print "Kota : ", kota;
   return;

# kalau pakai keyword argument : mau urutannya gimanapun input akan sesuai
print "Tanpa memakai keyword argument : "
```

```
cetak_biodata( kota="bandung", nama="miki", umur=50 )

print "\n"

# kalau tidak memakai keyword argument : mau urutannya gimanapun input tidak akan sesuai
print "Memakai keyword argument : "
cetak_biodata( "bandung", "miki", 50)

print "\n"

# kalau tidak memakai keyword argument : tapi urutannya sesuai maka input  akan sesuai
print "Memakai keyword argument : tapi urutannya sesuai "
cetak_biodata(  "miki", 50, "bandung")
```

Pada contoh diatas, Anda dapat melihat perbedaan antara *function* yang melewatkan *keyword argument* dengan yang tidak menggunakan *keyword argument*. Contoh yang tidak menggunakan *keyword argument* tidak akan menerima masukan sesuai yang dibutuhkan *function* ketika urutan argumennya diacak.

<< gambar 7.5 hasil eksekusi fungsi_5.py >>

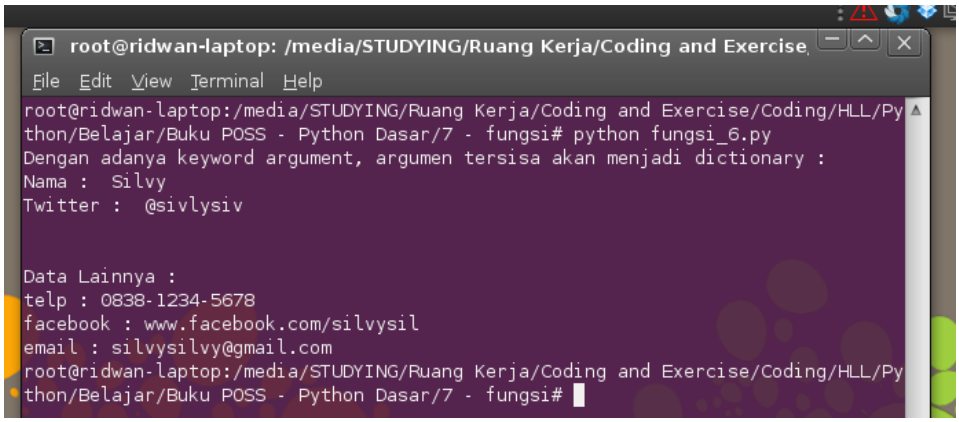
Keyword-length Argument pada Function

Keyword-length argument mempunyai cara penggunaan yang sama hanya saja, *keyword-length* ini menampung *keyword argument* yang berlebih ketika diberikan kepada *function* yang dipanggil. *Keyword argument* yang berlebih akan diterima dalam bentuk **dictionary**.

listing: fungsi_6.py

```
def cetak_perolehan_nilai( nama, twitter, **data_tambahan):
   print "Nama : ", nama;
   print "Twitter : ", twitter;
   print "\n"
   print "Data Lainnya : "
   i = 1
   for data in data_tambahan:
      print "%s : %s" % (data, data_tambahan[data])


   return;

# kalau parameter diisi semua
print "Dengan adanya keyword argument, argumen tersisa akan menjadi dictionary : "
cetak_perolehan_nilai("Silvy","@sivlysiv",email="silvysilvy@gmail.com",
facebook="www.facebook.com/silvysil", telp="0838-1234-5678")
```

Pada contoh diatas, *keyword argument* yang berlebih ditampung kedalam *argument* data_tambahan dan argumen berlebih tersebut disimpan dalam **dictionary**.



<< gambar 7.6 hasil eksekusi fungsi_6.py >>

Pass by Reference dan Pass by Value pada Python

Berikutnya terdapat masalah *pass by value* atau *pass by reference*. Di Python semua nilai akan dilewatkan secara *by reference*. Artinya jika kita mengubah argumen di dalam fungsi maka nilai argumen yang direferensi tersebut akan ikut berubah juga. Misalkan dibawah contoh berikut terdapat sebuah **list** yang akan diganti dengan nilai baru, dan ada juga yang ditambahkan nilai baru.

listing : fungsi_7.py

```
def sebuah_fungsi(sebuah_list):
    sebuah_list = [1, 2, 3, 4, 5]
    print sebuah_list

def sebuah_fungsi_lainnya(sebuah_list):
    sebuah_list.append([10, 20, 30])
    print sebuah_list

ini_list = [10, 20, 30]
sebuah_list = [100, 200, 300]

print "apakah ini_list berubah ? "
print ini_list
sebuah_fungsi(ini_list)
print ini_list
print ini_list
sebuah_fungsi_lainnya(ini_list)
print ini_list

print "apakah sebuah_list berubah ? "
print sebuah_list
sebuah_fungsi(sebuah_list)
print sebuah_list
print sebuah_list
sebuah_fungsi_lainnya(sebuah_list)
print sebuah_list
```

Pada kode diatas, Anda akan melihat sebuah perbedaan yang cukup penting. Ketika sebuah **list** diganti nilainya maka **list** yang ada di luar *function* tidak akan terpengaruh. Tapi ketika kita menambahkan data baru dengan menggunakan *method* pada **list** tersebut. Nilai diluar ikut berubah,. Hal ini terjadi karena pada *function* sebuah_fungsi_lainnya(), **list** sebuah_list masih menunjuk atau merujuk ke sebuah_list yang berada diluar. Atau dalam kata lain masih menunjuk ke “address” yang sama di memori utama.

<< gambar 7.7 hasil eksekusi fungsi_7.py >>

Variable Scope pada Python

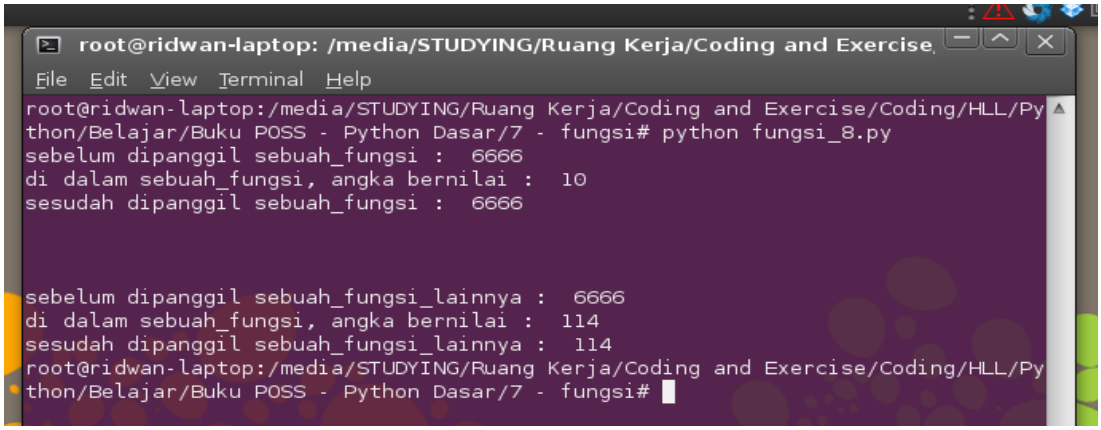
Variable scope adalah sebuah kondisi dimana variabel diakses secara lokal pada blok kode tertentu atau bersifat universal yang menyebabkan variabel tersebut dapat diakses dari blok kode manapun. Misal ada sebuah variabel di dalam *function*. Variabel tersebut bersifat lokal dan hanya dapat diakses didalam *function* tersebut. Lalu bagaimanakah kita menjadikan sebuah variabel agar bersifat global ?. Di Python terdapat sebuah *keyword* yang bernama **global**. *Keyword* ini digunakan untuk merujuk sebuah variabel di luar blok kode, misalnya sebuah variabel di dalam *function*, dengan nama yang sama.

listing : fungsi_8.py

```
def sebuah_fungsi():
    angka =  10
    print "di dalam sebuah_fungsi, angka bernilai : ", angka

def sebuah_fungsi_lainnya():
    global angka
    angka = 114
    print "di dalam sebuah_fungsi, angka bernilai : ", angka


angka = 6666

print "sebelum dipanggil sebuah_fungsi : ", angka
sebuah_fungsi()
print "sesudah dipanggil sebuah_fungsi : ", angka

print "\n\n"
```

```
print "sebelum dipanggil sebuah_fungsi_lainnya : ", angka
sebuah_fungsi_lainnya()
print "sesudah dipanggil sebuah_fungsi_lainnya : ", angka
```

Pada kode diatas variabel yang bernama angka dibubuhi *keyword* **global** pada *function* sebuah_fungsi_lainnya(). Hasilnya saat angka diubah nilainya. Maka nilai di variabel angka yang berada di luar blok *function* sebuah_fungsi_lainnya() ikut berubah.



<< gambar hasil eksekusi fungsi_8.py >>

8. Mengenal Exception

Jenis – Jenis Exception

Exception adalah sebuah cara di Python untuk menjebak *error*, dan menangangi *error* tak terduga pada program Python yang Anda tulis. *Exception* akan tetap menjalankan baris kode program dibawah bagian kode program yang *error*. Hal ini mempermudah proses *debugging*. Lalu apa bedanya jika kita menggunakan kondisional biasa yang menggunakan **if** untuk mencegah *error* ? Pertama Anda harus mencari cara untuk menangkap nilai – nilai yang *error*, misal ketika membuka *file* Anda harus menggunakan *method – method* yang ada pada *file* untuk mengetahui *error* atau tidak. Kedua dengan menggunakan kondisional **if** biasa, program yang Anda tulis akan langsung dihentikan ketika *error* terjadi. Ketiga pengambilan *error* akan otomatis ditangani oleh Python dan *error* tersebut akan ditangani sesuai dengan penanganan yang Anda lakukan, dan baris kode program dibawahnya akan tetap dieksekusi.

Python sendiri sudah menyediakan beberapa *standard error* yang dapat digunakan oleh *programmer* dalam menjaga pengembangan aplikasinya dari *error* yang tak terduga. Anda sendiri dapat membuat *error* menurut definisi Anda. Hal tersebut akan diulas di bagian akhir bab ini. Berikut adalah beberapa *standard error* yangt terdapat di Python :

No	Nama *Exception*	Keterangan
1	Exception	Menangani semua *exception*
2	StopIteration	*Exception* ini muncul ketika *method* next() tidak menunjuk ke objek apapun saat iterasi
3	SystemExit	*Exception* ini muncul ketika sys.exit() dipanggil
4	StandardError	*Exception* untuk menangani semua *built-in exception* kecuali StopIteration dan SystemExit
5	ArithmeticError	*Exception* untuk menangani *error* saat perhitungan angka
6	OverflowError	*Exception* ini muncul ketika perhitungan angka melebihi batas maksimum dari tipe angka yang dihitung
7	FloatingPointError	*Exception* ini muncul ketika terdapat kegagalan pada perhitungan angka bertipe *float*
8	ZeroDivisionError	*Exception* ini muncul jika ada pembagian atau modulus oleh 0 terhadap angka tipe apapun
10	AssertionError	*Exception* ini muncul ketika terjadi kegagalan pada saat perintah **assert** dijalankan
11	AttributeError	*Exception* ini muncul ketika gagal menunjuk atribut dari suatu objek
12	EOFError	*Exception* ini muncul ketika tidak ada input saat

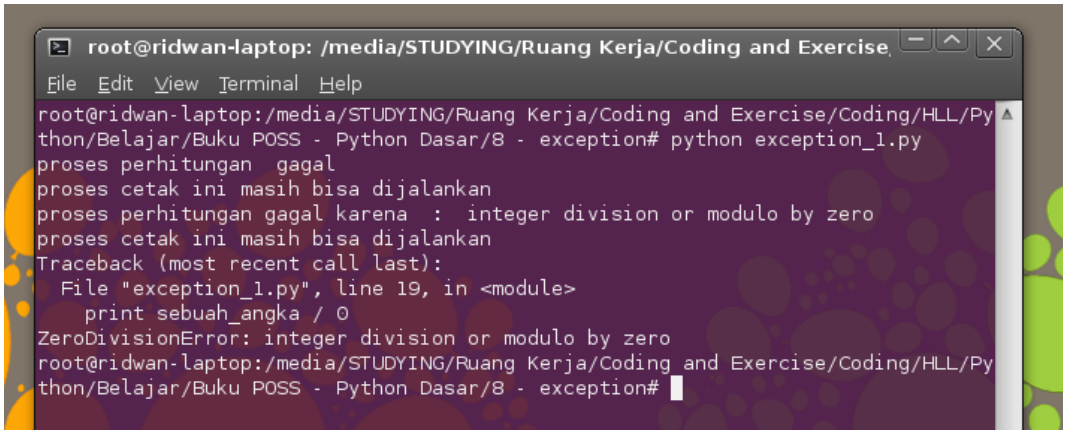
		menggunakan *function* raw_input() atau input dan telah mencapai bagian akhir file saat pembacaan file.
13	ImportError	*Exception* ini muncul ketika gagal saat menggunakan **import**
14	KeyboardInterrupt	*Exception* ini muncul ketika user meng-*interrupt* eksekusi program, biasanya ketika menekan kombinasi **ctrl + c**
15	LookupError	*Exception* muncul ketika gagal pada saat proses *look up*
16	IndexError	*Exception* ini muncul ketika tidak ada indeks yang dituju pada struktur data seperti **list** atau **tuple**
17	KeyError	*Exception* ini muncul ketika tidak ada *key* yang dituju pada *dictionary*
18	NameError	*Exception* ini muncul ketika variabel tidak ditemukan pada lingkup lokal di suatu function dan kondisional atau pada lingkup global
19	UnboundLocalError	*Exception* ini muncul ketika mencoba mengakses variabel lokal di *function* atau *method* tapi belum ada nilainya
20	EnvironmentError	*Exception* in muncul ketika terjadi kegagalan diluar lingkup Python
21	IOError	*Exception* in muncul ketika proses *input/output* gagal, misal saat menggunakan **print** atau saat membuka *file*
22	OSError	*Exception* ini muncul ketika terjadi kegagalan pada sistem operasi yang digunakan
23	SyntaxError	*Exception* ini muncul ketika terjadi kesalahan pada penggunaan sintaks Python
24	IndentationError	*Exception* ini muncul ketika indentasi pada blok kode tidak sesuai penggunaannya.
25	SystemError	*Exception* ini muncul ketika terdapat masalah internal pada *interpreter*, saat *error* ini muncul interpreter tidak akan keluar
26	TypeError	*Exception* ini muncul jika ada kesalahan tipe data saat proses perhitungan, misal huruf dibagi angka
27	ValueError	*Exception* ini muncul ketika argumen yang tidak sesuai diterima oleh *builtin function*
28	RuntimeError	*Exception* ini muncul ketika terjadi kesalahan yang tidak masuk kategori manapun
29	NotImplementedEror	*Exception* ini muncul ketika *abstract method* dari suatu *class* tidak diimplementasikan di *class* yang mewarisinya.

Agar lebih paham dibawah ini ada beberapa contoh kode yang penggunaan *exception*-nya sangat sering digunakan. Sebagai contoh pertama berikut terdapat kode yang berisi pembagian oleh angka nol.

listing : exception_1.py

```
sebuah_angka = 29

try:
    print sebuah_angka / 0
except:
    print "proses perhitungan  gagal "

print "proses cetak ini masih dapat dijalankan  "


try:
    print sebuah_angka / 0
except ZeroDivisionError, e:
    print "proses perhitungan gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "


print sebuah_angka / 0

# jika tidak memakai exception maka proses berikutnya tidak akan dijalankan
print "apakah proses cetak ini masih dapat dijalankan ??? "
```

Di dalam try terdapat kode yang kemungkinan akan memunculkan exception. Sedangkan di dalam except adalah kode yang akan dieksekusi jika exception tersebut muncul. Pada try-except yang pertama, semua error akan ditangani dan Anda tidak akan mengetahui jenis exception apa yang yang ditangani. Pada try-except yang kedua, Anda memprediksi akan menangani error jika terjadi pembagian oleh nol. Manakah yang lebih baik ? Pastikan Anda sudah memiliki perkiraan apa saja error yang akan terjadi sehingga saat debugging nanti akan mempermudah Anda untuk memperbaiki kode program Anda. Pada blok kode try-except sekalipun error kode dibawahnya akan tetap dieksekusi. Pada proses perhitungan di bagian akhir tidak ditangani oleh try-except sehingga kode dibawahnya tidak dieksekusi. Berikut hasil yang diberikan jika kode dieksekusi :



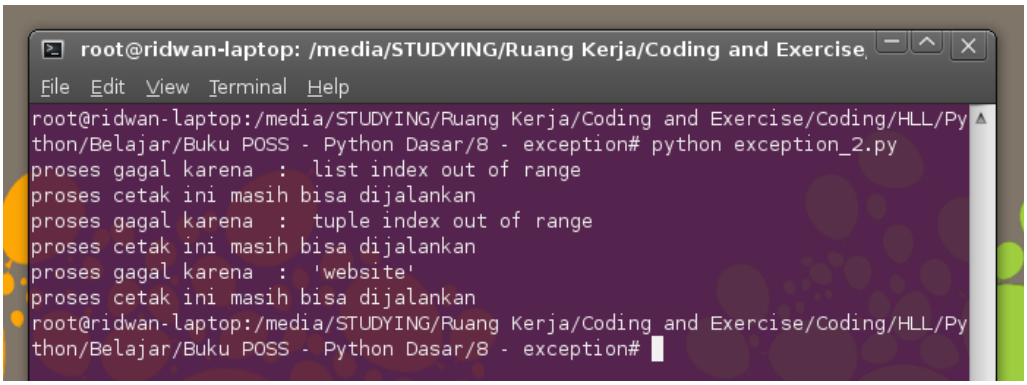
<< gambar 8.1 hasil eksekusi exception_1.py >>

Contoh lain yang umum dipakai adalah IndexError dan KeyError. Kedua *error* ini umum dipakai saat operasi **list**, **tuple**, dan *dictionary*. Berikut terdapat contoh menunjuk indeks dan *key* yang tidak terdapat pada **list**, **tuple**, dan *dictionary* yang didefinisikan dalam kode dibawah ini.

listing : exception_2.py

```
sebuah_list = [1, 2, 3, 4, 5]
sebuah_tuple = (1, 2, 3, 4, 5)
sebuah_dictionary = {'nama':'Mangaraja', 'email':'mangaraja@yahoo.com'}

try:
    print sebuah_list[10]
except IndexError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "

try:
    print sebuah_tuple[10]
except IndexError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "

try:
    print sebuah_dictionary['website']
except KeyError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Pada contoh diatas “sebuah_list” dan “sebuah_tuple” ditangani oleh *try-except* yang menangani *exception* IndexError. Pada masing – masing blok, kita ingin mencoba indeks yang tidak ada pada **list** dan **tuple** tersebut. Sedangkan pada blok *try-except* untuk *dictionary*, kita ingin mencoba menunjuk *key* “website” tapi karena *key* tersebut tidak ada, maka akan muncul exception KeyError.



<< gambar 8.2 hasil eksekusi exception_2.py >>

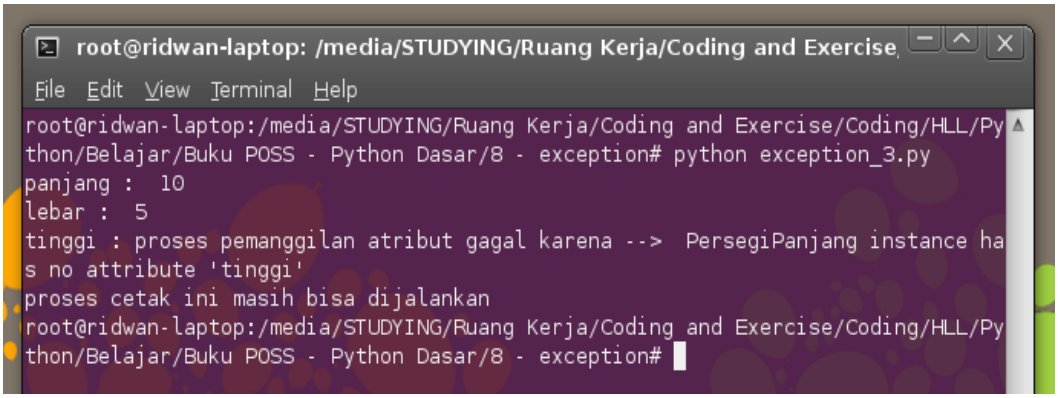
Berikutnya contoh *exception* yang tak kalah populer lainnya adalah AttributeError. *Exception* ini muncul ketika sebuah *class* tidak memiliki atribut (variabel) yang diakses oleh *programmer*. Hal ini sangat penting untuk diperhatikan ketika merancang sebuah aplikasi berbasis objek. Anda harus memeriksa apakah atribut yang Anda akses pada sebuah kelas ada pada saat perancangan atau tidak. Jika tidak yakin gunakanlah *try-except* untuk menjebak AttributeError tersebut.

listing : exception_3.py

```
class PersegiPanjang:
    panjang = 0
    lebar = 0
    def __init__(self, p, l):
        self.panjang = p
        self.lebar = l


prsg_pjg = PersegiPanjang(10, 5)

try:
    print "panjang : ", prsg_pjg.panjang
    print "lebar : ", prsg_pjg.lebar
    print "tinggi :", prsg_pjg.tinggi
except AttributeError, e:
    print "proses pemanggilan atribut gagal karena --> ", e
```

```
print "proses cetak ini masih dapat dijalankan"
```

Pada contoh diatas, kita ingin mencoba mengakses atribut tinggi pada objek prsg_pjg. Sebelumnya tahapan yang dilalui adalah proses instansiasi, dimana kita memanggil sebuah *template* objek yang akan dibentuk kedalam sebuah variabel. Kemudian di bagian *try-except* tersebut kita coba akses atribut tinggi. Karena atribut tersebut tidak ada di kelas persegi panjang, maka *exception* AttributeError akan muncul.



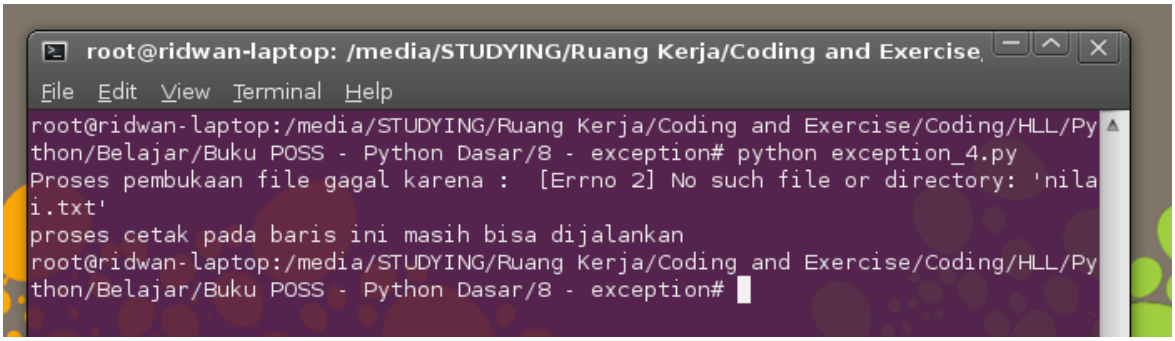
<< gambar 8.3 hasil eksekusi exception_3.py >>

Contoh yang terakhir dari sekian banyak *exception* yang terdapat di Python adalah IOError. *Exception* ini biasa terjadi ketika proses *input* data, saat mencetak data, atau saat operasi *file*. Pada contoh berikut kita akan membuka sebuah *file*, tapi *file* tersebut tidak ada. Secara *default* jika kita membuka *file* tanpa menyertakan mode pembacaan, maka mode tersebut adalah mode 'r' yang artinya *read* atau baca.

listing : exception_4.py

```
try :
    f = open('nilai.txt')
except IOError, e:
    print "Proses pembukaan file gagal karena : ", e

print "proses cetak pada baris ini masih dapat dijalankan"
```

Pada contoh diatas kita ingin *file* nilai.txt, tapi karena *file* tersebut belum pernah ditulis sebelumnya maka *exception* akan muncul yaitu IOError. Selain digunakan untuk *file*, IOError dapat terjadi juga saat pembacaan *built-in storage* milik Python seperti saat pembacaan pickle, shelve, dan marshal.



<< gambar 8.4 hasil eksekusi exception_4.py >>

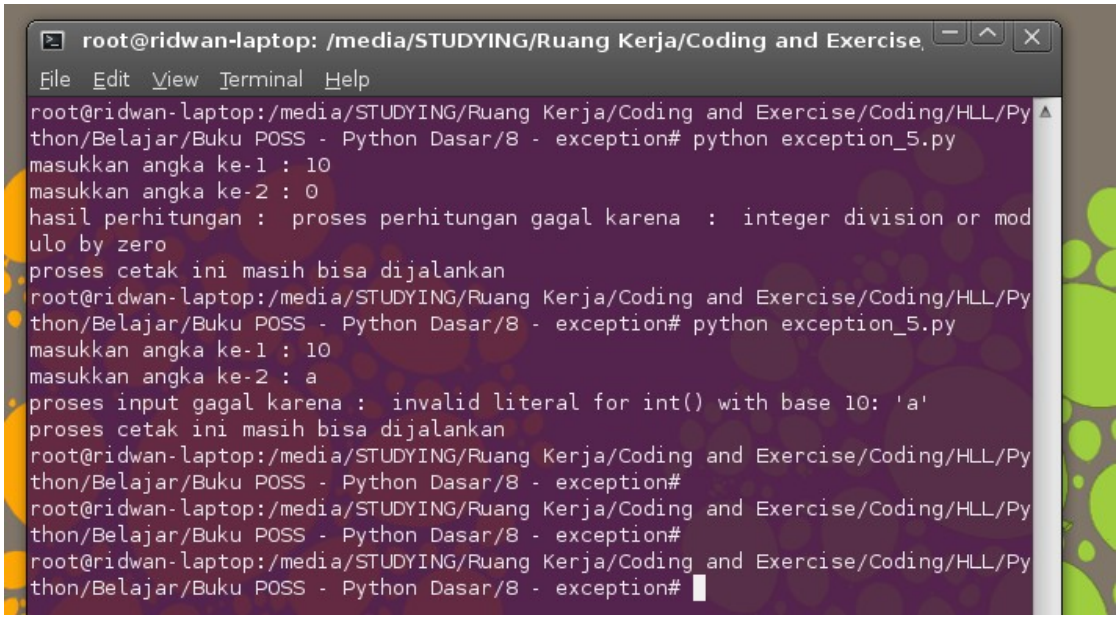
Menyusun Multiple Except

Apakah kita dapat menangkap *exception* dalam satu blok *try-except* ?. Di Python sendiri terdapat fitur *multiple except*, yaitu kita dapat menangkap *exception* dengan baris **except** yang berbeda. Hal ini dilakukan jika kita ingin memberikan perlakuan berbeda kepada setiap *exception* yang ditangani. Lebih lemgkapnya pantau kode dibawah ini.

listing : exception_5.py

```
try:

    angka1 = int(raw_input('masukkan angka ke-1 : '))
    angka2 = int(raw_input('masukkan angka ke-2 : '))

    print 'hasil perhitungan : ', angka1 / angka2

except ZeroDivisionError, e:
    print "proses pembagian gagal karena  : ", e
except ValueError, e:
    print "proses perhitungan gagal karena : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Pada kode diatas kita mencoba menjebak dua buah *exception* dengan dua baris *except* berbeda. Hal tersebut dilakukan agar perlakuan pada penanganan setiap *exception* memiliki penanganan yang berbeda. Misal pada baris **except** pembagian nol ada informasi “proses pembagian gagal karena : “ , sedangkan di baris **except** nilai *error* terdapat informasi “proses perhitungan gagal karena : “. Jadi dengan menggunakan baris **except** yang berbeda Anda dapat menangani *error* yang berbeda sesuai kebutuhan Anda.



<< gambar 8.5 hasil eksekusi exception_5.py >>

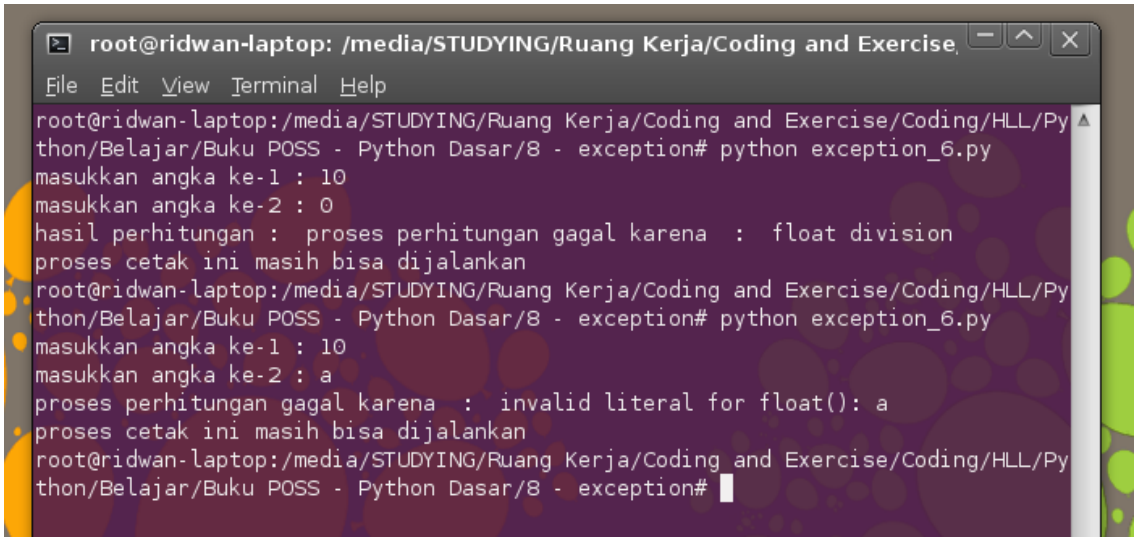
Menggunakan Multiple Exception

Berbeda sedikit pada contoh sebelumnya, jika pada setiap *exception* ditangani oleh setiap baris **except**. Maka pada kaidah *multiple exception* di satu **except** menangani beberapa *exception*. Bedanya, semua *exception* yang ditangani baris **except** tersebut akan mendapat penanganan yang sama.

listing : exception_6.py

```
try:

    angka1 = float(raw_input('masukkan angka ke-1 : '))
    angka2 = float(raw_input('masukkan angka ke-2 : '))

    print 'hasil perhitungan : ', angka1 / angka2

except (ZeroDivisionError, ValueError, TypeError),  e:
    print "proses perhitungan gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut :



<< gambar 8.6 hasil eksekusi exception_6.py >>

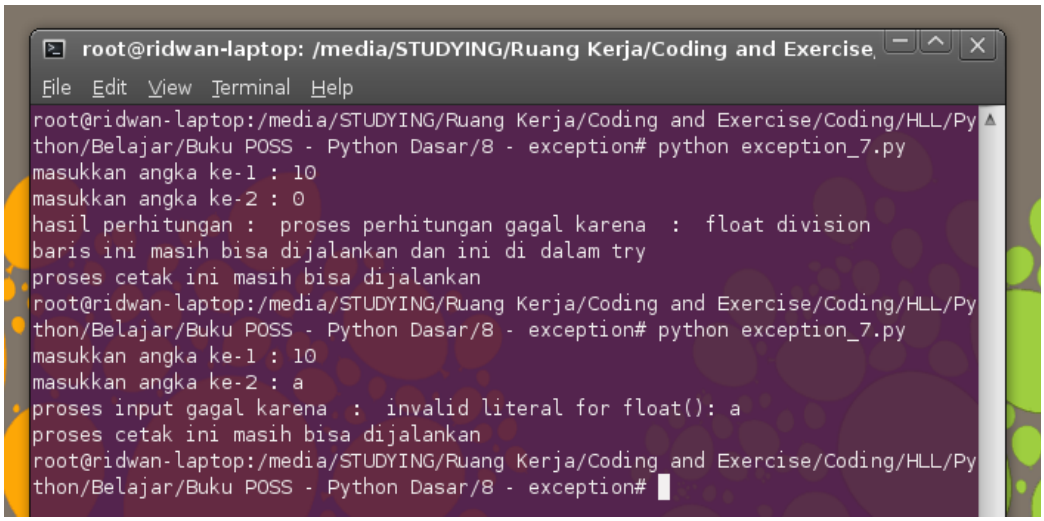
Try-Except Bersarang

Mirip seperti kondisional atau perulangan yang dapat ditambahkan blok kode kondisional atau perulangan didalamnya. *Try-except* pun mempunyai kaidah yang sama dimana *try-except* dapat disimpan didalam *try-except* yang lainnya. Prioritasnya adalah tangani yang luar terlebih dahulu. Jika terjadi di *try-except* terluar maka blok kode didalamnya yang terdapat *try-except* tidak akan dieksekusi. Jika di blok luar tidak terdapat *error*. Maka penanganan *exception* di *try-except* bagian dalam akan dilakukan.

listing : exception_7.py

```
try:

    angka1 = float(raw_input('masukkan angka ke-1 : '))
    angka2 = float(raw_input('masukkan angka ke-2 : '))

    try :
        print 'hasil perhitungan : ', angka1 / angka2
    except ZeroDivisionError, e:
        print "proses perhitungan gagal karena  : ", e

except ValueError,  e:
    print "proses input gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Jika pada contoh exception_5.py baris **except** ZeroDivisionError disimpan di tingkat pertama, sekarang baris tersebut disarangkan di *try-except* yang utama. Dengan demikian Anda dapat menangani *exception* dari dalam secara langsung.



<< gambar 8.7 hasil eksekusi exception_7.py >>

Membuat Exception Sendiri

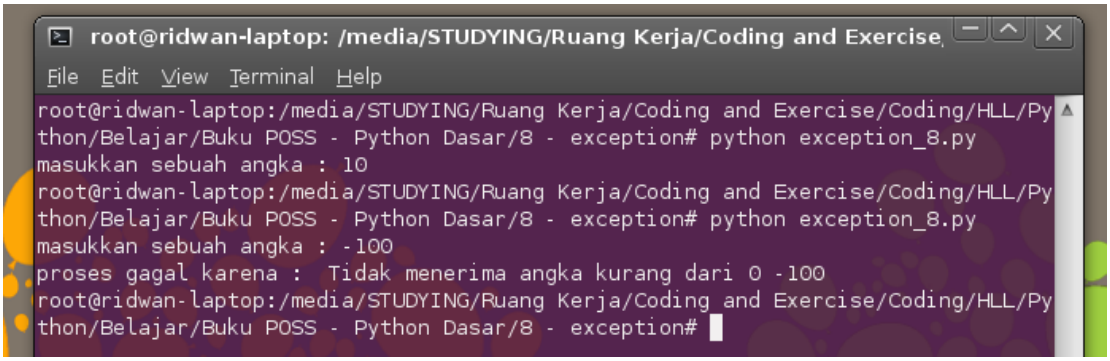
Lalu apakah kita terbatas pada penanganan *standard exception* Python saja ?. Anda dapat membuat *exception* Anda sendiri dengan membuat sebuah kelas yang diturunkan dari kelas Exception. Dengan cara tersebut, Anda dapat membuat *exception* Anda sesuai kebutuhan pada kasus yang akan Anda tangani. Misal kita ingin membuat sebuah *exception* jika angka yang dimasukkan adalah angka negatif. Pertama kita buat dulu **class** nya dengan nama yang diingikan, turunkan dari kelas Exception, dan tentukan penanganan *error* pada *method – method* di kelas tersebut.

listing : exception_8.py

```
class NegativeValueError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        s = "Tidak menerima angka kurang dari 0 " + str(self.value)
        return s

def cekAngka(angka):
    if angka < 0:
        raise NegativeValueError(angka)

try:
    sebuah_angka = int(raw_input("masukkan sebuah angka : "))
    cekAngka(sebuah_angka)
except (NegativeValueError, TypeError), e:
```

```
   print "proses gagal karena : ", e
```

Untuk memanggil *exception*-nya kita memerlukan *keyword* **raise** ketika *exception* tersebut dimunculkan maka *exception* akan ditangani **except** dan mengeluarkan pesan *error*-nya. Pesan tersebut berasal dari *function* __str__() yang sebelumnya telah kita definisikan pada kelas NegativeValueError.



<< gambar 8.8 hasil eksekusi exception_8.py >>

Menggunakan “finally” pada Try-Except

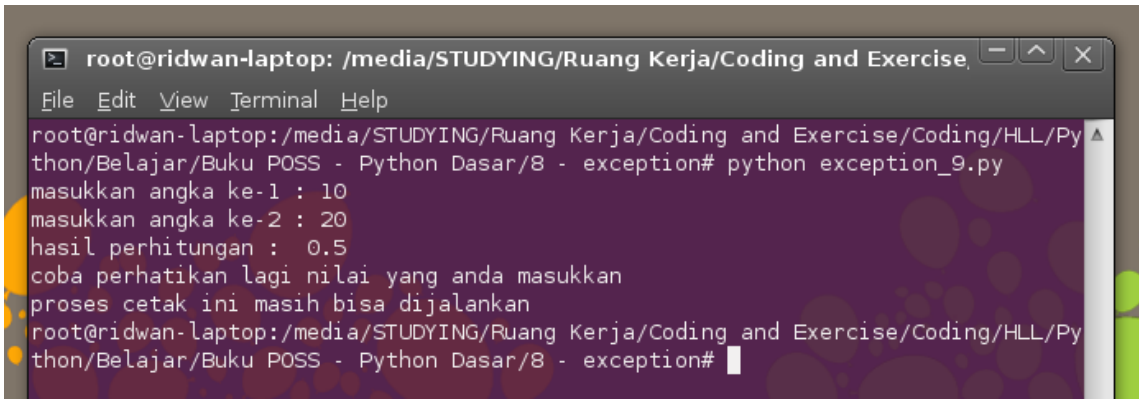
Dan akhirnya sekarang kita membahas **finally**. Keyword ini digunakan untuk menentukan penanganan apa yang harus dilakukan baik ketika *exception* muncul atau tidak. Misal saat mengambil data dari *database*, kita tidak akan tahu ada kegagalan apa yang akan terjadi. Agar program kita tetap aman dan data tidak rusak. Maka baik terjadi kegagalan atau tidak koneksi ke *database* harus ditutup. Hal tersebut juga bisa terjadi saat pembacaan *file*. Untuk mencegah kerusakan *file*, baik akan terjadi *error* atau tidak, *file* harus ditutup. Di blok **finally** ini penanganan yang terjadi ketika *exception* muncul atau tidak disimpan.

listing : exception_9.py

```
try:

   angka1 = float(raw_input('masukkan angka ke-1 : '))
   angka2 = float(raw_input('masukkan angka ke-2 : '))

   try :
      print 'hasil perhitungan : ', angka1 / angka2
   except ZeroDivisionError, e:
      print "proses perhitungan gagal karena  : ", e

except ValueError,  e:
   print "proses input gagal karena  : ", e
finally:
```

```
    print "coba perhatikan lagi nilai yang anda masukkan "


print "proses cetak ini masih dapat dijalankan  "
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut :



<< gambar 8.9 hasil eksekusi exception_9.py >>

9. Membuat File

Pengenalan File

Biasanya jika kita tidak menggunakan *file*, hasil pemrosesan data hanya akan disimpan di *main memory*. Setelah program dihentikan atau tiba – tiba komputer Anda mati, semua data akan hilang. Lalu bagaimana jika ingin menggunakan kembali data yang sudah diproses sebelumnya ?. Untuk menyimpan data agar bisa diproses di kesempatan selanjutnya, misal komputer dimatikan kemudian dinyalakan lagi hari esoknya. Kita butuh sebuah penyimpanan yang bersifat resident dan disimpan di *secondary storage* seperti *harddisk*. Python sendiri menyediakan beberapa media penyimpanan yang bisa digunakan oleh *programmer* Python, ada *file*, **shelve**, **marshal**, **pickle**, dan **sqlite3**.

Pada bab ini kita akan bahas mengenai *file* berupat txt. *File* di Python bisa berupa txt, csv, atau jenis lainnya. Txt merupakan contoh *file* yang sering digunakan. *File* jenis ini berisi *plain text*. *File* jenis ini menyimpan karakter ascii *standard* yang diterima dari *user*.

Pada pembuatan *file* terdapat beberapa mode dalam manipulasi *file*. Berikut daftar mode manipulasi *file* tersebut :

No	Mode	Keterangan
1	r	Membuka *file* dan hanya untuk pembacaan saja. *Pointer file* akan ditempatkan di awal *file*. Jika pada saat pembukaan *file* tidak disertakan mode manipulasi *file*, maka mode

		ini secara *default* dipakai untuk manipulasi *file*
2	w	Membuka *file* dan hanya untuk penulisan saja. Jika *file* yang dibuka sudah ada dan menggunakan mode 'w', maka *file* tersebut akan ditimpa. Jika *file* tidak ada maka akan dibuatkan *file* baru.
3	a	Membuka *file* untuk penambahan isi *file*. *Pointer file* disimpan di bagian akhir *file* jika *file* tersebut ada. Jika *file* tidak ada maka akan dibuatkan *file* baru.
4	b	Mode ini ditambahkan masing – masing pada mode r, w,a menjadi rb, wb, ab. Dengan menambahkan mode b pada setiap mode manipulasi standar. Maka pemba *file* caan *file* akan dilakukan dalam format biner
5	+	Mode ini ditambahkan kedalam mode r, w, a. • r+ : baca dan tulis • w+ : tulis dan baca • a+ : tambah dan baca

Membuat File Baru

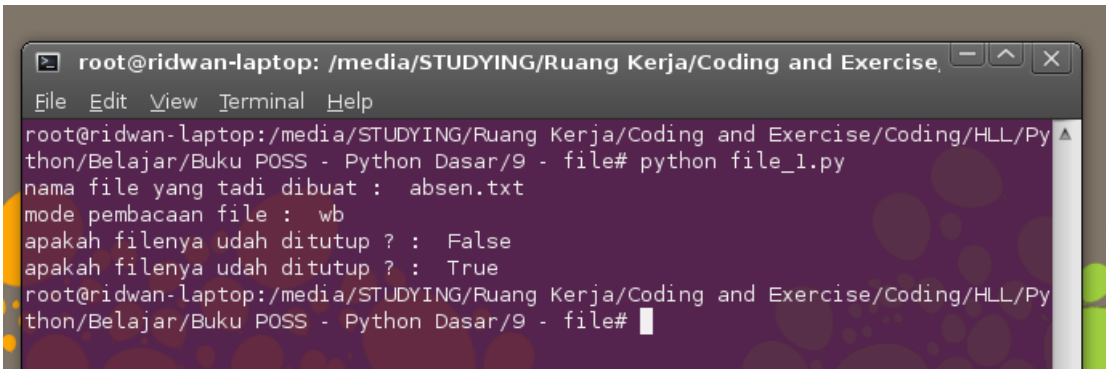
Agar lebih paham kita akan mencoba membuat sebuah *file* txt. Pertama kita biasakan cegah *error* dengan menggunakan *try-except* dan tangkap *exception* IOError, agar jika terjadi *error* kelak, kita bisa menanganinya dengan lebih mudah. Kemudian di dalam blok *try-except* tersebut buat sebuah objek *file* dengan menggunakan *built-in function* open(). Pada *function* tersebut terdapat dua parameter yang biasa digunakan yaitu, nama *file* serta mode manipulasi *file*-nya. Karena kita ingin membuat *file* baru, maka mode “w” digunakan pada kasus pertama berikut :

listing : file_1.py

```
try:
   sebuah_file = open("absen.txt", 'w')

   print "nama file yang tadi dibuat : ", sebuah_file.name
   print "mode pembacaan file : ", sebuah_file.mode
   print "apakah filenya udah ditutup ? : ", sebuah_file.closed

   sebuah_file.close()
   print "apakah filenya udah ditutup ? : ", sebuah_file.closed
except IOError, e:
   print "proses gagal karena : ", e
```

Ketika kita sudah membuka sebuah *file* dan terbentuk objek *file*. Kita dapat mengakses *method* dan

atribut pada objek *file* tersebut. Atribut yang sering diakses untuk pemrosesan *file* antara lain : name, mode, closed. Atribut name adalah nama *file* tersebut, mode adalah mode manipulasi *file* tersebut, dan closed menyatakan apakah *file* tersebut sudah ditutup atau belum. Sedangkan *method* yang diakses diatas adalah close(), yang digunakan untuk menutup *file* setelah penggunaan *file* selesai. Dengan menutup *file*, penggunaan memori utama akan dihemat. Jika tidak pernah menutup *file* dalam jumlah yang banyak bisa menyebabkan *memory leak*. Jadi berhati – hatilah :D.



<< gambar 9.1 hasil eksekusi file_1.py >>

Mengisi File

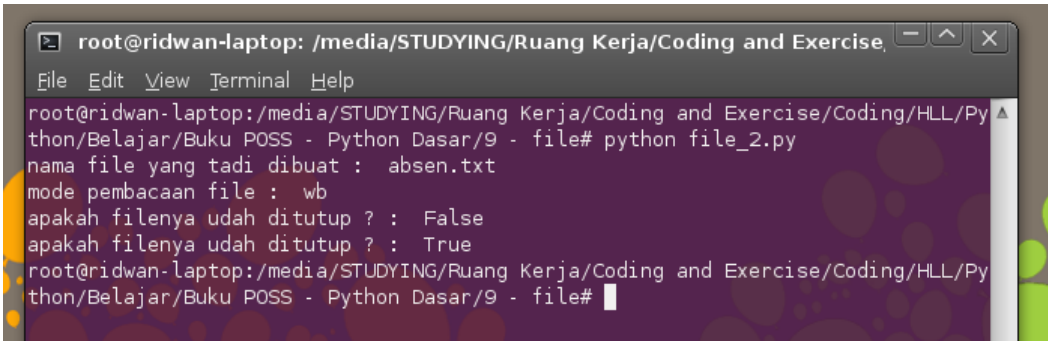
Pada contoh pertama, *file* yang kita buat masih kosong, belum berisi. Sesuai namanya kita sedang membuat *file* bernama 'absen.txt', yang didalamnya akan terdapat daftar hadir perkuliahan. Dengan menggunakan *method* write(), kita bisa menambahkan isi pada *file* 'absen.txt', dan yang akan kita isikan adalah teks. *Method* ini memerlukan parameter sebuah *string* yang akan ditulis di lokasi tertentu pada *file* berdasarkan posisi *pointer file* berada.

listing : file_2.py

```
try:
    sebuah_file = open("absen.txt", 'w')

    print "nama file yang tadi dibuat : ", sebuah_file.name
    print "mode pembacaan file : ", sebuah_file.mode
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed

    sebuah_file.write('1. Jajang Surahman, Teknik Informatika, ITENAS\n')
    sebuah_file.write('2. Angel Corine, Manajemen Informatika, UNIKOM\n')
    sebuah_file.write('3. Samsul Basri, Ilmu Komputer, UPI\n')

    sebuah_file.close()
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed
```

```
except IOError, e:
    print "proses gagal karena : ", e
```

Setelah kita menambahkan isi pada *file* teks yang kita buat, kita dapat membuka *file* yang telah dibuat dengan teks *editor* dan dapat melihat isi dari *file* tersebut.



<< gambar 9.2 hasil eksekusi file_2.py >>

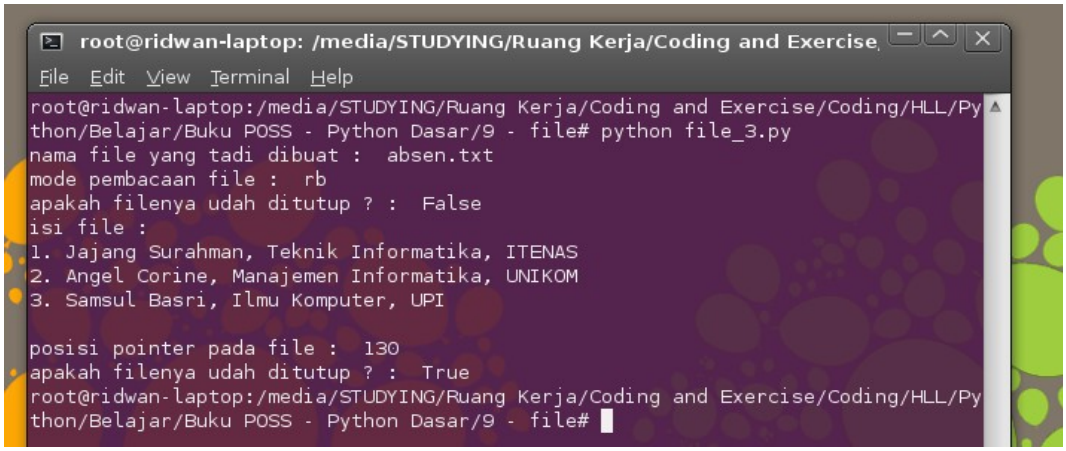
Membaca Isi File

Setelah mengisi *file* dengan *method* write(). Sekarang kita akan menggunakan *method* read() untuk membaca *file*. Pastikan, *file* yang akan dibaca harus dalam mode 'r', jika tidak dalam mode tersebut, misal dalam mode 'w', maka akan muncul *error* : ”OError: File not open for reading”. Kemudian untuk mengetahui posisi *pointer file* berada, kita gunakan *method* tell().

listing : file_3.py

```
try:
    sebuah_file = open("absen.txt", 'r')

    print "nama file yang tadi dibuat : ", sebuah_file.name
    print "mode pembacaan file : ", sebuah_file.mode
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed

    print "isi file : \n", sebuah_file.read()
    print "posisi pointer pada file : ", sebuah_file.tell()

    sebuah_file.close()
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed
except IOError, e:
    print "proses gagal karena : ", e
```

Dengan menggunakan *method* read(), kita dapat melihat isi dari *file* tersebut. Tapi *method* ini membaca sekaligus isi *file* yang dibaca, tidak perbaris. Jika pembacaan dilakukan sekaligus, ruang memori yang dibutuhkan jauh lebih besar daripada *file* yang dibaca perbaris. Kemudian dengan adanya *method* tell(), kita bisa mengetahui posisi *pointer file* berada dimana, agar mempermudah saat manipulasi *file*.



<< gambar 9.3 hasil eksekusi file_3.py >>

Membaca Isi File dengan Cara Baris Per Baris

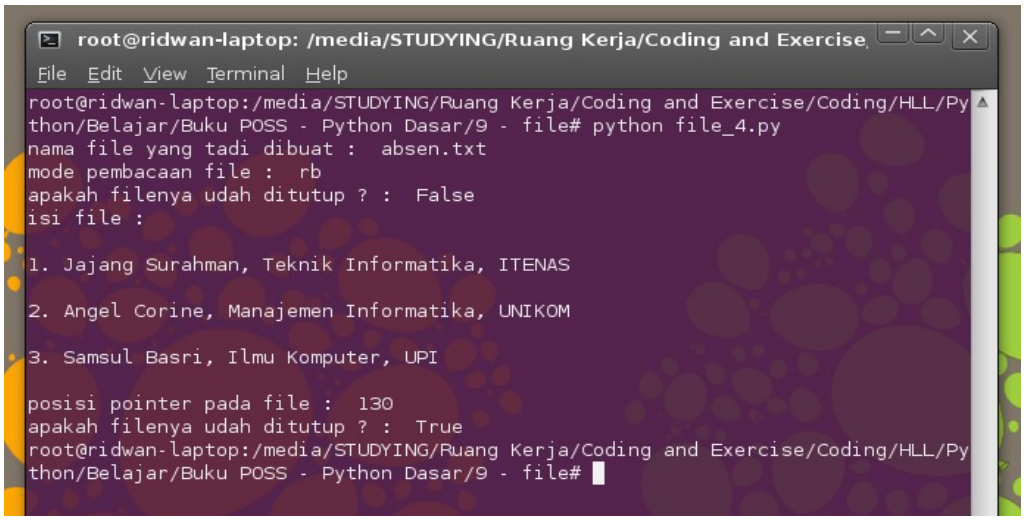
Jika pada contoh sebelumnya pembacaan *file* dilakukan sekaligus, pada contoh kali ini pembacaan *file* akan dilakukan baris perbaris. Pembacaan *file* teks dengan membaca perbaris ini bisa dilakukan dengan menggunakan pengulangan **for**. *File* ini diperlakukan layaknya **list** yang digunakan di pengulangan **for**. Disini *file* dianggap sebagai **list** yang berisi elemen *string*.

listing : file_4.py

```
try:
    sebuah_file = open("absen.txt", 'r')

    print "nama file yang tadi dibuat : ", sebuah_file.name
    print "mode pembacaan file : ", sebuah_file.mode
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed

    print "isi file : \n"

    for line in sebuah_file:
        print line
```

```
    print "posisi pointer pada file : ", sebuah_file.tell()

    sebuah_file.close()
    print "apakah filenya udah ditutup ? : ", sebuah_file.closed
except IOError, e:
    print "proses gagal karena : ", e
```

Hasil yang diperlihatkan hampir sama dengan contoh sebelumnya hanya saja teknik pembacaannya sedikit berbeda. Jika *file* berukuran besar, akan lebih bijak jika kita membacanya perbaris agar ruang memori yang digunakan tidak banyak terpakai.



<< gambar 9.4 hasil eksekusi file_4.py >>

Mengatur Posisi Pointer File

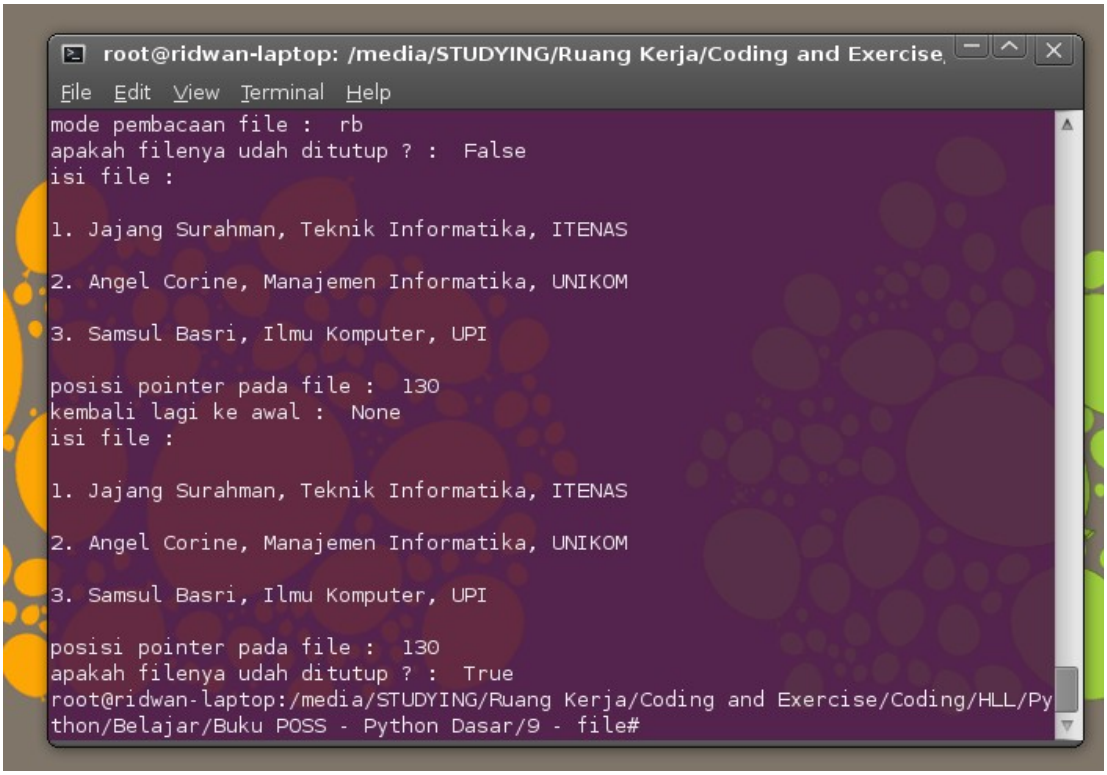
Suatu saat kita ingin mengisi *file* di lokasi tertentu di sebuah *file*. Biasanya kita menambahkan *file* di bagian akhir *file*, atau membaca *file* selalu dibagian awal *file*. Ada saatnya kita ingin membaca di posisi ke 15 dari awal *file*, atau posisi -15 karakter dari *pointer file*. Di objek *file* terdapat *method* yang dinamakan seek(). *Method* tersebut memiliki dua buah paramater yaitu jarak yang diinginkan dan patokan jarak tersebut. Jika parameter kedua diisi oleh angka 0, berarti patokan berada di awal file. Jika parameter kedua diisi oleh angka 1, berarti patokan berada di tempat pointer *file* berada. Jika parameter kedua diisi oleh angka 2, maka patokan berada di bagian akhir *file*. Jika parameter pertama diisi angka positif maka penentuan jarak akan dihitung ke sebelah kanan, jika diisi angka negatif maka penentuan jarak akan dihitung ke sebelah kiri.

listing : file_5.py

```
try:
    sebuah_file = open("absen.txt", 'rb')

    print "nama file yang tadi dibuat : ", sebuah_file.name
    print "mode pembacaan file : ", sebuah_file.mode
```

```
   print "apakah filenya udah ditutup ? : ", sebuah_file.closed

   print "isi file : \n"
   for line in sebuah_file:
      print line

   print "posisi pointer pada file : ", sebuah_file.tell()
   print "kembali lagi ke awal : ", sebuah_file.seek(0, 0)

   print "isi file : \n"
   for line in sebuah_file:
      print line

   print "posisi pointer pada file : ", sebuah_file.tell()

   sebuah_file.close()
   print "apakah filenya udah ditutup ? : ", sebuah_file.closed
except IOError, e:
   print "proses gagal karena : ", e
```

Pada contoh diatas, *pointer file* dipindahkan kembali ke posisi awal. Dengan memberikan jarak 0, dan menentukan patokan di awal *file*, maka posisi *pointer file* pindah ke bagian awal *file*. Dengan demikian *file* bisa dibaca ulang untuk kedua kalinya.



<< gambar 9.5 hasil eksekusi file_5.py >>

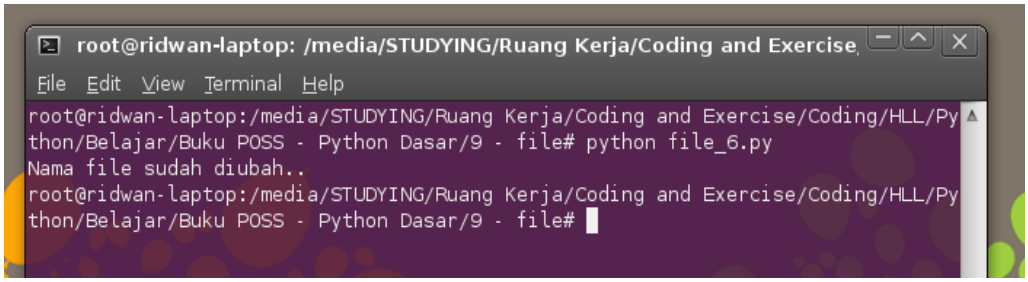
Mengganti Nama File

Dalam manipulasi *file*, terdapat operasi seperti pengubahan nama *file*, memindahkan *file*, ataupun menghapus *file*. Python sendiri menyediakan module **os** yang didalamnya terdapat fitur- fitur tersebut. Sebagai contoh pertama kita akan mengganti nama *file* dari "absen.txt" ke "daftar-hadir.txt". Pertama kita harus meng-*import* modul **os**. Kemudian kita gunakan *method* rename(). *Method* tersebut memiliki dua parameter yaitu nama *file* yang akan diubah namanya, dan nama baru yang diinginkan.

listing : file_6.py

```
import os

try:
    os.rename('absen.txt', 'daftar-hadir.txt')
    print "Nama file sudah diubah.."
except (IOError, OSError), e:
    print "proses error karena : ", e
```

Setelah kode diatas dijalankan, coba lihat *file* yang sebelumnya "absen.txt" akan berubah menjadi "daftar-hadir.txt".



<< gambar 9.6 hasil eksekusi file_6.py >>

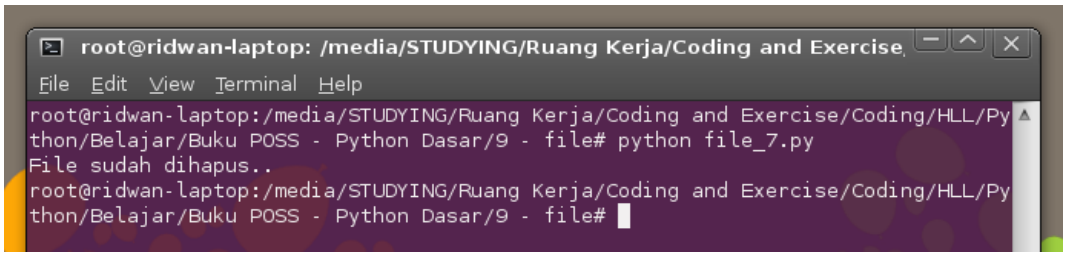
Menghapus File

Contoh lainnya adalah menghapus *file*. Kita bisa gunakan *method* remove() untuk menghapus *file* yang diinginkan. Parameter yang dibutuhkan adalah nama *file* yang akan dihapus.

listing : file_7.py

```
import os

try:
    os.remove('daftar-hadir.txt')
    print "File sudah dihapus.."
except (IOError, OSError), e:
    print "proses error karena : ", e
```

Kemudian jika *file* sudah dihapus, kita tidak dapat membuka *file* tersebut. Karena *file* tersebut sudah hilang dari penyimpanan sekunder.



<< gambar 9.7 hasil eksekusi file_7.py >>

Untuk contoh – contoh lainnya Anda bisa membuka dokumentasi resmi Python, atau coba kunjungi beberapa website yang ada di lampiran – 2.

10. Pengenalan Class

Membuat Class dan Object

Pemrograman berorientasi objek atau dalam bahasa inggris dikenal dengan *Object Oriented Programming* (OOP), merupakan sebuah paradigma pemrograman dimana kita memodelkan perangkat lunak kita dari berbagai kumpulan objek yang saling berinteraksi. Objek tersebut memiliki karakteristik dan aksi. Di dalam OOP, karakteristik tersebut berupa variabel yang dinamakan atribut. Kemudian aksi yang dimiliki objek tersebut berupa method yang menghasilkan output atau cuma melakukan aksi saja tanpa output. Ada istilah lain juga yang menyebut aksi sebuah objek sebagai perilaku atau *behaviour*. Objek itu sendiri mempunyai *template* yang diistilahkan dengan kelas atau *class*. Sebuah kelas merupakan *template* bagi objek – objek yang akan dibuat. Proses pembuatan objek baru dinamakan dengan instansiasi.

Ada beberapa hal yang harus diingat dalam membuat sebuah kelas. Pertama *keyword* **class**, *keyword* ini digunakan untuk mendefinisikan sebuah kelas. Disusul dengan nama kelas yang diinginkan dan tanda titik dua. Blok kode kelas tersebut ditulis setelah tanda titik dua tersebut, dan seperti biasa diperlukan indentasi agar blok kode yang kita tulis masuk kedalam blok kode kelas. Kedua adalah konstruktor, di Python konstruktor ditulis dengan sebuah *function* bernama __init__() . *Method* dan *function* sebenarnya sama, hanya saja beda istilah pada paradigma OOP. *Method* __init__() ini merupakan *method* yang akan selalu dieksekusi saat instansiasi objek. Biasanya __init__() digunakan untuk mengisi variabel dengan nilai awal pada atribut – atribut yang dimiliki objek. Ketiga adalah *keyword* **self,** *keyword* tersebut digunakan pada *method* yang akan dinyatakan sebagai *method* dari kelas yang kita rancang. *Keyword* ini disertakan di parameter pertama. Jika *method* tersebut tidak disertakan **self** pada *method* yang dimiliki kelas tersebut, akan muncul *error* : “TypeError: **nama_function()** takes exactly **n** arguments (1 given)
” yang artinya *method* tersebut tidak bisa dipanggil oleh objek yang telah terinstansiasi.

Sebagai contoh disini kita akan membuat sebuah kelas bernama PersegiPanjang yang memiliki dua atribut yaitu panjang dan lebar. Kelas ini memiliki lima method yang terdiri dari :

- __init__(), konstruktor kelas persegi panjang
- hitung_luas(), method untuk menghitung luas persegi panjang
- hitung_keliling(), method untuk menghitung keliling persegi panjang
- gambar_persegi_panjang(), menggambar persegi panjang yang direpresentasikan dengan kumpulan bintang.
- gambar_persegi_panjang_tanpa_isi(), menggambar persegi panjang tetapi hanya batas luarnya saja, isinya tak digambar.

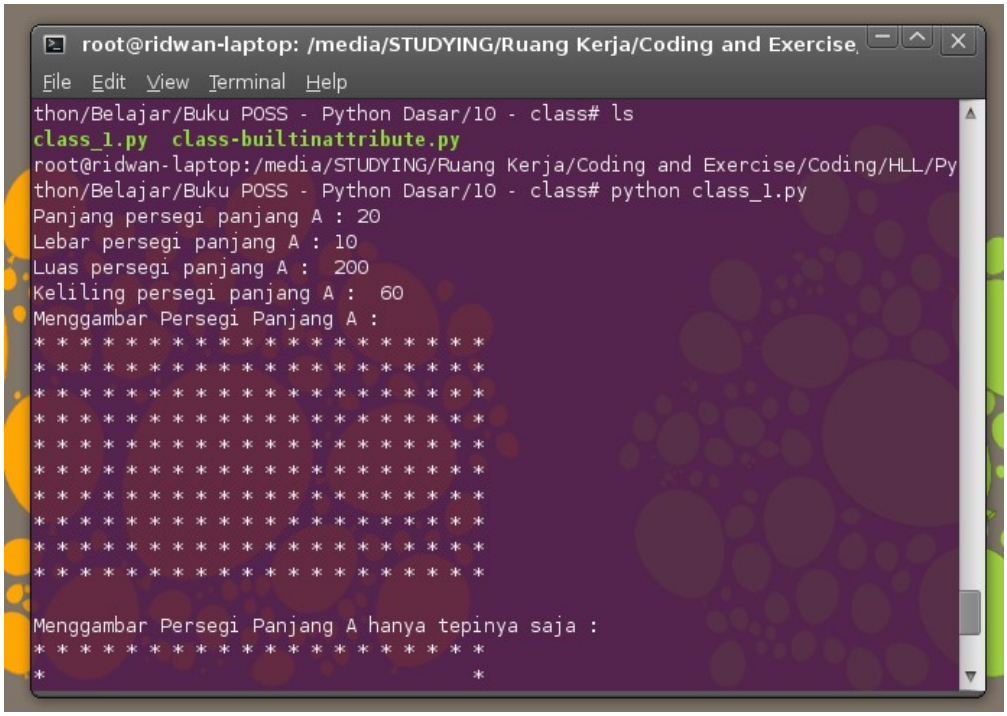
listing : class_1.py

```
class PersegiPanjang:

    def __init__(self, panjang, lebar):
        self.panjang = panjang
        self.lebar = lebar

    def hitung_luas(self):
        return self.panjang * self.lebar

    def hitung_keliling(self):
        return (2*self.panjang) + (2*self.lebar)

    def gambar_persegi_panjang(self):
        for i in range(0, self.lebar):
            for j in range(0, self.panjang):
                print '*',
            print ""

    def gambar_persegi_panjang_tanpa_isi(self):
        for i in range(0, self.lebar):
            if i > 0 and i < self.lebar-1:
                for j in range(0, self.panjang):
                    if j > 0 and j < self.panjang-1:
                        print '  ',
                    else:
                        print '*',
            else:
                for j in range(0, self.panjang):
                    print '*',

            print ""


PersegiPanjangA = PersegiPanjang(20, 10)
PersegiPanjangB = PersegiPanjang(10, 5)

print "Panjang persegi panjang A :", PersegiPanjangA.panjang
print "Lebar persegi panjang A :", PersegiPanjangA.lebar
print "Luas persegi panjang A : ", PersegiPanjangA.hitung_luas()
print "Keliling persegi panjang A : ", PersegiPanjangA.hitung_keliling()
print "Menggambar Persegi Panjang A : "
PersegiPanjangA.gambar_persegi_panjang()

print "\nMenggambar Persegi Panjang A hanya tepinya saja : "
PersegiPanjangA.gambar_persegi_panjang_tanpa_isi()

print "\n"

print "Panjang persegi panjang B :", PersegiPanjangB.panjang
print "Lebar persegi panjang B :", PersegiPanjangB.lebar
print "Luas persegi panjang B : ", PersegiPanjangB.hitung_luas()
```

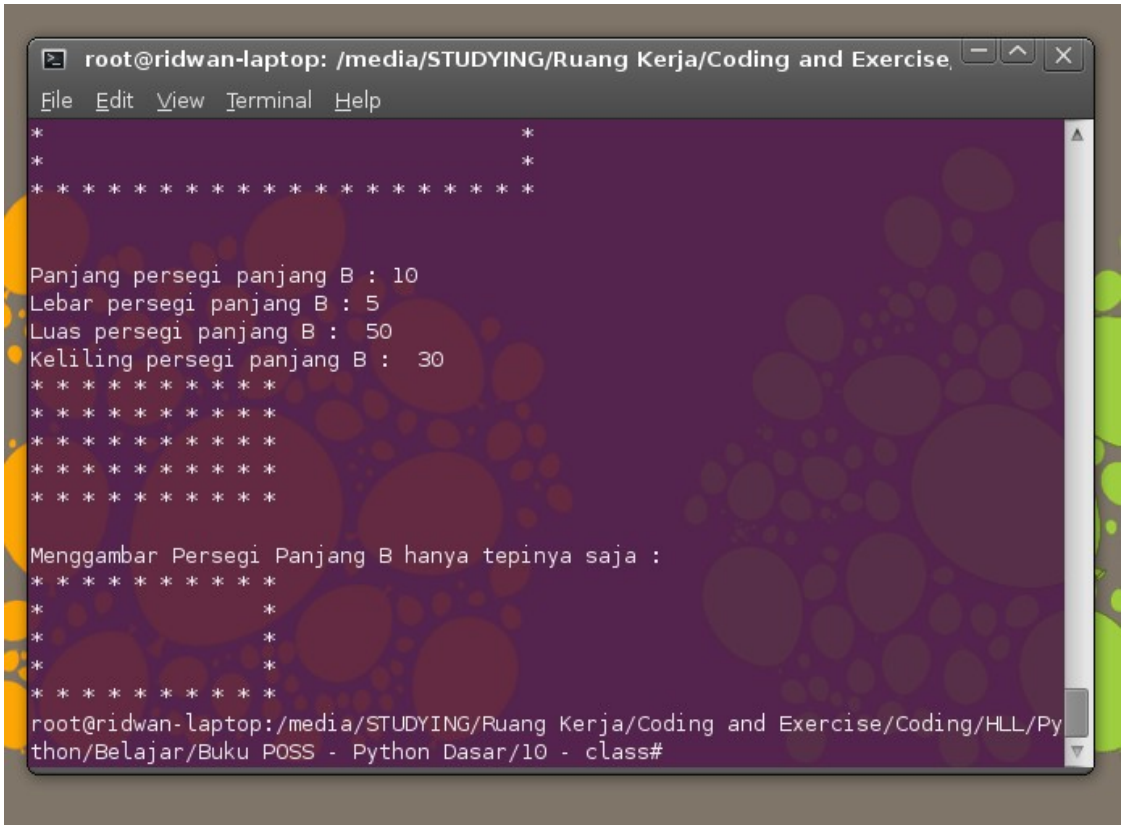
```
print "Keliling persegi panjang B : ", PersegiPanjangB.hitung_keliling()
PersegiPanjangB.gambar_persegi_panjang()
print "\nMenggambar Persegi Panjang B hanya tepinya saja : "
PersegiPanjangB.gambar_persegi_panjang_tanpa_isi()
```

Proses instansiasi dilakukan dengan menentukan nama objek yang diinginkan kemudian panggil nama kelas yang akan diinginkan dan masukan parameter awal yang dibutuhkan. Kemudian saat objek berhasil dibuat, kita bisa mengakses atribut dan *method* dari objek tersebut. Seperti yang kita lakukan pada manipulasi *file*, kita bisa mengakses *method* close(), write(), dan read() serta mengakses atribut **close**, **name**, dan **mode**. Dengan parameter berbeda namun karakterisitik sama, persegi panjang yang dihasilkan jadi beragam.

Tujuan dari OOP ini sendiri, menghemat penulisan kode program yang kita buat. Tanpa OOP kita harus membuat atribut untuk setiap objek. Dan penulisan kode program pun menjadi menumpuk, Karena untuk *method* yang sama harus di tulis ulang.



<< gambar 10.1 hasil eksekusi class_1.py bagian ke - 1 >>



<< gambar 10.2 hasil eksekusi class_1.py bagian ke - 2 >>

Mengenal Built-in Function pada Class dan Object

Berikutnya kita akan mengenal beberapa *built-in class attribute* yang akan bisa digunakan saat kita membuat kelas apapun. *Built-in class attribute* akan selalu menyertai kelas yang kita rancang. Berikut beberapa atribut yang bisa Anda gunakan untuk mengakses informasi dari sebuah kelas :

- __doc__, digunakan untuk mengakses dokumentasi yang terdapat pada kelas
- __name__, digunakan untuk mengakses nama kelas
- __dict__, digunakan untuk mendapatkan namespace dari kelas tersebut. Kalau pada objek yang sudah diinstansiasi method ini akan mengeluarkan informasi tentang atribut yang sudah terisi nilai
- __module__, digunakan untuk mendapatkan informasi dimana lokasi modul yang mendefinisikan kelas tersebut
- __bases__, digunakan untuk melihat darimana kelas tersebut diwariskan. Pewarisan pada OOP adalah menggunakan karakteristik suatu kelas pada kelas yang ingin menggunakan karakteristik kelas yang mewariskannya.

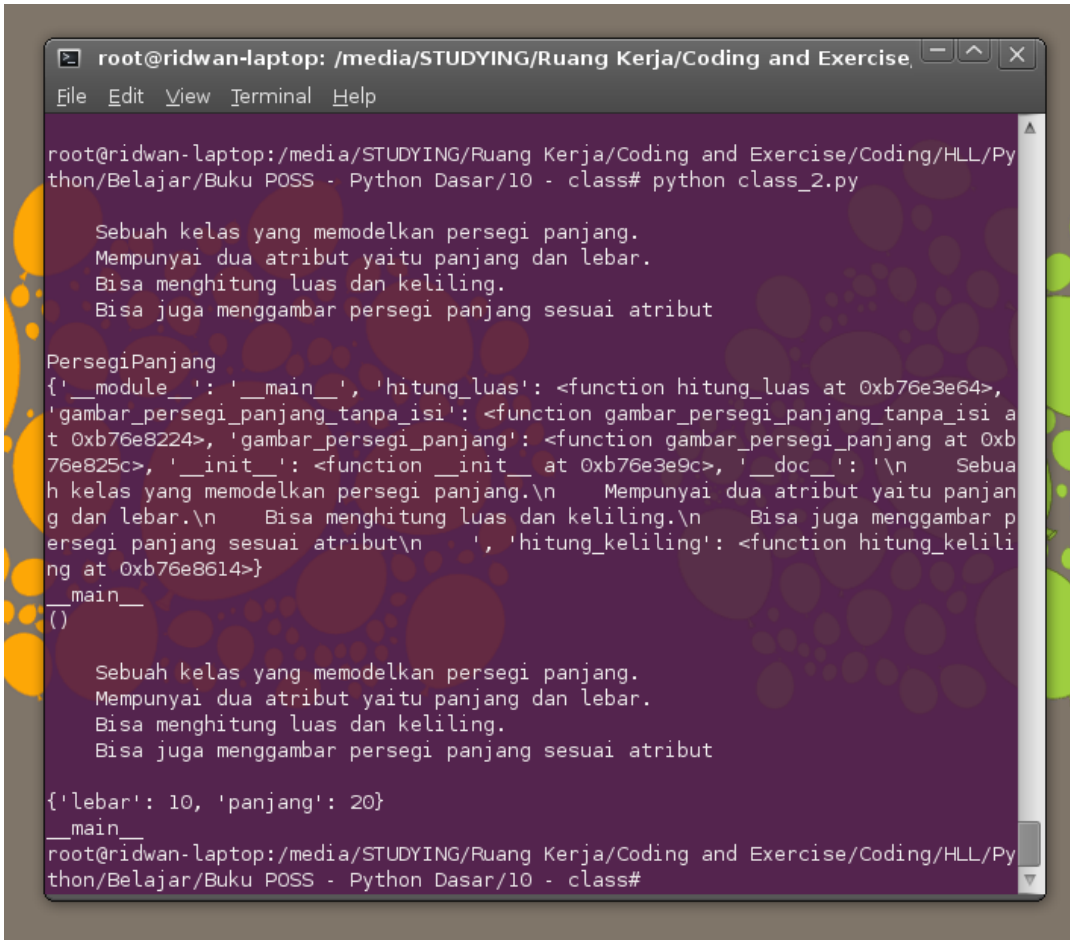
Sebagai contoh ada beberapa *built-in class attribute* yang bisa diakses kelas dan objek hasil instansiasi dan ada yang hanya bisa diakeses kelas.

listing : class_2.py

```
class PersegiPanjang:
    '''
    Sebuah kelas yang memodelkan persegi panjang.
    Mempunyai dua atribut yaitu panjang dan lebar.
    Bisa menghitung luas dan keliling.
    Bisa juga menggambar persegi panjang sesuai atribut
    '''
    def __init__(self, panjang, lebar):
        self.panjang = panjang
        self.lebar = lebar

    def hitung_luas(self):
        return self.panjang * self.lebar

    def hitung_keliling(self):
        return (2*self.panjang) + (2*self.lebar)

    def gambar_persegi_panjang(self):
        for i in range(0, self.lebar):
            for j in range(0, self.panjang):
                print '*',
            print ""

    def gambar_persegi_panjang_tanpa_isi(self):
        for i in range(0, self.lebar):
            if i > 0 and i < self.lebar-1:
                for j in range(0, self.panjang):
                    if j > 0 and j < self.panjang-1:
                        print '-',
                    else:
                        print '*',
            else:
                for j in range(0, self.panjang):
                    print '*',

            print ""


PersegiPanjangA = PersegiPanjang(20, 10)

print PersegiPanjang.__doc__
print PersegiPanjang.__name__
print PersegiPanjang.__dict__
print PersegiPanjang.__module__
print PersegiPanjang.__bases__
```

```
print PersegiPanjangA.__doc__
print PersegiPanjangA.__dict__
print PersegiPanjangA.__module__
```

Pada contoh diatas, atribut __name__ dan __bases__ hanya bisa diakses oleh kelas. Sedangkan objek hasil instansiasi tidak bisa mengaksesnya.



<< gambar 10.3 hasil class_2.py >>

Pembahasan mengenai OOP ini tidak bisa dibahas secara keseluruhan dalam satu bab. Ada banyak hal yang harus diulas seperi *inheritance, polymorphism, abstract, overriding, overload,* dan lain – lain.

11. Pengenalan Module

Module dan Packages

Module adalah istilah *file* yang berisi *kode* Python. Jadi dari awal sebenarnya kita sudah membuat *module* Python. Hanya saja pada konsep *module* ini, kode Python yang akan digunakan berulang akan dipisahkan dari *file* utama ke dalam *file* lain yang khusus menampung kode Python tersebut. Di dalam *module* kita bisa menyimpan *class*, *function*, variabel, dan struktur data yang bisa digunakan oleh program. Misal kita ingin membuat sebuah kode yang hanya berisi jenis – jenis segitiga seperti segitiga sama kaki, segitiga sembarang, segitiga sama sisi, dan segitiga siku – siku. Kenapa tidak dicampur saja dengan jenis bidang yang lain ? Hal ini dilakukan agar kita mudah dalam mengelola kode Python yang kita tulis. Contoh lainnya misal kita menulis kode yang berinteraksi dengan database dan kode untuk melakukan proses penulisan laporan secara terpisah.

Dalam hal ini *module* mempunyai kode Python yang *reusable* agar kode yang ditulis pada program kita terduplikasi. Sedangkan *file* Python yang akan dijalankan dan memanggil *function*, *class*, atau variabel dari kumpulan *module* yang dibuat berisi *runnable code*. Kode yang dieksekusi oleh *interpreter* Python untuk menampilkan wujud dari program yang dibuat.

Kemudian *module – module* yang sudah ditulis bisa dikelompokan kedalam sebuah *package*. *Package* ini sendiri berupa *folder* yang memiliki file __init__.py, agar *folder* tersebut dikenali sebagai *module*. Di dalam *package* ini *module – module* memiliki tujuan dan fungsional yang seragam. Misal pada contoh yang akan kita coba, terdapat sebuah *package* bidang, yang berisi module bidang segitiga dan persegi. Di dalamnya terdapat *file* __init__.py yang bertugas untuk me-*load* semua *module* yang ada di dalam *package*, segitiga.py yang berisi *class* segitiga, dan persegi.py yang berisi *class* persegi. Di dalam *file* segitiga.py dan persegi.py masing – masing bisa diisi berbagai jenis bidang yang sesuai nama *module* tersebut. Hanya saja untuk contoh kali ini dibatasi kepada satu jenis bidang saja

Membuat Module – Module di dalam Packages

Setelah memahami konsep module, mari kita coba program yang agak banyak ini. Sebelumnya di direktori tempat kita akan menulis program, terlebih dahulu buatlah sebuah folder baru bernama bidang. Folder tersebut merupakan package yang akan menyimpan persegi.py, segitiga.py, dan __init__.py.

listing : persegi.py

```
class Persegi:
    def __init__(self, s):
        self.sisi = s

    def SetSisi(self, s):
        self.sisi = s

    def GetSisi(self):
        return self.sisi
```

```
    def HitungKeliling(self):
        return 4 * self.sisi

    def HitungLuas(self):
        return self.sisi * self.sisi
```

Kode persegi.py diatas hanya bersegi *class* Persegi yang mempunyai atribut sisi dan *method* – *method*-nya. Di dalam *module* ini kita bisa saja menulis kelas PersegiPanjang. Hal tersebut memudahkan kita agar bidang yang jenisnya persegi tidak tercampur dengan bidang yang jenisnya segitiga. Pastikan Anda menyimpan *file* persegi.py di dalam *folder* bidang.

listing : segitiga.py

```
import math


class Segitiga:
    def __init__(self, a, t):
        self.alas = a
        self.tinggi = t

    def SetAlas(self, a):
        self.alas = a

    def GetAlas(self):
        return self.alas

    def SetTinggi(self, t):
        self.tinggi = t

    def GetTinggi(self):
        return self.tinggi

    def GetSisiMiring(self):
        return math.sqrt(self.alas**2 + self.tinggi**2)

    def HitungKeliling(self, s):
        return self.alas + self.tinggi + s

    def HitungLuas(self):
        return (self.alas * self.tinggi) / 2
```

Hampir sama dengan fungsi dari *module* persegi.py, hanya saja *module* segitiga.py akan diisi berbagai jenis segitiga. Selain itu pada kode diatas kita memanggil *module* **math** karena saat nanti module segitiga.py ini diload, kode yang menggunakan method – method pada math harus di load juga dari module **math**. Pastikan module ini tersimpan di folder bidang.

listing : __init__.py

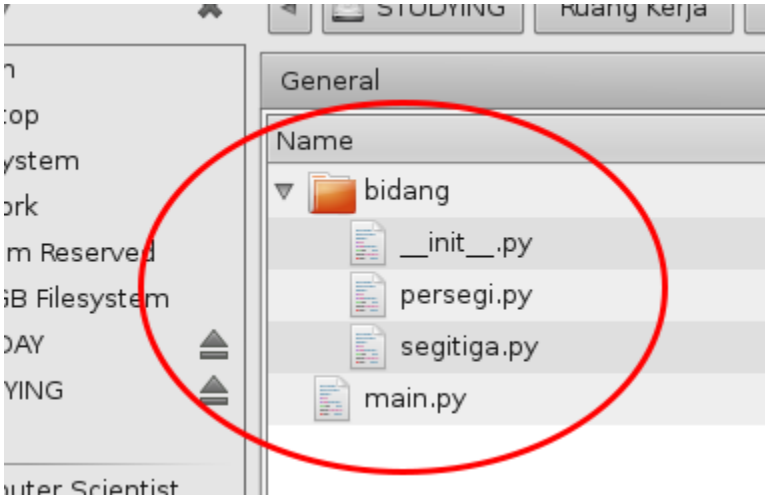
```
from segitiga import Segitiga
from persegi import Persegi

if __name__ == '__main__':
    pass
```

Kemudian *file* yang mesti ada di dalam sebuah *package* adalah __init__.py. *File* tersebut berfungsi untuk me-*load* isi module ke dalam memori agar isi *module* bisa digunakan di *file* yang berisi *runnable code*. Pada kode diatas, terdapat sintaks : **from** segitiga **import** Segitiga. *Keyword* **from** adalah *keyword* yang digunakan untuk menentukan *package* atau *module* mana yang akan kita rujuk, sedangkan *import* digunakan untuk mengambil *class*, *function* atau variabel yang didefinisikan di dalam *module*. Disana kita meng-*import* dua buah kelas yaitu Segitiga dan Persegi dari dua *module* berbeda yaitu segitiga.py dan persegi.py. Sedangkan kode dibawahnya digunakan jika *file* __init__.py ingin menjalankan perintah tertentu. Pastikan *file* ini disimpan di *folder* bidang.

Menggunakan Module di File Utama

Sampai akhirnya kita tiba untuk menulis kode utama. Kode utama ini merupakan kode yang berisi *runnable code*, dan menggunakan *class* yang sudah didefinisikan di *module – module* sebelumnya. Dengan demikian kode program tidak akan menumpuk di *file* utama.

Jika Anda berhasil mengikuti petunjuk pada bab ini, *module, packages* dan *file* utama harus mempunyai sususan seperti berikut :



<< gambar 12.1 struktur module packages bidang >>

listing : main.py

```
from bidang import Segitiga, Persegi

sgtgA = Segitiga(3, 9)
prsgA = Persegi(5)

print "Luas Segitiga A : ", sgtgA.HitungLuas()
print "Sisi Miring Segitiga A : ", sgtgA.GetSisiMiring()
print "Keliling Segitiga A : ", sgtgA.HitungKeliling(sgtgA.GetSisiMiring())
print "\n"
print "Luas Persegi A : ", prsgA.HitungLuas()
print "Keliling Segitiga A : ", prsgA.HitungKeliling()
```

Pada kode diatas kita meng-*import* kelas dari *package* bidang. Kemudian melakukan instansiasi dan memberikan nilai sesuai yang kita inginkan. Kemudian kita akses *method – method* yang terdapat pada kelas tersebut untuk mendapatkan informasi luas, dan keliling pada masing – masing bidang. Jika berhasil maka kode yang akan dijalankan seperti berikut :

<< gambar 12.2 eksekusi main.py >>

Daftar Pustaka

- Python Software Foundation Team. *Python v2.7.2 Documentation* 1990 – 2011. The Python Software Foundation.
- Downey, Allen, Jeffrey Elkner, dan Chris Meyers. *How to Think Like a Computer Scientist : Learning with Python*. 2002. Green Tea Press : Wellesley, Massachusetts
- Swaroop. *A Byte of Python*. 2005. IonLab : Bangalore, India
- Craven, Paul Vincent. *Introduction to Computer Science Using Python and Pygame*. 2011. Simpson College, Computer Science Department : Indianola, Iowa

Lampiran 1 – Teknologi yang Menggunakan Python

Django		Sebuah *web framework* yang memiliki motto "The Web Framework for Perfectionist with Deadline". Django merupakan salah satu megaframework yang sudah memiliki template engine, object relational mapper, session, security, pagination, logging, authentication, caching, dan lain – lain. Lebih lengkap kungjungi link berikut : http://www.djangoproject.com
PyGame		PyGame adalah *wrapper* untuk Simple Direct Media Library, sebuah *library* untuk memanipulasi grafis dan media berupa audio dan video. Dengan PyGame Anda bisa membuat *game* berbasis 2D. Kalaupun ingin membuat *game* 3D dibutuhkan *library* lain untuk mendukung pengolahan 3D Fitur – fitur yang bisa didapatkan dari module – module PyGame : • cdrom, mengelola cdrom dan pemutar suara • cursors, me-load gambar kursor, dan menyertakan kursor standard • display, mengendalikan layar • draw, menggambar grafis sederhana pada Surface • event, mengelola event dan antrian event • font, membuat dan menggunakan Truetype fonts • image, menyimpan dan me-load gambar • joystick, mengelola joystick • key, mengelola keyboard • mouse, mengelola mouse • movie, memainkan film bertipe mpeg • sndarray, memanipulasi suara dalam angka • surfarray, memanipulasi gambar dalam angka • time, mengendalikan waktu • transform, memperbesar, memutar, dan membalik gambar Bagi teman – teman yang ingin menggunakan Pygame lebih lanjut bisa kunjungi link berikut : http://www.pygame.org

Panda 3D		Panda 3D adalah 3D *Engine*, *library* dari kumpulan fungsi – fungsi untuk 3D *rendering* dan pengembangan game. *Library*-nya ditulis dengan C++. Untuk pengembangan *game* dengan Panda3D, Anda harus menulis dalam bahasa Python yang mengendalikan *library* di Panda3D. Panda3D mempunyai dukungan seperti : The Scene Graph, Model dan Actor,Texturing, Shaders, Camera Control, Sound, Interval, Task dan Event Handling, Text dan Image Rendering, DirectGUI, Render Effect, Collision Detection, dan lainnya lebih lengkap kunjungi link berikut ini : http://www.panda3d.org
SimpleCV		SimpleCV merupakan singkatan dari Simple Computer Vision, merupakan framework python yang mudah digunakan dan membungkus library computer vision open source dan algoritma terkait untuk pemecahan masalah. Beberapa fitur yang didukung oleh SimpleCV antara lain : membaca gambar, konversi gambar ke RGB, konversi gambar ke HLS, konversi gambar ke HSV, konversi gambar ke Gray, membuat gambar baru dalam format bitmap, menyalin gambar, memperbesar gambar, pencerminan gambar, memperhalus gambar, edge detection, dan lain – lain. Lebih lengkapnya checklink berikut ini : http://www.simplecv.org
NLTK		Teknologi Natural Language Processing semakin hari semakin maju. Sebagai contoh, banyak smartphones, yang sudah mendukung pengenalan tulisan, kemudian banyak mesin pencari yang mendukung penulisan teks ta struktur, ada juga penerjemahan bahasa. NLTK hadir sebagai salah satu tools yang ditulis dalam Python untuk mendukung teknologi Natural Language Processing. Beberapa fitur yang didukung oleh NLTK antara lain : Language Processing, Text Corpora, Processing Raw Text, Categorizing and Tagging Words, Parsing text, Semantic Analysis, dan lain – lain. Lebih lanjut cobe kunjungi link berikut : http://www.nltk.org
Flask		Flask merupakan micro web framework yang mendukung untuk diintegrasikan dengan berbagai

		library pendukung lainnya. Flask memerlukan WSGI Toolkit yang dinamakan Weurkzeug dan Template Engine Jinja2. Flask memiliki fitur seperti : templating engine, testing application, logging application error, debuggin application error, configuration handling, signal, pluggable view, extension, dan lain – lain. Lebih lengkap kunjungi link berikut : http://www.flask.org

Lampiran 2 - Bahan Belajar Online

Tutorial :

Java2s Python Tutorial	:	http://www.java2s.com/Tutorial/Python
Java2s Python Example	:	http://www.java2s.com/Code/Python
TutorialsPoint Python	:	http://www.tutorialspoint.com/python
Zetcode Python	:	http://www.zetcode.com/python
Learn Python The Hardway	:	http://learnpythonthehardway.org/
Python Module of The Week	:	http://pymotw.com

Forum Diskusi di Facebook:

Python	:	https://www.facebook.com/groups/gpython
Python Indonesia	:	https://www.facebook.com/groups/473114992729831/
Python Developers	:	https://www.facebook.com/groups/2249498416
Python Programmer Community	:	https://www.facebook.com/groups/pythonneres/
Python Programming	:	https://www.facebook.com/groups/pythondev/
Python User Group Malaysia	:	https://www.facebook.com/groups/python.malaysia/
Learn Python	:	https://www.facebook.com/groups/learnpython.org/

Tentang Buku Ini

Buku ini ditulis dengan menggunakan Libre Writer 3.3 dan dikonversi ke PDF dengan menggunakan *software* yang sama. *Screenshot* pada buku ini didapatkan dengan menggunakan Shutter, sebuah *software* untuk menangkap bagian layar dan merupakan FOSS (*Free Open Source Software*). Kode program ditulis dengan menggunakan Geany Text Editor. Semua kode program dicoba di Ubuntu 10.04 32 Bit dengan menggunakan Python 2.6.

Buku ini merupakan bagian dari visi POSS – UPI untuk membumikan *open source* dan mengajak masyarakat untuk menggunakan FOSS seperti Python ini dalam lingkup mengembangkan aplikasi. Untuk mendapatkan bahan belajar lainnya dari POSS – UPI, bisa didapatkan di http://www.poss-upi.org/download

Tentang GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is sinstruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or

Back−Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque". Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read an edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML produced by some word processors for output purposes only. The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must

either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modification

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of
it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher

of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Document's

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique

number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Collection of Document's

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self−contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have

received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future of Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address newproblems or concerns. See http://www.gnu.org/copyleft/ (http://www.gnu.org/copyleft/).

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to Use This License for This Document's

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Tentang Python License

History of Software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see http://www.zope.com/). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

Terms and conditions for accessing or otherwise using Python

PSF LICENSE AGREEMENT FOR PYTHON 2.7

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.7 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.7 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2010 Python Software Foundation; All Rights Reserved" are retained in Python 2.7 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.7 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.7.

4. PSF is making Python 2.7 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.7 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.7 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.7, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 2.7, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com (“BeOpen”), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization (“Licensee”) accessing and otherwise using this software in source or binary form and its associated documentation (“the Software”).

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python” logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary

form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose