Python Dasar

Tentang dokumen ini

- Dokumen ini membahas dasar-dasar bahasa pemrograman Python, untuk versi 2.x.
- Dokumen ini merupakan revisi ke 7, dimana pada awalnya, ditulis untuk Python versi 2.5, namun telah sedikit disesuaikan untuk Python 2.7.
- Python Dasar (c) 2009-2012,2014 Noprianto (noprianto.com)
- Revisi terbaru (dalam format ODT/PDF) selalu bisa didapatkan secara bebas dari https://github.com/nopri/id-python atau http://noprianto.com.
- Dokumen ini diharapkan dapat berguna namun tidak menggaransi apapun.
- Pada revisi 7, telah ditambahkan dasar-dasar untuk Python Database, Python GTK dan Python CGI (ketiga konten tersebut tidak diperbaharui sejak tahun 2009).

1. Penulisan source code

a) Indentasi adalah 4 karakter.

b) Untuk indentasi, gunakan hanya spasi, jangan gunakan TAB.

c) Panjang baris maksimal adalah 79 karakter. Pindah baris langsung dalam kurung dan hanya gunakan backslash (\) apabila diperlukan.

d) Panjang baris maksimal untuk docstring dan komentar adalah 72 karakter.

e) Gunakan pemisah dua baris kosong antara class / fungsi top-level.

f) Gunakan pemisah satu baris kosong antara method dalam class.

g) Import setiap modul dilakukan per baris, dengan urutan kelompok: standard library, pustaka pihak ketiga, pustaka lokal. Pisahkanlah masing-masing kelompok import dengan sebuah baris kosong.

h) Struktur program Python:

   1. Shebang Line
   2. Komentar dan docstring
   3. Import
   4. Global dan konstanta
   5. Class dan fungsi

i) Aturan nama:

   1. Paket: lowercase
   2. Modul: lowercase, underscore bisa digunakan kalau diperlukan.
   3. Class: kapitalisasi per kata (huruf pertama setiap kata menggunakan kapital)
   4. Exception: kapitalisasi per kata (huruf pertama setiap kata menggunakan kapital). Gunakan tambahan Error untuk menandakan kesalahan.
   5. Fungsi dan method class: lowercase, setiap kata dipisahkan underscore.
   6. Variabel: lowercase, setiap kata dipisahkan underscore.
   7. Konstanta: UPPERCASE, setiap kata dipisahkan underscore.

j) Case sensitive!

Untuk informasi lebih lanjut, bacalah juga PEP 8 (Style Guide for Python Code, http://legacy.python.org/dev/peps/pep-0008/).

2. Sekilas tentang Python

a) Open Source

   1. Sesuai dengan sertifikasi Open Source Initiative.
   2. Kompatibel dengan GPL (versi 2.0.1 dan yang lebih baru), menurut Free Software Foundation. Walau demikian, tidak ada pembatasan copyleft GPL.
   3. Bebas digunakan, termasuk untuk produk proprietary.
   4. Selengkapnya, https://www.python.org/psf/.

b) Sintaks sederhana, jelas, fleksibel, mudah dipelajari.

c) Mendukung multi-paradigma, salah satunya Object-oriented.

d) Tipe data very high level.

e) Standard library yang lengkap.

f) Berjalan di banyak sistem.

g) Dapat diextend dengan C/C++

h) Type system: Strong, Dynamic, Duck

Untuk informasi lebih lanjut, kunjungilah situs web Python
https://www.python.org/

3. Interpreter Python (interaktif)

a) Path default executable python:

   1. Windows: c:\pythonXY\python.exe. XY adalah versi Python. Contoh: c:\python25\python.exe
   2. Linux: /usr/bin/python atau /usr/bin/pythonX.Y (umumnya menggunakan symlink /usr/bin/python). XY adalah versi Python.

b) Untuk menjalankan interpreter Python:

   1. Windows: Masuk ke command prompt, masuk ke direktori instalasi Python, jalankan python.exe. Dapat pula mengakses dari Start Menu. Daftarkanlah direktori instalasi Python ke PATH apabila diperlukan.
   2. Linux: Masuk ke terminal, berikan perintah: 'python' (tanpa tanda kutip).

c) Keluar dari sesi interaktif interpreter Python:

   1. Windows: Ctrl-Z, ENTER.
   2. Linux: Ctrl-D

d) Untuk mencetak ke standard output, gunakan print.

e) Untuk membaca string dari standard input, raw_input() bisa digunakan.

f) Beberapa fungsi lain:

   1. abs()
   2. chr()
   3. dir()
   4. max()
   5. min()
   6. pow()

4. Script Python

a) Simpan dengan ekstensi nama file py. Contoh: hello.py.

b) Shebang

   1. #, !, diikuti Path ke executable Python
   2. Contoh di Linux: #!/usr/bin/python
   3. Contoh di Windows: #!c:\python25\python.exe
   4. Di Linux, bisa gunakan bantuan env, sehingga shebang line menjadi #!/usr/bin/env python

      a) Sesuaikan path ke env

c) Jalankan di command line: python <script.py>

d) Di Linux, berikan hak akses executable dengan perintah:

   1. chmod +x <script.py>
   2. Selanjutnya, bisa dijalankan langsung dengan ./script.py atau script.py (apabila direktori aktif terdaftar di $PATH) dari prompt.

5. Tipe builtin, collection dan operator

Tipe	Detil	Catatan
None	None Object	

Number	• Boolean (bool)
	  • True atau False
	• Integer (int)
	  • Batas int bisa dilihat pada sys.maxint
	• Long integer (long)
	  • Gunakan suffix L atau l
	  • Batas long sesuai memori
	• Floating point (float)
	  • pecahan
	  • bisa ditulis dalam e atau E
	• Complex (complex)
	  • bilangan kompleks
	  • memiliki atribut real (real) dan imag (imajiner).
	  • imajiner bisa dituliskan dengan suffix j atau J.
	  • dir: conjugate, imag, real	

Set	• Set (set): mutable
	  • dir: add, clear, copy, difference, difference_update, discard, intersection, intersection_update, issubset, issuperset, pop, remove, symmetric_difference, symmetric_difference_update, union, update
	• Frozen Set (frozenset): immutable
	  • dir: copy, difference, intersection, issubset, issuperset, symmetric_difference, union
		• Unordered
		• Semua item dalam set harus immutable
		• Dapat dibuat dari item iterable
		• Unik
		• Dapat dioperasikan sebagai himpunan

Sequence	• String (str)	• Memiliki

Tipe	Detil	Catatan

	• immutable
	• dapat dibentuk dengan pasangan:
	  • kutip tunggal (')
	  • kutip ganda ('')
	  • triple kutip tunggal (' ' ')
	  • triple kutip ganda ('' '' '')
	• Pasangan triple-kutip bisa mengembed newline.
	• Berlaku penggunaan escape character.
	• Dapat dituliskan dalam bentuk raw string (diawali r atau R). Contoh: r'c:\window'
	• Raw string, apabila diakhiri dengan backslash (\), harus berjumlah genap.
	• dir: capitalize, center, count, decode, encode, endswith, expandtabs, find, index, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, join, ljust, lower, lstrip, partition, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill.
	• Unicode String (unicode)
	  • immutable
	  • Diawali dengan u atau U, diikuti 4 digit hexa kode karakter unicode.
	  • Berlaku penggunaan escape character.
	  • Bisa diisi dengan \N{Name}, dimana Name sesuai aturan unicode. Contoh: \N{Copyright Sign}.
	  • Apabila menggunakan raw string, ditulis dengan ur.
	  • dir: method pada str, ditambah isdecimal dan isnumeric.
	• Tuple (tuple)
	  • immutable
		sifat iterable

Tipe	Detil	Catatan
	• Akses lebih cepat • Dapat dibuat dengan tuple(), atau dengan menempatkan anggota dalam kurung (dan). • Koma tambahan bisa dituliskan setelah anggota terakhir. Contoh: (1,2,3,) • Tuple dengan satu anggota harus dituliskan dengan menambahkan koma. Contoh: (1,) dan bukan (1) • dir: index, count (versi 2.6 ke atas) • List (list) • mutable • Lihat range() • Dapat dibuat dengan list(), atau dengan menempatkan anggota dalam kurung [dan]. • Koma tambahan bisa dituliskan setelah anggota terakhir. Contoh: [1,2,3,] • dir: append, count, extend, index, insert, pop, remove, reverse, sort. • Xrange (xrange) • immutable • Dibuat dengan xrange() • Untuk perulangan, relatif lebih cepat dari penggunaan range().	
Mapping	• Dictionary (dict) • Key harus berupa tipe immutable • Value dapat berupa hampir semua object Python • Dapat dibuat dengan dict(), atau dengan menempatkan key:value dalam kurung { dan }. • dir: clear, copy, fromkeys, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values	• Unordered (lihatlah juga pembahasan Collection)
File	• File (file) • Dibuka dengan open(name[, mode[,	

Tipe	Detil	Catatan
	buffering]]) atau file(name[, mode[, buffering]]) • Mode file: r(read, default), w(write), a(append) • Ditutup dengan method close() objek file • dir: close, closed, encoding, fileno, flush, isatty, mode, name, newlines, next, read, readinto, readline, readlines, seek, softspace, tell, truncate, write, writelines, xreadlines	

Collection

Sebagai alternatif dari container built-in seperti dict, list, set dan tuple, terdapat pula berbagai collection, yang diimplementasikan dalam module collections (versi 2.4).

- namedtuple, versi 2.6
- deque, versi 2.4
- Counter, versi 2.7
- OrderedDict, versi 2.7
- defaultdict, versi 2.5

Operator

- Perbandingan boolean
  - <, lebih kecil
  - <=, lebih kecil sama dengan
  - >, lebih besar
  - >=, lebih besar sama dengan
  - ==, sama dengan
  - !=, tidak sama dengan
- Operator Logical
  - not, logical negasi
  - and, logical and

  - or, logical or
- Operator aritmatika
  - *, perkalian
  - /, pembagian
  - //, pembagian integer
  - %, sisa bagi
  - +, penjumlahan
  - -, pengurangan
- Operator bit
  - ~, bitwise complement
  - <<, shift left
  - >>, shift right
  - &, bitwise and
  - |, bitwise or
  - ^, bitwise xor
- Operator pada sequence
  - in: terdapat di dalam. Gunakan not in untuk kebalikannya.
- Operator tambahan string:
  - %: pemformatan

Index dan Slice:

Beberapa sequence seperti string, unicode string, list dan tuple mendukung index dan slicing, dengan sintaks:

- s[i]: item ke i dari sequence s, dimulai dari 0
- s[i:j]: item ke i sampai j dari sequence s
- s[i:j:k]: item ke i sampai j dari sequence s, dengan step k

Catatan untuk i,j dan k:

- Apabila nilai i dan j negatif, maka index relatif dari akhir sequence.
- Jika i tidak diberikan, maka default ke 0
- Jika j tidak diberikan, maka default ke len(s)
- Jika k tidak diberikan, maka default ke 1. Nilai k tidak bisa diisi dengan 0.
- Jika i lebih besar atau sama dengan j, maka menghasilkan slice kosong.

Untuk tipe xrange, index didukung namun slicing tidak didukung.

Lain-lain:

- Beberapa fungsi:
  - type(object): mengembalikan tipe object
  - id(object): mengembalikan alamat memori object
  - ininstance(object, class-or-type-or-tuple): mengembalikan apakah object merupakan instance dari class.
- Kategori tipe juga mencakup Callable (dapat dipanggil), Modules (setelah diimport), Classes dan Type.

6. Kondisi

Sintaks if

```
if <expression>:
    <statement>
    <statement>
    ..
[elif <expression>:
    <statement>
    <statement>
    ..
]
[else:
    <statement>
    <statement>
    ..
]
```

Contoh if:

```
>>> a=10
>>> if a > 5:
...     print 'a besar dari 5'
...
a besar dari 5
```

Catatan:

- pass dapat digunakan untuk blok kosong.
- Python tidak mendukung switch/case

7. Perulangan

Sintaks for

```
for <target_list> in <expression_list>:
    <statement>
    <statement>
    ...
[else:
    <statement>
    <statement>
    ..
]
```

Contoh for:

```
>>> for i in range(1,10,2):
...     print i
...
1
3
5
7
9
```

Sintaks while

```
while <expression>:
    <statement>
    <statement>
    ...
[else:
    <statement>
    <statement>
    ..
]
```

Contoh while:

```
>>> a=1
>>> while a<5:
...     print a
...     a += 1
...
1
2
3
4
```

Catatan

- pass dapat digunakan untuk blok kosong.
- Statement break digunakan untuk keluar dari perulangan.
- Statement continue digunakan untuk melanjutkan ke iterasi berikutnya.
- Statement pada blok else akan dikerjakan apabila perulangan selesai, namun tidak oleh statement break.

8. Fungsi

Deklarasi fungsi

```
def <func_name>([parameter_list]):
    <statement>
    <statement>
    ...
    [return <return_value>]
```

Catatan:

- ketika return tidak didefinisikan, None akan dikembalikan

Documentation String (docstring)

Setiap fungsi dapat memiliki docstring (documentation string), yang umumnya mendeskripsikan bagaimana cara menggunakan fungsi tersebut. Docstring dapat dituliskan sebagai string langsung setelah deklarasi fungsi. Contoh:

```
def test():
...     'docstring fungsi test'
...     print 'test'
...
```

Untuk mengetahui docstring fungsi test, akseslah atribut __doc__ dari fungsi test. Contoh:

```
>>> test.__doc__
'docstring fungsi test'
```

Docstring yang didefinisikan juga berguna dalam penggunaan bersama fungsi help() di sesi interactive:

```
>>> help(test)
```

```
Help on function test in module __main__:


test()
    docstring fungsi test
```

Pemanggilan fungsi

Pemanggilan fungsi harus melihatkan penggunaan (), sesuai argumen fungsi. Tanpa (), python tidak akan menganggap sintaks tersebut sebagai sintaks yang salah, tetapi tidak akan dikerjakan.

Contoh yang salah:

```
>>> test
<function test at 0xb7c305dc>
```

Contoh yang benar:

```
>>> test()
test
```

Variabel global

Setiap fungsi akan memiliki variabel lokal sendiri. Namun, ada kalanya, akses ke variabel global diperlukan.

Untuk mengassign nilai tertentu ke variabel global, gunakan keyword global.

Contoh berikut dimaksudkan untuk mengubah variabel global x menjadi 20, namun tidak bekerja:

```
>>> x=10
>>> def testx():
...     x=20
...     print x
...
>>> x
10
```

```
>>> testx()
20
>>> x
10
>>>
```

Contoh yang benar adalah:

```
>>> x=10
>>> def testx():
...     global x
...     x=20
...     print x
...
>>> x
10
>>> testx()
20
>>> x
20
>>>
```

Walau demikian, untuk sekedar mereferensi ke variabel global, keyword global tidak diperlukan.

Argumen fungsi

Argumen fungsi bisa diberikan, tanpa harus menyebutkan tipe data. Apabila lebih dari 1 argumen, maka deretkanlah dengan dipisahkan oleh koma. Pemanggilan fungsi selanjutnya harus disesuaikan dengan argumen yang didefinisikan, kecuali default argument digunakan (lihat pembahasan berikutnya).

Contoh 1:

```
>>> def kuadrat(x):
...     return x*x
...
>>> hasil=kuadrat(12)
```

```
>>> print hasil
144
```

Contoh 2:

```
>>> def kali(a,b):
...     return a*b
...
>>> hasil=kali(2,4)
>>> print hasil
8
```

Argumen juga bisa dipanggil dengan menyebutkan nama (keyword) pada parameter formal, sehingga dapat dipanggil dengan urutan yang tidak sesuai dengan saat deklarasi. Contoh:

```
>>> def kali2(a,b):
...     print '(%d x %d) = %d' %(a, b, a*b)
...
>>> kali2(b=20, a=10)
(10 x 20) = 200
>>>
```

Ketika menggunakan keyword argument, pastikan kesalahan-kesalahan seperti berikut tidak dilakukan:

- non-keyword argument diberikan keyword argument
- duplikasi argumen
- salah memberikan keyword

Argumen default

Argumen default memungkinkan pengguna fungsi untuk memanggil fungsi dengan argumen yang lebih sedikit/sederhana dalam kondisi normal, namun memiliki keleluasaan untuk memanggil fungsi dengan semua argumen diberikan.

Argumen default digunakan apabila sebuah argumen memiliki nilai tertentu yang umum (digunakan oleh pemanggil fungsi), namun tetap dimungkinkan untuk diberikan nilai lain.

Argumen default sangat umum digunakan di Python.

Sebagai contoh, kita mendefinisikan sebuah fungsi cetak_nama, dengan dua argumen:

- name: nama yang akan dicetak
- prefix: prefix pencetakan

Dalam kondisi normal, fungsi akan selalu mencetak tulisan 'Nama Anda adalah ' (prefix) diikuti oleh name yang diberikan. Apabila prefix ingin diganti, pengguna fungsi tetap dapat melakukannya.

```
>>> def cetak_nama(name, prefix='Nama Anda adalah '):
...     print prefix + name
...
>>> cetak_nama('piton')
Nama Anda adalah piton
>>> cetak_nama('piton','Nama=')
Nama=piton
```

*Argumen *arg (tuple)*

Ketika parameter formal dituliskan dalam bentuk *arg, maka fungsi akan menerima argumen berupa tuple. Contoh:

```
>>> def testt(*t):
...     for i in t:
...         print i
...
>>> testt(1,2,3,4,5)
1
2
3
4
5
>>> testt('halo','apa','kabar')
```

```
halo
apa
kabar
```

Dengan cara seperti ini, jumlah argumen tidak didefinisikan secara kaku.

*Argumen **arg (dictionary)*

Ketika parameter formal dituliskan dalam bentuk **arg, maka fungsi akan menerima argumen berupa dictionary. Contoh:

```
>>> def testd(**arg):
...     keys = arg.keys()
...     for k in keys:
...         print '%s=>%s' %(k, arg[k])
...

>>> testd(nama='piton', umur=28, kabar='Baik')
nama=>piton
kabar=>Baik
umur=>28
>>>
```

Dengan cara seperti ini, argumen tidak didefinisikan secara kaku. Catatan mulai versi 2.6: pada saat pemanggilan fungsi dengan sintaks **, mapping lain bisa dipergunakan (tidak harus dictionary).

9. Class

Berikut adalah sintaks class:

```
class <class_name> ([baseclasses]):
    <statement>
    <statement>
    ...
```

Catatan:

- Dideklarasikan dengan keyword class
- Class mendukung docstring seperti halnya fungsi
- Setiap definisi method class akan menggunakan keyword self sebagai argumen pertama method, yang dapat digunakan untuk merujuk ke diri sendiri (object).
- Untuk merujuk ke anggota dalam class, gunakanlah kata kunci self.
- Constructor class adalah method dengan nama __init__.
- Python mendukung multiple inheritance
- Python menggunakan name mangling untuk identifier yang diawali oleh paling tidak dua underscore dan diakhiri paling banyak satu underscore. Name mangling tersebut dapat digunakan sebagai 'private variable'.
- Mengenal dua tipe class: old-style (classic) dan new-style (diperkenalkan sejak Python versi 2.2). Default adalah old-style pada python 2.x. Untuk membuat class new-style, gunakan base class berupa class new-style atau object (apabila base class tidak diperlukan).
- Instansiasi menggunakan notasi fungsi. Argumen dilewatkan pada method __init__.
- Terdapat berbagai nama method spesial, yang diantaranya dapat dilakukan untuk kustomisasi, emulasi, koersi, context manager with (versi 2.5 ke atas), dan lainnya. Ini merupakan pendekatan yang dilakukan oleh Python untuk operator overloading.

Contoh 1, class kosong:

```
>>> class MyClass1:
...     pass
...
>>> c1 = MyClass1()
```

```
>>> type(c1)
<type 'instance'>
>>> c1
<__main__.MyClass1 instance at 0xb7bf1fcc>
```

Contoh 2, class dengan docstring dan constructor:

```
>>> class MyClass2:
...     'keterangan MyClass2'
...     def __init__(self, x):
...         print 'Inisialisasi...'
...         print x
...
>>> c2 = MyClass2(10)
Inisialisasi...
10
>>> c2.__doc__
'keterangan MyClass2'
>>>
```

Contoh 3, atribut class:

```
>>> class MyClass3:
...     def __init__(self, x):
...         self.x = x
...     def method1(self):
...         print '-' * self.x
...     def method2(self):
...         self.method1()
...
>>> c3 = MyClass3(5)
>>> c3.x
5
>>> c3.method1()
-----
>>> c3.x = 10
```

```
>>> c3.method1()
----------
>>> c3.method2()
----------
>>>
```

Contoh 4, inheritance sederhana:

```
>>> class Base:
...     def method1(self):
...         print 'base....'
...
>>> class Derived1(Base):
...     def method2(self):
...         print 'derived1...'
...
>>> class Derived2(Base):
...     def method1(self):
...         print 'method1 of derived2...'
...     def method2(self):
...         print 'derived2...'
...
>>> c4 = Derived1()
>>> c4.method1()
base....
>>> c4.method2()
derived1...
>>>
>>> c5 = Derived2()
>>> c5.method1()
method1 of derived2...
>>> c5.method2()
derived2...
>>>
```

Catatan: pada class new-style, super() bisa digunakan untuk merefer ke class parent. Lihatlah juga contoh 8.

Contoh 5, name mangling sederhana:

```
>>> class Mangling:
...     def __init__(self, x):
...         self.__x = x
...     def method1(self):
...         print self.__x
...
>>> c6 = Mangling(10)
>>> c6.__x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Mangling instance has no attribute '__x'
>>> c6.method1()
10
>>> c6.__x=20
>>> c6.method1()
10
>>>
```

Contoh 6, class sebagai struct/record:

```
>>> class Struct1:
...     pass
...
>>> c7 = Struct1()
>>> c7.name = 'test'
>>> c7.address = 'test address'
>>>
>>> print c7.name
test
>>> print c7.address
test address
```

Contoh 7, akses constructor parent pada class old-style

```
>>> class Base:
...     def __init__(self):
...         print 'Inisialisasi (Base)...'
...
>>>
>>> class Derived1(Base):
...     def __init__(self):
...         print 'Inisialisasi (Derived1)...'
...
>>> class Derived2(Base):
...     def __init__(self):
...         Base.__init__(self)
...         print 'Inisialisasi (Derived2)...'
...
>>> d1 = Derived1()
Inisialisasi (Derived1)...
>>> d2 = Derived2()
Inisialisasi (Base)...
Inisialisasi (Derived2)...
```

Pada class Derived2, pemanggilan constructor parent dilakukan.

Contoh 8, akses constructor parent pada class new-style

```
>>> class Base(object):
...     def __init__(self):
...         print 'Inisialisasi (Base)...'
...
>>> class Derived1(Base):
...     def __init__(self):
...         print 'Inisialisasi (Derived1)...'
...
>>> class Derived2(Base):
...     def __init__(self):
```

```
...         super(Derived2, self).__init__()
...         print 'Inisialisasi (Derived2)...'
...
>>> d1 = Derived1()
Inisialisasi (Derived1)...
>>> d2 = Derived2()
Inisialisasi (Base)...
Inisialisasi (Derived2)...
```

Pada class Derived2, pemanggilan constructor parent dilakukan.

Contoh 9, operator overloading

```
>>> class Overload1:
...     def __init__(self, x):
...         self.x = x
...     def __add__(self, y):
...         return self.x + y
...
>>> o1 = Overload1(10)
>>> o1 + 5
15
>>>

>>> class Overload2:
...     def __init__(self, x):
...         self.x = x
...     def __sub__(self, y):
...         return self.x - y
...     def __mul__(self, y):
...         return self.x * y
...
>>> o2 = Overload2(10)
>>> o2 - 5
5
>>> o2 * 10
```

```
100
>>>

>>> class MyStr:
...     def __init__(self, x):
...         self.x = str(x)
...     def __mul__(self, y):
...         return str(self.x) * y
...     def __sub__(self, s):
...         return str(self.x).replace(s, '')
...
>>>
>>> a = 'halo apa kabar'
>>> a * 2
'halo apa kabarhalo apa kabar'
>>> a - 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
>>>
>>> s = MyStr('halo apa kabar')
>>> s * 2
'halo apa kabarhalo apa kabar'
>>> s - 'a'
'hlo p kbr'
>>>
```

Contoh terakhir mendemonstrasikan class MyStr, dimana kita ingin mirip dengan str, namun mengoverload operator -.

10. Modul-modul

Python datang dengan sangat baik modul siap pakai. Pengguna juga dapat menginstall modul tambahan dari pihak ketiga, ataupun menggunakan modul yang dibuat sendiri. Modul-modul bisa pula dikelompokkan menjadi package.

Modul dapat digunakan setelah diimport. Contoh penggunaan modul os untuk mendapatkan nama os:

```
>>> import os
>>> os.name
'posix'
```

Nama yang lebih pendek bisa digunakan, dengan import <module> as <alias>, sebagai contoh:

```
>>> import platform as p
>>> p.uname()
('Linux', 'nop1', '2.6.21.5-smp', '#2 SMP Tue Jun 19 14:58:11 CDT 2007', 'i686', 'Intel(R) Celeron(R) CPU 2.50GHz')
>>>
```

Statement import selengkapnya:

- import <module> [as <name>] (, <module> [as <name>])
- from <relative_module> import <identifier> [as <name>] (, <identifier> [as <name>])
- from <relative_module> import (<identifier> [as <name>] (, <identifier> [as <name>])... [,])
- from <module> import *

Contoh modul

Beberapa contoh modul:

- time : bekerja dengan waktu, contoh:
  - Epoch = waktu dimulai (1 Januari 1970, pukul 00:00)
  - Dapatkan detik sejak epoch:
    - time.time()

    - 1228025944.210458
- Informasi detil waktu sejak epoch (GMT):
    - time.gmtime(-100000000)
        - (1966, 10, 31, 14, 13, 20, 0, 304, 0)
    - time.gmtime(0)
        - (1970, 1, 1, 0, 0, 0, 3, 1, 0)
    - time.gmtime()
        - (2008, 11, 30, 6, 19, 48, 6, 335, 0)
- Informasi detil waktu sejak epoch (lokal):
    - time.localtime()
        - (2008, 11, 30, 15, 43, 36, 6, 335, 0)
- Parse time dari string (format bisa baca referensi modul time)
    - time.strptime('2008-11-12', '%Y-%m-%d')
        - (2008, 11, 12, 0, 0, 0, 2, 317, -1)
- Mengembalikan string time dari detik sejak epoch (lokal):
    - time.ctime(1000)
        - 'Thu Jan 1 07:16:40 1970'
- Mengembalikan string time dari sequence time (lokal):
    - time.asctime((2008,10,11,23,11,22,0,0,0))
        - 'Mon Oct 11 23:11:22 2008'
- Mengembalikan detik sejak epoch dari sequence time (lokal):
    - time.mktime((2008,10,11,23,11,22,0,0,0))
        - 1223741482.0
- Memformat waktu dari sequence time (GMT atau lokal, bacalah referensi modul time):
    - time.strftime('%d-%m-%Y', time.localtime(1000000000))
        - '09-09-2001'
- Dapatkan nama timezone (gunakan index 0, apabila non-DST):
    - time.tzname
        - ('WIT', 'WIT')
- Dapatkan selisih detik dari GMT:
    - time.timezone
        - -25200
- Menunda eksekusi selama beberapa waktu tertentu (dalam detik):
    - time.sleep(1)

    - time.sleep(0.5)
- random : bekerja dengan bilangan acak, contoh:
    - Mendapatkan bilangan acak antara 0.0 dan 1.0:
        - random.random()
            - 0.47981142386967368
    - Mendapatkan bilangan acak antara dua integer (termasuk):
        - random.randint(1000,9999)
            - 9923
    - Memilih bilangan acak dari range():
        - random.randrange(1, 100, 2)
            - 51
    - Memilih bilangan acak dari sequence:
        - random.choice([1,2,3,4])
            - 2
    - Mengacak sequence:
        - x=[1,2,3,4,5,6]
        - random.shuffle(x)
        - x
            - [1, 4, 2, 5, 3, 6]
- sys: akses ke objek yang digunakan atau berhubungan dengan interpreter
    - contoh: argv, maxint, version, platform, prefix, executable
- os : rutin sistem operasi
    - contoh: path, name
- platform : identitas platform
    - contoh: uname, system
- datetime : bekerja dengan tipe datetime
- calendar : fungsi pencetakan kalender
- math: fungsi-fungsi matematika
- base64: encoding Base16, Base32, Base64
- hashlib: (versi 2.5 ke atas) secure hash dan message digest seperti SHA1/SHA224/SHA256/SHA384/SHA512 dan MD5
- binascii: konversi binary-ascii
- csv: baca tulis file csv
- glob: pathname pattern expansion
- linecache: akses ke baris tertentu dalam file

- shutil: operasi file
- tarfile: bekerja dengan file tar, tar.gz dan tar.bz2

Selengkapnya, lihatlah index modul pada dokumentasi Python.

Search path

Ketika suatu modul diimport, Python akan mencari ke:

- direktori aktif
- daftar direktori pada variabel PYTHONPATH
- default path, lokasi instalasi python

Mulai versi 2.6, per-user site-packages didukung. Untuk informasi selengkapnya, bacalah juga environment variable PYTHONUSERBASE dan PYTHONNOUSERSITE.

Compiled module

Ketika suatu modul berhasil diimport, versi byte-compiled modul (file bernama sama dengan modul, namun dengan ekstensi .pyc/.pyo) akan coba dibuat. File byte-compiled tersebut dapat diload lebih cepat.

Mulai versi 2.6, pembuatan .pyc/.pyo bisa dicegah dengan switch -B, atau environment variabel PYTHONDONTWRITEBYTECODE.

Nama module

Setiap modul python memiliki nama masing-masing, yang dapat diakses dari properti __name__. Contoh:

```
>>> import os
>>> os.__name__
'os'
>>> import random
>>> random.__name__
'random'
```

Sebuah nama spesial __main__ akan didefinisikan apabila modul dijalankan

standalone. Dengan memeriksa apakah __name__ == '__main__', kita bisa memeriksa apakah modul diimport atau dijalankan standalone.

Contoh:

```
$ cat module1.py
#!/usr/bin/env python

if __name__ == '__main__':
    print 'Dijalankan standalone'
else:
    print 'Diimport'

$ python module1.py
Dijalankan standalone

>>> import module1
Diimport
```

Package

Modul-modul dapat dikelompokkan menjadi package. Berikut adalah beberapa catatan tentang pembuatan package:

- Sebuah package terdiri dari sebuah direktori di file system. Secara opsional, direktori tersebut dapat pula berisikan subdirektori-subdirektori.
- Masing-masing subdirektori tersebut (apabila ada) mewakili subpackage.
- Sebuah file spesial, __init__.py harus ada di dalam direktori agar direktori tersebut dianggap sebagai package. File tersebut bisa saja berupa file kosong, atau dapat berisikan inisialisasi.
- Sebuah variabel spesial __all__ (opsional; bertipe list) dapat ditentukan di dalam __init__.py, yang pada akhirnya akan menentukan apa yang tersedia ketika package diimport dengan: from package import *.

Contoh package1

```
package1
package1/__init__.py
package1/module2.py
package1/module1.py
```

```
>>> import package1.module1
>>> package1.module1.function1()
function 1
>>>
```

Contoh package2, dengan __all__

```
package2
package2/__init__.py
package2/module2.py
package2/module1.py

file __init__.py berisikan:
__all__ = ['module1']

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> from package2 import *
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'module1']
>>> module1.function1()
function 1
>>>
```

11. Exception

Kerjakan apa yang ingin dikerjakan, dan apabila terjadi kesalahan, kita siapkan handlernya.

Try...except...finally

Sintaks try...except...finally:

```
try:
    <statement>
    <statement>
    ...
except [expression [, target]]:
    <statement>
    <statement>
    ...
[else:
    <statement>
    <statement>
]
[finally:
    <statement>
    <statement>
]
```

Alur kerja:

- Statement diantara try dan except akan dikerjakan
- Apabila tidak terdapat kesalahan, maka klausa except akan dilewati
- Apabila terjadi kesalahan, sisa perintah yang masih ada di dalam try akan dilewati dan apabila exception yang bersesuaian ditemukan, maka klausa except tersebut akan dikerjakan.
- Apabila terjadi kesalahan namun tidak dihandle, maka secara otomatis, akan dilewatkan ke blok try yang lebih luar, dan apabila tidak dihandle juga, maka kesalahan tersebut merupakan unhandled exception dan eksekusi akan berhenti sesuai dengan kesalahannya.

Catatan:

- Klausa else akan dikerjakan, apabila diberikan, dan tidak ada kesalahan yang terjadi. Berguna untuk perintah yang akan dikerjakan apabila exception tidak terjadi.
- try...except...finally berlaku mulai Python v2.5. Sebelumnya, try..except harus ditempatkan bersarang di try...finally
- Untuk menangani lebih dari satu exception pada satu waktu, deretkanlah ke dalam satu tuple (versi 2.6 ke atas).

Contoh tanpa exception:

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Contoh dengan exception sederhana:

```
>>> try:
...     1/0
... except ZeroDivisionError:
...     print 'Kesalahan: pembagian dengan nol'
...
Kesalahan: pembagian dengan nol
```

Contoh dengan exception, melibatkan else:

```
>>> try:
...     1/1
... except ZeroDivisionError:
...     print 'Kesalahan: pembagian dengan nol'
... else:
...     print 'Tidak ada kesalahan yang terjadi'
...
1
Tidak ada kesalahan yang terjadi
```

Contoh dengan exception, else dan finally:

```
>>> try:
```

```
...     1/1
... except ZeroDivisionError:
...     print 'Kesalahan: pembagian dengan nol'
... else:
...     print 'Tidak ada kesalahan yang terjadi'
... finally:
...     print 'Pembagian selesai'
...
1
Tidak ada kesalahan yang terjadi
Pembagian selesai
```

Try...finally

Sintaks try...finally

```
try:
    <statement>
    <statement>
    ...
finally:
    <statement>
    <statement>
    ...
```

Catatan:

- try...finally berguna sebagai cleanup action, apapun kondisi yang terjadi.

Informasi exception

Apabila exception terjadi, informasi tertentu mungkin akan tersedia, sesuai dengan jenis exceptionnya. Untuk mendapatkan, lewatkan argumen pada exception, seperti contoh berikut.

```
>>> try:
...     1/0
... except ZeroDivisionError, e:
...     print 'Kesalahan: ',  e
...
Kesalahan: integer division or modulo by zero
```

Catatan:

– Kita dapat pula mengakses atribut message apabila dibutuhkan. Contoh:

```
>>> try:
...     1/0
... except ZeroDivisionError, e:
...     print 'Kesalahan: ' + e.message
...
Kesalahan: integer division or modulo by zero
```

Membangkitkan exception

Untuk membangkitkan kesalahan tertentu, gunakanlah raise. Argumen opsional bisa diberikan.

```
>>> raise NameError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError

>>> raise NameError, 'Ini detil kesalahan'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: Ini detil kesalahan
```

12. File

Operasi file umumnya melibatkan pembukaan file, melakukan operasi tertentu, dan kemudian menutup file terbuka tersebut.

Membuka dan menutup file

Untuk membuka file, gunakanlah fungsi:L

```
open(path [,mode [,buffersize]])
```

Catatan:

- path adalah path file yang ingin dibuka
- mode dapat bernilai:
  - r: membuka file untuk dibaca
  - w: membuka file untuk ditulis. Apabila file yang ingin dibuka telah terdapat di filesystem, maka isinya akan dihapus. Apabila file tidak ditemukan, maka akan dibuat secara otomatis.
  - a: membuka file untuk diupdate. Isi file yang sudah ada tidak dihapus.
  - r+: membuka file untuk dibaca dan ditulis. Isi file yang sudah ada tidak dihapus.
  - W+: membuka file untuk ditulis dan dibaca. Isi file yang sudah ada akan dihapus.
  - a+: membuka file untuk dibaca dan ditulis. Isi file yang sudah ada tidak dihapus.
  - b: apabila ditambahkan ke salah satu dari r, w, atau a, maka akan membuka file dalam modus binary.
  - U: apabila ditambahkan ke salah satu dari r, w atau a, maka akan mengaplikasikan universal newline. Newline pada setiap platform bisa berbeda-beda. Contoh: Linux menggunakan \n, Windows menggunakan \r\n.
- buffersize: ukuran buffer

Fungsi open yang berhasil akan mengembalikan objek file. Setelah digunakan, file tersebut harus ditutup dengan method close() (kecuali pada statement with yang dibahas pada akhir bab ini).

Contoh:

```
>>> f = open('/etc/hosts')
>>> f
```

```
<open file '/etc/hosts', mode 'r' at 0xb7bf54e8>
>>> f.close()
>>> f
<closed file '/etc/hosts', mode 'r' at 0xb7bf54e8>
>>>
```

Membaca isi file

Untuk membaca isi file yang telah dibuka, beberapa cara bisa digunakan:

- Membaca sekaligus isi file dengan method readlines(). Hasil pembacaan akan disimpan di list.

```
>>> f = open('/tmp/abc')
>>> isi = f.readlines()
>>> f.close()
>>> isi
['test baris 1\n', 'test baris 2\n', 'test baris 3\n']
>>>
```

- Membaca sekaligus isi file dengan method read(). Hasil pembacaan akan disimpan di string.

```
>>> f = open('/tmp/abc')
>>> isi = f.read()
>>> f.close()
>>> isi
'test baris 1\ntest baris 2\ntest baris 3\n'
```

- Membaca baris demi baris dengan method readline().

```
>>> f = open('/tmp/abc')
>>> while True:
...     baris = f.readline()
...     if not baris:
...         break
...     print baris
```

```
...
test baris 1

test baris 2

test baris 3

>>> f.close()
>>>
```

– Membaca langsung baris tertentu dengan module linecache.

```
>>> import linecache
>>> baris2 = linecache.getline('/tmp/abc',2)
>>> print baris2
test baris 2

>>> linecache.clearcache()
```

– Membaca sejumlah byte tertentu dengan method read().

```
>>> f = open('/tmp/abc')
>>> buf3 = f.read(3)
>>> print buf3
tes
>>> f.close()
>>>
```

Menulis ke file

Untuk menulis ke file yang telah dibuka, beberapa cara bisa digunakan:

– Menulis string dengan method write.

```
>>> f = open('/tmp/test','w')
>>> f.write('halo apa kabar')
>>> f.close()
```

```
>>>
>>> print open('/tmp/test').readlines()
['halo apa kabar']
```

- Menulis sequence dengan method writelines.

```
>>> lines = ['halo', 'apa', 'kabar']
>>> f = open('/tmp/test2', 'w')
>>> f.writelines(lines)
>>> f.close()
>>>
>>> print open('/tmp/test2').readlines()
['haloapakabar']
>>>
```

Statement with

Pada python versi 2.5, statement with diperkenalkan sebagai fitur opsional, dimana untuk menggunakannya, kita melakukan:

```
from __future__ import with_statement
```

Pada python versi 2.6, statement with bisa langsung dipergunakan.

```
with expression [as variable]:
    with-block
```

Statement ini dapat digunakan untuk memastikan kode untuk melakukan clean-up akan dikerjakan, seperti halnya pada try/finally. Dan, ini berlaku untuk object yang mendukung context management protocol.

Contoh yang mendukung adalah file. Dengan demikian, kita bisa bekerja dengan file dengan cara berikut:

```
>>> with open('/tmp/test.txt') as f:
...     for line in f:
```

```
...         print line
...
```

Objek file f dalam contoh tersebut akan ditutup secara otomatis.

13. Database dan DB-API 2.0

- Berbagai relational database datang dengan fungsi umum, disamping fungsi spesifik database system tersebut, namun bisa dengan API yang berbeda-beda antar database (koneksi, query, transaksi, dan lain-lain). API yang berbeda menyebabkan perpindahan ke database lain menjadi sangat merepotkan, karena hampir semua fungsi yang digunakan harus diganti.
- Python datang dengan DB-API (versi 2.0), yang menyediakan API umum untuk beragam modul Python, untuk berbagai database system. Dengan demikian, perpindahan ke database lain menjadi sangat mudah, karena hampir semua fungsi yang digunakan adalah sama. Hanya modulnya saja yang berbeda. Pada penerapan di dunia nyata, terkadang hanya beberapa baris saja yang perlu diubah.
- Developer sebisa mungkin diharapkan tidak mengakses fitur non-standard database system tertentu.
- DB-API 2.0 dapat dibaca pada PEP 249 (http://www.python.org/dev/peps/pep-0249/)
- Tidak semua modul/adapter database kompatibel dengan DB-API 2.0.

Koneksi database

Sebelum bekerja dengan PostgreSQL atau MySQL, koneksi ke database server harus dilakukan terlebih dahulu. Walaupun kedua modul (psycopg2 dan mysql-python) kompatibel dengan DB-API 2.0, ada sedikit perbedaan dalam melakukan koneksi.

PostgreSQL

Untuk melakukan koneksi, method connect() psycopg2 digunakan. Parameter untuk connect() dapat berupa string data source name (dsn) ataupun dengan keyword. Training hanya akan membahas penggunaan keyword argument, yang terdiri dari:

- database: nama database
- host: alamat host database (default: UNIX socket)
- port: port database (default: 5432)
- user: nama user yang melakukan koneksi
- password: password user
- sslmode: mode SSL

Setelah koneksi berhasil dilakukan, sebuah objek connection akan dikembalikan.

Setelah selesai bekerja dengan database, koneksi dapat ditutup dengan method close() objek connection. Penting diingat: PostgreSQL adalah database yang mendukung transaksi, apabila koneksi ditutup tanpa melakukan commit (akan dibahas pada query dan transaksi), maka rollback akan dipanggil secara implisit, sehingga perubahan tidak tersimpan.

Contoh koneksi PostgreSQL dapat pula dilihat pada source pgsql_connect.py.

pgsql_connect.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')
print conn
conn.close()
```

MySQL

Untuk melakukan koneksi, method connect() MySQLdb digunakan. Parameter untuk connect() adalah keyword, yang diantaranya terdiri dari:

- db: nama database
- host: alamat host database
- port: port database
- user: nama user yang melakukan koneksi
- passwd: password user
- dan lain sebagainya, yang dapat dibaca pada file connections.py paket MySQLdb.

Setelah koneksi berhasil dilakukan, sebuah objek connection akan dikembalikan.

Setelah selesai bekerja dengan database, koneksi dapat ditutup dengan method close() objek connection.

Penting diingat:

- Apabila menggunakan engine yang mendukung transaksi, maka apabila koneksi ditutup tanpa melakukan commit (akan dibahas pada query dan transaksi), maka rollback akan dipanggil secara implisit, sehingga perubahan tidak tersimpan.
- Apabila menggunakan engine tanpa transaksi, maka sebaiknya commit tetap dilakukan (walau tanpa efek) setiap ada perubahan pada database. Hal ini akan mempermudah apabila database (atau storage engine) yang digunakan, diganti ke yang lainnya.

Contoh koneksi MySQL dapat pula dilihat pada source mysql_connect.py.

mysql_connect.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')
print conn
conn.close()
```

Query

Untuk melakukan query, kita bekerja dengan cursor, yang akan mengatur eksekusi query dan mendapatkan hasilnya.

- Objek Cursor dibuat dengan memanggil method cursor() objek connection.
- Setelah digunakan, cursor bisa ditutup dengan method close() objek Cursor.
- Apabila database mendukung transaksi, maka transaksi akan dimulai otomatis begitu cursor dibuat.

Tidak semua database system mendukung transaksi. Untuk database yang mendukung transaksi, semua perubahan pada database tidak tersimpan sampai commit dilakukan. Ketika bekerja dengan database yang tidak mendukung transaksi, commit sebaiknya juga tetap dilakukan (walau tanpa efek), sehingga dapat mempermudah apabila database yang digunakan akan diganti ke yang lainnya.

Mengirimkan query

Untuk mengirimkan query, gunakan method execute() cursor. Developer bisa mengirimkan query berupa string (keyword query, atau parameter pertama). Hanya, satu hal yang perlu diperhatikan adalah query string yang dikirimkan, apabila melibatkan input dari user, maka developer haruslah berhati-hati. Tidak semua input dari user bisa dipercaya, karena bisa saja merupakan serangan SQL injection.

Agar lebih aman, gunakan parameter substitution. Setiap input bisa diwakili oleh sebuah parameter dan modul database akan otomatis melakukan pemeriksaan dan tindakan lain yang diperlukan agar query dapat dikirimkan dengan aman.

Format parameter yang digunakan bisa berbeda-beda. Developer bisa mengamati format yang digunakan, dengan melihat pada <module>.paramstyle. Berikut ini adalah format yang mungkin:

- Qmark: tanda tanya, contoh: where name=?
- Numeric: parameter berupa nomor (posisi), contoh: where name=:1
- Named: parameter berupa nama, contoh: where name=:name
- Format: format seperti printf C, contoh: where name=%s
- Pyformat: format extended Python, contoh: where name=%(name)s

Untuk materi training, yang akan digunakan adalah format. Sebuah tuple/list berisikan parameter-parameter diberikan sebagai argumen kedua (keyword vars untuk psycopg2 atau keyword args untuk MySQLdb) method execute() cursor.

Contoh-contoh query PostgreSQL dapat dilihat pada pgsql_createtbl.py dan pgsql_query_insert.py. Contoh-contoh query MySQL dapat dilihat pada mysql_createtbl.py dan mysql_query_insert.py.

pgsql_createtbl.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()
query = '''
```

```
create table books (id serial not null, title varchar(128), author
varchar(128),
 primary key(id));
'''
cur.execute(query)
conn.commit()

cur.close()
conn.close()
```

pgsql_query_insert.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()
cur.execute('''
insert into books(author, title) values(%s, %s)
''', ('author1', 'title1'))
conn.commit()

cur.close()
conn.close()
```

mysql_createtbl.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')
```

```
cur = conn.cursor()
query = '''
create table books (id integer auto_increment not null,title varchar(128),
autho
r varchar(128),primary key(id));
'''
cur.execute(query)


cur.close()
conn.close()
```

mysql_query_insert.py:

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


import MySQLdb as mysql


conn = mysql.connect(user='user', db='testing')


cur = conn.cursor()
cur.execute('''
insert into books(author, title) values(%s, %s)
''', ('author1', 'title1'))
conn.commit()


cur.close()
conn.close()
```

Catatan:

Dalam contoh-contoh insert, nilai parameter diberikan oleh developer, yang tidak berisikan nilai yang invalid. Dapat pula diberikan langsung tanpa harus disubstitusi. Contoh telah dibuat lebih sederhana tanpa input user.

Mendapatkan hasil query

Untuk mendapatkan hasil query manipulasi berupa insert/update/delete (affected rows), setelah query dikirimkan, akseslah properti rowcount

cursor.

Untuk mendapatkan hasil query select, gunakan salah satu method cursor berikut:

- fetchone(): mendapatkan row berikut dalam result set. Akan mengembalikan sequence tunggal, atau None apabila tidak ada data.
- fetchmany([size=cursor.arraysize]): mendapatkan sejumlah row berikut dalam result set. Akan mengembalikan sequence dari sequence, atau sequence kosong apabila tidak ada data.
- fetchall(): mendapatkan semua row tersisa dalam result set. Akan mengembalikan sequence dari sequence, atau sequence kosong apabila tidak ada data.

Contoh-contoh query PostgreSQL dapat dilihat pada pgsql_query_rowcount.py dan pgsql_query_select.py. Contoh-contoh query MySQL dapat dilihat pada mysql_query_rowcount.py dan mysql_query_select.py.

pgsql_query_rowcount.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()
cur.execute('''
insert into books(author, title) values(%s, %s)
''', ('author1', 'title1'))
conn.commit()

print 'Rowcount: %d' %(cur.rowcount)

cur.close()
conn.close()
```

pgsql_query_select.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()

print 'Fetch all'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

print 'Fetch one'
cur.execute('''select * from books''')
books = []
while True:
    temp = cur.fetchone()
    if temp:
        books.append(temp)
    else:
        break
print books

print 'Fetch first 3'
cur.execute('''select * from books''')
books = cur.fetchmany(3)
print books

cur.close()
conn.close()
```

mysql_query_rowcount.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')

cur = conn.cursor()
cur.execute('''
insert into books(author, title) values(%s, %s)
''', ('author1', 'title1'))
conn.commit()

print 'Rowcount: %d' %(cur.rowcount)

cur.close()
conn.close()
```

mysql_query_select.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')

cur = conn.cursor()

print 'Fetch all'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

print 'Fetch one'
cur.execute('''select * from books''')
```

```
books = []
while True:
    temp = cur.fetchone()
    if temp:
        books.append(temp)
    else:
        break
print books

print 'Fetch first 3'
cur.execute('''select * from books''')
books = cur.fetchmany(3)
print books

cur.close()
conn.close()
```

Representasi tipe data

Salah satu fitur DB-API 2.0 yang sangat menarik adalah representasi tipe data pada database ke tipe data Python, secara otomatis.

Sebagai contoh, apabila query select mengembalikan kolom date/datetime/sejenis, maka sequence hasil kembalian dapat mengandung data dengan tipe datetime. Begitupun halnya dengan bilangan, string dan lainnya (seperti contoh-contoh sebelumnya).

Lihatlah contoh-contoh pgsql_query_date.py dan mysql_query_date.py.

pgsql_query_date.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()
```

```
cur.execute('''select now()''')
dateinfo = cur.fetchall()
print dateinfo

cur.close()
conn.close()
```

mysql_query_date.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')

cur = conn.cursor()
cur.execute('''select now()''')
dateinfo = cur.fetchall()
print dateinfo

cur.close()
conn.close()
```

Catatan:

- Tipe NULL SQL diwakili dengan None Python.
- Selengkapnya, bacalah PEP 249.

Transaction

Untuk database yang mendukung transaksi, method commit() dan rollback() objek connection dapat digunakan, masing-masing untuk melakukan transaction commit dan rollback.

Lihatlah contoh-contoh pgsql_trans.py dan mysql_trans.py. Terdapat perbedaan yang mencolok, karena PostgreSQL mendukung transaksi dan engine yang digunakan pada MySQL adalah yang tidak mendukung transaksi.

Program contoh akan:

- Mendapatkan semua record dalam tabel books
- Menghapus isi tabel books
- Rollback. Apabila mendukung transaksi, maka penghapusan isi tabel tidak akan disimpan.
- Mendapatkan semua record dalam tabel books.

pgsql_trans.py:

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

import psycopg2 as pgsql

conn = pgsql.connect(user='user', database='testing')

cur = conn.cursor()

print 'Select *'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

print 'Delete from'
cur.execute('''delete from books''')
print 'Rowcount: %d' %(cur.rowcount)
conn.rollback()

print 'Rollback, Select *'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

cur.close()
conn.close()
```

mysql_trans.py:

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

import MySQLdb as mysql

conn = mysql.connect(user='user', db='testing')

cur = conn.cursor()

print 'Select *'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

print 'Delete from'
cur.execute('''delete from books''')
print 'Rowcount: %d' %(cur.rowcount)
conn.rollback()

print 'Rollback, Select *'
cur.execute('''select * from books''')
books = cur.fetchall()
print books

cur.close()
conn.close()
```

14. GTK+ dan PyGTK

a) GTK+ (GIMP Tool Kit): GUI toolkit multiplatform yang lengkap, terdokumentasi baik, tersedia binding ke berbagai bahasa pemrograman (default C, dengan ide berorientasi objek), telah digunakan di berbagai proyek besar (contoh: GNOME) dan dilisensikan di bawah LGPL (free software, dapat digunakan pada aplikasi proprietary).

b) GTK+ dibangun di atas GDK (GIMP Drawing Kit), yang merupakan wrapper untuk fungsi drawing low level platform.

c) PyGTK: modul Python yang menyediakan interface ke GTK+. PyGTK merupakan binding official GNOME.

d) Informasi selengkapnya:

   1. GTK+: http://gtk.org

   2. PyGTK: http://pygtk.org

b) Untuk bekerja dengan PyGTK:

   1. import pygtk

   2. pygtk.require('2.0')

      a) Kita ingin menggunakan PyGTK versi 2.x dan mencegah digunakannya versi lain apabila terinstall.

   3. import gtk

c) Dalam bekerja dengan beragam widget GTK+, developer bisa mempergunakan stock GTK+ yang menyediakan ID icon siap pakai (dan mengandung informasi stock id, string label, modifier, keyval dan translation domain). Berikut adalah tabel stock PyGTK 2.10 (GTK+ versi 2.10).

gtk.STOCK_ABOUT	gtk.STOCK_ADD	gtk.STOCK_APPLY	gtk.STOCK_BOLD
gtk.STOCK_CANCEL	gtk.STOCK_CDROM	gtk.STOCK_CLEAR	gtk.STOCK_CLOSE
gtk.STOCK_COLOR_PICKER	gtk.STOCK_CONVERT	gtk.STOCK_CONNECT	gtk.STOCK_COPY
gtk.STOCK_CUT	gtk.STOCK_DELETE	gtk.STOCK_DIALOG_AUTHENTICATION	gtk.STOCK_DIALOG_ERROR
gtk.STOCK_DIALOG_INFO	gtk.STOCK_DIALOG_QUESTION	gtk.STOCK_DIALOG_WARNING	gtk.STOCK_DIRECTORY
gtk.STOCK_DISCONNECT	gtk.STOCK_DND	gtk.STOCK_DND_MULTIPLE	gtk.STOCK_EDIT
gtk.STOCK_EXECUTE	gtk.STOCK_FILE	gtk.STOCK_FIND	gtk.STOCK_FIND_AND_REPLACE

gtk.STOCK_FLOPPY	gtk.STOCK_FULLSCREEN	gtk.STOCK_GOTO_BOTTOM	gtk.STOCK_GOTO_FIRST
gtk.STOCK_GOTO_LAST	gtk.STOCK_GOTO_TOP	gtk.STOCK_GO_BACK	gtk.STOCK_GO_DOWN
gtk.STOCK_GO_FORWARD	gtk.STOCK_GO_UP	gtk.STOCK_HARDDISK	gtk.STOCK_HELP
gtk.STOCK_HOME	gtk.STOCK_INDENT	gtk.STOCK_INDEX	gtk.STOCK_INFO
gtk.STOCK_ITALIC	gtk.STOCK_JUMP_TO	gtk.STOCK_JUSTIFY_CENTER	gtk.STOCK_JUSTIFY_FILL
gtk.STOCK_JUSTIFY_LEFT	gtk.STOCK_JUSTIFY_RIGHT	gtk.STOCK_LEAVE_FULLSCREEN	gtk.STOCK_MEDIA_FORWARD
gtk.STOCK_MEDIA_NEXT	gtk.STOCK_MEDIA_PAUSE	gtk.STOCK_MEDIA_PLAY	gtk.STOCK_MEDIA_PREVIOUS
gtk.STOCK_MEDIA_RECORD	gtk.STOCK_MEDIA_REWIND	gtk.STOCK_MEDIA_STOP	gtk.STOCK_MISSING_IMAGE
gtk.STOCK_NETWORK	gtk.STOCK_NEW	gtk.STOCK_NO	gtk.STOCK_OK
gtk.STOCK_OPEN	gtk.STOCK_ORIENTATION_LANDSCAPE	gtk.STOCK_ORIENTATION_PORTRAIT	gtk.STOCK_ORIENTATION_REVERSE_LANDSCAPE
gtk.STOCK_ORIENTATION_REVERSE_PORTRAIT	gtk.STOCK_PASTE	gtk.STOCK_PREFERENCES	gtk.STOCK_PRINT
gtk.STOCK_PRINT_PREVIEW	gtk.STOCK_PROPERTIES	gtk.STOCK_QUIT	gtk.STOCK_REDO
gtk.STOCK_REFRESH	gtk.STOCK_REMOVE	gtk.STOCK_REVERT_TO_SAVED	gtk.STOCK_SAVE
gtk.STOCK_SAVE_AS	gtk.STOCK_SELECT_ALL	gtk.STOCK_SELECT_COLOR	gtk.STOCK_SELECT_FONT
gtk.STOCK_SORT_ASCENDING	gtk.STOCK_SORT_DESCENDING	gtk.STOCK_SPELL_CHECK	gtk.STOCK_STOP
gtk.STOCK_STRIKETHROUGH	gtk.STOCK_UNDELETE	gtk.STOCK_UNDERLINE	gtk.STOCK_UNDO
gtk.STOCK_UNINDENT	gtk.STOCK_YES	gtk.STOCK_ZOOM_100	gtk.STOCK_ZOOM_FIT
gtk.STOCK_ZOOM_FIT	gtk.STOCK_ZOOM_OUT		

Hello World

Dalam program contoh hello.py, sebuah window kosong dengan title 'Hello World' akan dibuat dan ditampilkan.

hello.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Hello World')

        self.window.show_all()

app = Main()
gtk.main()
```

Jalankanlah dengan perintah:

```
$ python hello.py
```

Untuk keluar dari program, tombol close pada window tidak dapat digunakan. Kembali ke shell dan tekanlah kombinasi tombol CTRL-C. Pesan berikut bisa diabaikan:

```
Traceback (most recent call last):
  File "hello.py", line 15, in <module>
    gtk.main()
KeyboardInterrupt
```

Penjelasan:

- Untuk membuat window, gunakan: gtk.Window(type=gtk.WINDOW_TOPLEVEL)
  - Argumen type bisa berupa:
    - gtk.WINDOW_TOPLEVEL: window toplevel
    - gtk.WINDOW_POPUP: dapat digunakan untuk window pop-up seperti pop-up menu atau tooltip.
  - Mengembalikan objek gtk.Window
- Untuk mengatur title Window, gunakan: gtk.Window.set_title(title)
- Agar window dapat ditampilkan, gunakan: gtk.Widget.show_all().
  - Perhatikanlah bahwa show_all() adalah method gtk.Widget. Class gtk.Window diturunkan dari: gobject.GObject -> gtk.Object -> gtk.Widget -> gtk.Container -> gtk.Bin.
  - Method show_all() akan menampilkan widget secara rekursif, termasuk semua child widget.
  - Untuk widget tunggal, method show() bisa dipergunakan.
  - Sebelum di-show, widget tidak akan ditampilkan.
- Selengkapnya, bacalah class reference untuk class-class terkait.

Layout Container

Container merupakan widget yang dapat mengandung widget lain (child widget). Beberapa widget container, selain mengandung widget lain, dapat pula melakukan layout (layout container). Dengan menggunakan layout container, peletakan berbagai widget dalam satu container dapat dilakukan dengan mudah.

Sebagai catatan, tidak semua layout dibahas. Pemilihan layout juga bergantung pada preferensi developer dan user interface yang dirancang. Dalam materi training ini selanjutnya, karena jumlah dan posisi widget yang sederhana, gtk.Box yang akan dipergunakan. Namun, dalam penggunaan sehari-hari, dengan user interface yang rumit, gtk.Table atau kombinasi antara gtk.Box dan gtk.Table dipergunakan.

Untuk keluar dari semua contoh program di dalam bab ini, tombol close pada window tidak dapat digunakan. Kembali ke shell dan tekanlah kombinasi tombol CTRL-C.

gtk.Box

Class gtk.Box merupakan base class abstrak untuk container box:

- gtk.HBox: melayout child widget secara horizontal
- gtk.VBox: melayout child widget secara vertikal

Contoh penggunaan dapat dilihat pada hello_hbox.py dan hello_vbox.py.

Constructor:

```
gtk.HBox(homogeneous=False, spacing=0)

gtk.VBox(homogeneous=False, spacing=0)
```

- homogeneous: menentukan apakah child widget diberikan alokasi ruang yang sama.
- spacing: ruang kosong (horizontal untuk Hbox dan vertikal untuk VBox) antara child widget, dalam pixel.

Bekerja dengan child widget:

- Menambahkan child widget dari awal, gunakan: gtk.Box.pack_start(child, expand=True, fill=True, padding=0)
  - child: child widget yang akan ditambahkan.
  - expand: menentukan apakah child widget diberikan ruang kosong ekstra. Ruang kosong ekstra ini akan dibagi rata antara semua child widget yang ditambahkan dengan opsi ini.
  - fill: menentukan apakah ruang kosong ekstra yang diberikan (lewat expand) benar-benar dialokasikan untuk child widget, dan bukan sekedar digunakan sebagai padding. Opsi ini tidak memiliki efek apabila expand tidak diset True.
  - padding: ruang kosong tambahan (dalam pixel) antara child widget dan widget sekitarnya, di atas spacing gtk.Box.
- Menambahkan child widget dari akhir, gunakan: gtk.Box.pack_end(child, expand=True, fill=True, padding=0).
- Penambahan child bisa dilakukan bersarang. Dengan demikian, gtk.HBox bisa ditambahkan ke gtk.HBox lain atau gtk.VBox, dan seterusnya.

hello_hbox.py:

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

```
import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Hello World')

        self.hbox = gtk.HBox()

        self.entry = gtk.Entry()
        self.button = gtk.Button('_uppercase')

        self.hbox.pack_start(self.entry)
        self.hbox.pack_start(self.button)

        self.window.add(self.hbox)

        self.window.show_all()

app = Main()
gtk.main()
```

hello_vbox.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
```

```
        self.window.set_title('Hello World')

        self.vbox = gtk.VBox()

        self.entry = gtk.Entry()
        self.button = gtk.Button('_uppercase')

        self.vbox.pack_start(self.entry)
        self.vbox.pack_start(self.button)

        self.window.add(self.vbox)

        self.window.show_all()

app = Main()
gtk.main()
```

Catatan contoh:

- Contoh hello_hbox.py dan hello_vbox.py mempergunakan widget gtk.Button dan gtk.Entry. Kedua widget tersebut bisa diabaikan terlebih dahulu, dan fokus diberikan pada gtk.HBox atau gtk.VBox.
- Lebih lanjut tentang gtk.Button dan gtk.Entry dapat dilihat pada pembahasan Beragam Widget (Bab 6).
- Widget-widget ditambahkan pada gtk.HBox atau gtk.VBox. Setelah itu, gtk.HBox atau gtk.VBox yang ditambahkan ke gtk.Window.

gtk.Table

Widget-widget dapat diatur dalam baris dan kolom dengan layout container gtk.Table. Selain peletakan pada baris dan kolom, ukuran widget juga dapat diset dalam ukuran baris dan kolom. Jumlah baris dan kolom gtk.Table bisa ditentukan dan diresize.

Contoh penggunaan dapat dilihat pada hello_table.py.

Constructor:

gtk.Table(rows=1, columns=1, homogeneous=False)

- rows: jumlah baris
- columns: jumlah kolom
- homogeneous: menentukan apakah semua cell dalam tabel akan memiliki ukuran seperti ukuran cell terbesar.

Bekerja dengan child widget:

- Untuk menambahkan child widget, gunakan: gtk.Table. attach(child, left_attach, right_attach, top_attach, bottom_attach, xoptions=gtk.EXPAND|gtk.FILL, yoptions=gtk.EXPAND|gtk.FILL, xpadding=0, ypadding=0)
  - child: widget yang akan ditambahkan.
  - left_attach: kolom untuk menempatkan sisi kiri child widget. Kolom dimulai dari 0.
  - right_attach: kolom untuk menempatkan sisi kanan child widget. Kolom dimulai dari 0.
  - top_attach: baris untuk menempatkan sisi atas child widget. Baris dimulai dari 0.
  - bottom_attach: baris untuk menempatkan sisi bawah child widget. Baris dimulai dari 0.
  - xoptions: menentukan property child widget ketika tabel diresize secara horizontal. Dapat merupakan kombinasi gtk.EXPAND (ekspansi semua ruang kosong ekstra yang dialokasikan), gtk.SHRINK (disusutkan apabila cell tabel disusutkan) dan gtk.FILL (mengisi ruang kosong yang dialokasikan kepada widget dalam cell tabel).
  - yoptions: menentukan property child widget ketika tabel diresize secara vertikal. Lihat nilai untuk xoptions.
  - xpadding: padding sisi kiri dan kanan widget.
  - ypadding: padding sisi atas dan bawah widget.
- Untuk mengubah ukuran tabel, gunakan: gtk.Table.resize(rows, columns). Argumen rows dan columns masing-masing menentukan baris dan kolom yang baru.
- Untuk mengatur spasi baris: gtk.Table.set_row_spacing(row, spacing)
- Untuk mengatur spasi kolom: gtk.Table.set_col_spacing(column, spacing)

hello_table.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Hello World')

        self.table = gtk.Table(1,2)

        self.entry = gtk.Entry()
        self.button = gtk.Button('_uppercase')

        self.table.attach(self.entry, 0, 1, 0, 1)
        self.table.attach(self.button, 1, 2, 0, 1)

        self.window.add(self.table)

        self.window.show_all()

app = Main()
gtk.main()
```

Catatan contoh:

- Contoh hello_table.py mempergunakan widget gtk.Button dan gtk.Entry. Kedua widget tersebut bisa diabaikan terlebih dahulu, dan fokus diberikan pada gtk.Table.
- Lebih lanjut tentang gtk.Button dan gtk.Entry dapat dilihat pada pembahasan Beragam Widget (Bab 6).
- Widget-widget ditambahkan pada gtk.Table. Setelah itu, gtk.Table yang ditambahkan ke gtk.Window.

gtk.Fixed

Untuk mengatur posisi child widget pada koordinat tertentu, gunakanlah layout container gtk.Fixed. Walaupun relatif lebih mudah digunakan, penggunaan gtk.Fixed cukup merepotkan apabila container/window diresize, dan harus tetap mempertahankan rasio ukuran dan posisi child widget.

Contoh penggunaan dapat dilihat pada hello_fixed.py.

Constructor:

gtk.Fixed()

Bekerja dengan child widget:

- Untuk menambahkan child widget, gunakan: gtk.Fixed.put(widget, x, y):
  - widget: widget yang akan ditambahkan.
  - x: posisi x
  - y: posisi y
- Untuk memindahkan child widget ke lokasi lain, gunakanlah: gtk.Fixed.move(widget, x, y):
  - widget: widget yang akan ditambahkan.
  - x: posisi x baru
  - y: posisi y baru
- Nilai x=0 dan y=0 adalah sudut kiri atas.

hello_fixed.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Hello World')
```

```
        self.window.set_size_request(300,100)

        self.fixed = gtk.Fixed()

        self.entry = gtk.Entry()
        self.button = gtk.Button('_uppercase')

        self.fixed.put(self.entry, 10, 10)
        self.fixed.put(self.button, 200, 10)

        self.window.add(self.fixed)

        self.window.show_all()

app = Main()
gtk.main()
```

Catatan contoh:

- Contoh hello_fixed.py mempergunakan widget gtk.Button dan gtk.Entry. Kedua widget tersebut bisa diabaikan terlebih dahulu, dan fokus diberikan pada gtk.Fixed.
- Lebih lanjut tentang gtk.Button dan gtk.Entry dapat dilihat pada pembahasan Beragam Widget (Bab 6).
- Widget-widget ditambahkan pada gtk.Fixed sesuai posisi yang ditentukan. Setelah itu, gtk.Fixed yang ditambahkan ke gtk.Window.
- Window utama diresize untuk memudahkan demonstrasi gtk.Fixed.

Signal dan callback

GTK+ adalah event driven toolkit. Dengan demikian, developer bisa mengatur agar event tertentu yang terjadi bisa dihandle. Contoh event sederhana adalah klik tombol mouse pada suatu tombol. Semua event yang terjadi dimonitor di dalam gtk.main().

Terminologi event di dalam GTK+ setidaknya bisa diterima sebagai:

- signal

- event (dari windowing system).

Widget bisa menerima signal, baik signal yang umum (diturunkan dari base class), ataupun signal spesifik widget tersebut. Daftar signal yang bisa diterima oleh widget beserta deskripsinya bisa didapatkan dari class reference.

Baik signal ataupun event bisa ditangani oleh fungsi tertentu, yang umumnya disebut sebagai callback di dalam GTK+.

Contoh-contoh program sebelumnya tidak bisa ditutup dengan klik pada tombol close window. Hal ini disebabkan karena program tidak menangani event yang terjadi.

Sebagai catatan, event pada GTK+ sangatlah luas dan kompleks. Selalulah merujuk ke referensi class untuk informasi selengkapnya.

Koneksi signal-callback

Koneksi

Untuk mengatur agar signal tertentu dihubungkan ke callback tertentu, gunakanlah method connect() widget:

```
gobject.GObject.connect(detailed_signal, handler, ...).
```

- detailed_signal: string berisikan nama signal
- handler: fungsi atau method callback
- ...: argumen opsional tambahan
- Nilai kembalian adalah handler ID (integer)

Catatan:

- Berbagai signal bisa diset agar ditangani.
- Banyak callback bisa dihubungkan ke satu signal tertentu, dan akan dikerjakan sesuai urutan koneksi.

Definisi callback

Callback secara umum dapat didefinisikan sebagai berikut:

```
def callback (widget, data)
```

Apabila argumen opsional tidak digunakan pada connect(), maka callback secara umum dapat didefinisikan sebagai berikut:

```
def callback (widget, data=None)
```

atau

```
def callback (widget)
```

Catatan:

- Callback harus menyesuaikan dengan jumlah argumen opsional yang diberikan.
- Apabila callback digunakan dalam class, maka argumen pertama pada definisi callback tetap adalah self.

Block dan unblock

Koneksi signal dan callback bisa diblock atau diunblock sementara waktu dengan:

```
gobject.GObject.handler_block(handler_id)
```

dan

```
gobject.GObject.handler_unblock(handler_id)
```

handler_id adalah nilai integer yang dikembalikan oleh connect().

Diskoneksi

Apabila signal tidak lagi ingin ditangani, maka method disconnect() widget bisa digunakan:

```
gobject.GObject.disconnect(handler_id)
```

handler_id adalah nilai integer yang dikembalikan oleh connect().

Apabila diperlukan, periksalah apakah suatu handle terkoneksi dengan:

gobject.GObject.handler_is_connected(handler_id)

yang akan mengembalikan True apabila callback dengan ID handler_id terkoneksi.

Mengaktifkan (emit) signal

Signal bisa diaktifkan dengan:

gobject.GObject.emit(detailed_signal, ...).

- detailed_signal: string berisikan nama signal
- ...: argumen tambahan

Jumlah dan tipe argumen tambahan harus sesuai dengan jumlah dan tipe yang disyaratkan callback.

signal_simple.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Signal')
        self.window.connect('destroy', self.destroy)

        self.window.show_all()

    def destroy(self, widget, data=None):
```

```
        gtk.main_quit()


app = Main()
gtk.main()
```

signal_hello.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Hello World')
        self.window.connect('destroy', self.destroy)

        self.hbox = gtk.HBox()

        self.entry = gtk.Entry()
        self.button = gtk.Button('_uppercase')
        self.button.connect('clicked', self.do_uppercase, self.entry)

        self.hbox.add(self.entry)
        self.hbox.add(self.button)

        self.window.add(self.hbox)

        self.window.show_all()
```

```
    def destroy(self, widget, data=None):
        gtk.main_quit()

    def do_uppercase(self, widget, data=None):
        text = data.get_text().upper()
        data.set_text(text)

app = Main()
gtk.main()
```

signal_multi.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Signal')
        self.window.connect('destroy', self.destroy)

        self.button = gtk.Button('button')
        self.button.connect('clicked', self.button_click)
        self.button.connect('clicked', self.button_click2)
        self.button.connect('enter', self.button_enter)
        self.button.connect('leave', self.button_leave)

        self.window.add(self.button)

        self.window.show_all()
```

```
    def destroy(self, widget, data=None):
        gtk.main_quit()

    def button_click(self, widget):
        print 'Button click'

    def button_click2(self, widget):
        print 'Button click (2)'

    def button_enter(self, widget):
        print 'Button mouse enter'

    def button_leave(self, widget):
        print 'Button mouse leave'

app = Main()
gtk.main()
```

signal_block.py:

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

import pygtk
pygtk.require('2.0')
import gtk
import time

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Signal block')
        self.window.connect('destroy', gtk.main_quit)

        self.vbox = gtk.VBox(homogeneous=True)
```

```
        self.hbox = gtk.HBox(homogeneous=True)

        self.label = gtk.Label()

        self.handle = -1

        self.button1 = gtk.Button('Button 1')
        self.handle = self.button1.connect('clicked', self.button1_click,
self.label)
        self.button2 = gtk.Button('Button 1 block')
        self.button2.connect('clicked', self.button2_click)
        self.button3 = gtk.Button('Button 1 unblock')
        self.button3.connect('clicked', self.button3_click)

        self.hbox.add(self.button1)
        self.hbox.add(self.button2)
        self.hbox.add(self.button3)

        self.vbox.add(self.hbox)
        self.vbox.add(self.label)

        self.window.add(self.vbox)

        self.window.show_all()

    def button1_click(self, widget, data=None):
        data.set_text(time.asctime())

    def button2_click(self, widget, data=None):
        self.button1.handler_block(self.handle)
        print 'block, handle %d' %(self.handle)

    def button3_click(self, widget, data=None):
        self.button1.handler_unblock(self.handle)
        print 'unblock, handle %d' %(self.handle)
```

```
app = Main()
gtk.main()
```

signal_disconnect.py:

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

import pygtk
pygtk.require('2.0')
import gtk
import time

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Signal disconnect')
        self.window.connect('destroy', gtk.main_quit)

        self.vbox = gtk.VBox(homogeneous=True)
        self.hbox = gtk.HBox(homogeneous=True)

        self.label = gtk.Label()

        self.handle = -1

        self.button1 = gtk.Button('Button 1')
        self.button2 = gtk.Button('Button 1 Connect')
        self.button2.connect('clicked', self.button2_click)
        self.button3 = gtk.Button('Button 1 Disconnect')
        self.button3.connect('clicked', self.button3_click)

        self.hbox.add(self.button1)
```

```
        self.hbox.add(self.button2)
        self.hbox.add(self.button3)

        self.vbox.add(self.hbox)
        self.vbox.add(self.label)

        self.window.add(self.vbox)

        self.window.show_all()

    def button1_click(self, widget, data=None):
        data.set_text(time.asctime())

    def button2_click(self, widget, data=None):
        if not self.button1.handler_is_connected(self.handle):
            self.handle = self.button1.connect('clicked',
self.button1_click, self.label)
            print 'connect, handle %d' %(self.handle)

    def button3_click(self, widget, data=None):
        if self.button1.handler_is_connected(self.handle):
            self.button1.disconnect(self.handle)
            print 'disconnect, handle %d' %(self.handle)


app = Main()
gtk.main()
```

signal_emit.py:

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

import pygtk
pygtk.require('2.0')
```

```
import gtk
import time

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Signal emit')
        self.window.connect('destroy', gtk.main_quit)

        self.hbox = gtk.HBox(homogeneous=True)

        self.button1 = gtk.Button('Button 1')
        self.button1.connect('clicked', self.button1_click)
        self.button2 = gtk.Button('Click Button 1')
        self.button2.connect('clicked', self.button2_click)

        self.hbox.add(self.button1)
        self.hbox.add(self.button2)

        self.window.add(self.hbox)

        self.window.show_all()

    def button1_click(self, widget, data=None):
        print 'Button 1 click'

    def button2_click(self, widget, data=None):
        self.button1.emit('clicked')


app = Main()
gtk.main()
```

Catatan contoh:

- gtk.main_quit() dapat digunakan untuk keluar dari mainloop gtk.
- Di contoh signal_simple.py, window kini bisa diclose dengan klik pada tombol close.
- Di contoh signal_hello.py, begitu tombol uppercase diklik, maka teks pada entry akan di-uppercase.
- Di contoh signal_multi.py, beberapa signal pada satu widget akan ditangani. Salah satu signal akan ditangani oleh lebih dari satu callback.
- Contoh signal_block.py mendemonstrasikan block dan unblock signal-callback.
- Contoh signal_disconnect.py mendemonstrasikan diskoneksi signal-callback.
- Contoh signal_emit.py mendemonstrasikan pengaktifan (emit) signal tertentu.

Koneksi event-callback

Pengaturan koneksi sama dengan pengaturan pada signal. Sementara, definisi callback sedikit berbeda. Untuk hal-hal lainnya, rujuklah juga kepada pembahasan koneksi signal-callback sebelumnya.

Definisi callback

Callback secara umum dapat didefinisikan sebagai berikut:

```
def callback (widget, event, data)
```

Apabila argumen opsional tidak digunakan pada connect(), maka callback secara umum dapat didefinisikan sebagai berikut:

```
def callback (widget, event, data=None)
```

atau

```
def callback (widget, event)
```

Catatan:

- Callback harus menyesuaikan dengan jumlah argumen opsional yang diberikan.
- Apabila callback digunakan dalam class, maka argumen pertama pada definisi callback tetap adalah self.

- Argumen event akan dilewatkan sebagai GdkEvent. Bacalah referensi untuk gtk.gdk.Event.
- Return value callback, apabila True, diartikan sebagai event telah ditangani dan tidak dipropagasi lagi. Sementara, apabila False, event handling normal akan dilakukan.

Daftar event

Daftar event diantaranya:

- event
- button_press_event
- button_release_event
- scroll_event
- motion_notify_event
- delete_event
- destroy_event
- expose_event
- key_press_event
- key_release_event
- enter_notify_event
- leave_notify_event
- configure_event
- focus_in_event
- focus_out_event
- map_event
- unmap_event
- property_notify_event
- selection_clear_event
- selection_request_event
- selection_notify_event
- proximity_in_event
- proximity_out_event
- visibility_notify_event
- client_event
- no_expose_event
- window_state_event

Representasi tipe data Python dapat dibaca pada referensi untuk gtk.gdk.Event.

event_simple.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Event')
        self.window.connect('destroy', self.destroy)
        self.window.connect('delete_event', self.delete_event)

        self.window.show_all()

    def delete_event(self, widget, event, data=None):
        #return True #will NOT be destroyed
        return False #will be destroyed

    def destroy(self, widget):
        gtk.main_quit()

app = Main()
gtk.main()
```

event_3button.py:

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

```
import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Event')
        self.window.connect('destroy', self.destroy)

        self.button = gtk.Button('triple click me')
        self.button.connect('button_press_event', self.button_tripleclick)

        self.window.add(self.button)

        self.window.show_all()

    def button_tripleclick(self, widget, event):
        if event.type == gtk.gdk._3BUTTON_PRESS:
            print 'Great!'

    def destroy(self, widget):
        gtk.main_quit()

app = Main()
gtk.main()
```

event_keypress.py:

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

import pygtk
pygtk.require('2.0')
import gtk
```

```
class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Event')
        self.window.set_size_request(300, 200)
        self.window.connect('destroy', self.destroy)

        self.label = gtk.Label()

        self.window.connect('key_press_event', self.window_keypress,
self.label)

        self.window.add(self.label)

        self.window.show_all()

    def window_keypress(self, widget, event, label):
        label.set_label(event.string)

    def destroy(self, widget):
        gtk.main_quit()

app = Main()
gtk.main()
```

event_keypress2.py:

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

import pygtk
pygtk.require('2.0')
import gtk
```

```
class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title('Event')
        self.window.set_size_request(500, 200)
        self.window.connect('destroy', self.destroy)

        self.label = gtk.Label()

        self.window.connect('key_press_event', self.window_keypress,
self.label)

        self.window.add(self.label)

        self.window.show_all()

    def window_keypress(self, widget, event, label):
        state = event.state.value_names
        if state:
            text =  '+'.join(state) + ' ' + event.string
        else:
            text = event.string
        label.set_label(text)

    def destroy(self, widget):
        gtk.main_quit()

app = Main()
gtk.main()
```

Catatan contoh:

- Contoh event_simple.py mendemonstrasikan delete_event untuk window (window ditutup). Apabila callback mengembalikan True, maka event

dianggap telah ditangani. Window tidak ditutup. Apabila mengembalikan False, maka window ditutup. Penanganan event seperti ini umum ditemukan seperti pada konfirmasi keluar dari program.

- Contoh event_3button.py mendemonstrasikan button_press_event. Tulisan 'Great!' akan dicetak ke stdout apabila tombol ditriple-click. Event button_press_event adalah event ketika tombol mouse ditekan. Kita perlu memeriksa atribut type milik event untuk mengetahui apakah tombol mouse ditriple-click (gtk.gdk._3BUTTON_PRESS).
- Contoh event_keypress.py mendemonstrasikan key_press_event. Apa yang dilakukan hanyalah mengatur label milik self.label menjadi event.string.
- Contoh event_keypress2.py melengkapi event_keypress2.py sehingga dapat mendeteksi penekanan CTRL, SHIFT, ALT(MOD1) dan lainnya.

Beragam widget

GTK+ menyediakan widget set lengkap untuk membangun aplikasi GUI. Widget-widget tersebut diturunkan dari gtk.Widget, dan memiliki diantaranya beberapa method berikut:

- set_sensitive(sensitive). Apabila True, maka widget dapat digunakan (enabled). Sebaliknya, widget tidak dapat digunakan (disabled).
- show(). Menampilkan widget.
- show_all(). Menampilkan widget dan setiap child widget.
- hide(). Menyembunyikan widget.
- hide_all(). Menyembunyikan widget dan setiap child widget.
- destroy(). Menghapus widget. Apabila widget berada dalam container, maka widget akan dihapus dari container. Apabila widget adalah toplevel, maka semua child widget juga akan dihapus. Oleh karena itu, destroy() umumnya hanya perlu digunakan pada toplevel.
- reparent(new_parent). Memindahkan widget dari satu container ke container lainnya.
- grab_focus(). Menjadikan widget memiliki fokus keyboard.
- grab_default(). Menjadikan suatu widget menjadi default, yang dapat diaktifkan ketika user menekan Enter.
- set_size_request(width, height). Mengatur ukuran minimum widget.

Untuk informasi selengkapnya, lihatlah class reference untuk widget yang digunakan, ditambah dengan class reference untuk class-class orangtuanya.

gtk.Label

Widget untuk menampilkan teks terbatas, yang hanya dapat dibaca.

```
gtk.Label(str=None)
```

Contoh:

```
w = gtk.Label('Hello')
```

gtk.Button

Widget tombol, yang dapat pula berisikan gambar tertentu (stock atau eksternal). Gunakan underscore sebelum karakter accelerator key dalam label button.

```
gtk.Button(label=None, stock=None, use_underline=True)
```

Contoh:

```
w = gtk.Button('_Save', gtk.STOCK_SAVE)
```

gtk.ToggleButton

Merupakan button dengan fitur dapat di-toggle. Gunakan underscore sebelum karakter accelerator key dalam label button.

```
gtk.ToggleButton(label=None, use_underline=True)
```

Contoh:

```
w = gtk.ToggleButton('Option _1')
```

gtk.CheckButton

Merupakan toggle button dengan checkbox dan label. Gunakan underscore sebelum karakter accelerator key dalam label button.

```
gtk.CheckButton(label=None, use_underline=True)
```

Contoh:

```
w = gtk.CheckButton('Option _1')
```

gtk.RadioButton

Merupakan checkbutton namun hanya satu yang dapat aktif dalam satu group. Dapat digunakan sebagai pilihan berganda.

```
gtk.RadioButton(group=None, label=None, use_underline=True)
```

Catatan:

- Argumen group akan membentuk group radio button.
- Group baru: group = None.
- Anggota group: group = radio button yang mengawali group.
- Group dapat pula diset dengan set_group(group).

Contoh:

```
w1 = gtk.RadioButton(None, 'Option _1')
w2 = gtk.RadioButton(w1, 'Option _2')
w3 = gtk.RadioButton(w1, 'Option _3')
```

gtk.SpinButton

SpinButton dapat digunakan untuk mendapatkan nilai integer atau floating-point dari user, dimana user dapat klik pada tombol panah atas dan bawah untuk menambahkan atau mengurangi nilai.

```
gtk.SpinButton(adjustment=None, climb_rate=0.0, digits=0)
```

Catatan:

Untuk memudahkan pengaturan nilai, batas bawah, batas atas, increment, page increment dan page size, buatlah gtk.Adjustment terlebih dahulu.

```
gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0,
page_size=0)
```

Contoh:

```
adj = gtk.Adjustment(0, 0, 10, 1, 4)
w = gtk.SpinButton(adj)
```

gtk.ColorButton

Merupakan button yang ketika diklik, akan membuka dialog pemilihan warna. Warna yang dipilih kemudian akan tampil pada button.

```
gtk.ColorButton(color=gtk.gdk.Color(0,0,0))
```

Catatan:

- Untuk mendapatkan warna aktif/yang dipilih, gunakanlah get_color(), yang akan mengembalikan objek gtk.gdk.Color.

```
gtk.gdk.Color(red=0, green=0, blue=0, pixel=0)
```

- Widget untuk pemilihan warna (yang tampil pada dialog) adalah gtk.ColorSelection.
- Dialog untuk pemilihan warna adalah gtk.ColorSelectionDialog.

Contoh:

```
w = gtk.ColorButton()
```

gtk.FontButton

Merupakan button yang ketika diklik, akan membuka dialog pemilihan font. Font yang dipilih kemudian akan tampil pada button.

```
gtk.FontButton(fontname=None)
```

Catatan:

- Gunakan get_font_name() untuk mendapatkan nama font.
- Widget untuk pemilihan font (yang tampil pada dialog) adalah gtk.FontSelection.
- Dialog untuk pemilihan font adalah gtk.FontSelectionDialog.

Contoh:

```
w = gtk.FontButton()
```

gtk.FileChooserButton

Merupakan button yang ketika diklik, akan membuka dialog pemilihan file. Nama file yang dipilih kemudian akan tampil pada button.

```
gtk.FileChooserButton(title, backend=None)
gtk.FileChooserButton(dialog)
```

Catatan:

- Untuk mendapatkan pilihan user dan pengaturan lainnya, lihatlah method-method pada referensi class gtk.FileChooser.
- Dialog untuk pemilihan file adalah gtk.FileChooserDialog.
- Lihatlah juga gtk.FileChooserWidget dan gtk.FileSelection.

Contoh:

```
w = gtk.FileChooserButton('Pilih file')
w.set_current_folder('/tmp')
```

gtk.Entry

Entry adalah widget yang menyediakan input berupa satu baris teks.

```
gtk.Entry(max=0)
```

Catatan:

- Untuk input berupa password, dimana setiap karakter yang diketik user di-mask dengan karakter tertentu, seperti * atau x, lakukan langkah-langkah:
  - set_visibility(False)
  - set_invisible_char(ch). Contoh: set_invisible_char('x'). Nilai default ch adalah *.
- Untuk mendapatkan teks yang diinput, gunakanlah get_text().
- Untuk dapat bekerja dengan text completion (dari sejumlah predefined-text), sehingga user dapat menekan beberapa karakter pertama dan sebuah popup akan ditampilkan sesuai input user, lakukanlah langkah-langkah berikut:
  - Buat sebuah gtk.ListStore dengan satu kolom bertipe str.
  - Tambahkan setiap predefined-text ke objek gtk.ListStore yang dibuat, dalam bentuk sequence (contoh: list).
  - Buat sebuah gtk.EntryCompletion dan set_model ke objek gtk.ListStore yang dibuat sebelumnya.
  - Tentukan kolom pada model untuk mendapatkan teks. Dalam contoh completionn sederhana ini, teks didapat dari kolom 0. Gunakanlah gtk.EntryCompletion.set_text_column(0).
  - Set completion Entry dengan gtk.Entry.set_completion(completion).
  - Lebih lanjut, lihatlah pada contoh 4.

Contoh 1:

```
w = gtk.Entry(3) #maksimal 3
```

Contoh 2:

```
w = gtk.Entry()
w.set_text('input your password here')
```

Contoh 3:

```
w = gtk.Entry()
w.set_visibility(False)
w.set_invisible_char('x')
```

Contoh 4:

```
w = gtk.Entry()

liststore = gtk.ListStore(str)
```

```
texts = ['abc', 'bcd', 'bce', 'cde', 'cdf', 'def', 'deg']
for t in texts:
     liststore.append([t])

completion = gtk.EntryCompletion()
completion.set_model(liststore)
completion.set_text_column(0)

w.set_completion(completion)
```

gtk.ComboBox

ComboBox menyediakan pilihan untuk user, dalam bentuk drop down list. ComboBox di PyGTK dapat digunakan setidaknya dengan dua cara:

- cara sederhana
- menggunakan gtk.TreeModel

Cara sederhana

Dalam cara sederhana, combobox dibuat dengan:

```
gtk.combo_box_new_text()
```

Fungsi tersebut akan mengembalikan objek gtk.ComboBox untuk item berupa teks. Selanjutnya, untuk bekerja dengan combobox:

- append (menambahkan di akhir) teks: gtk.ComboBox.append_text(text)
- insert teks: gtk.ComboBox.insert_text(position, text)
- prepend (menambahkan di awal): gtk.ComboBox.prepend_text(text)
- menghapus teks: gtk.ComboBox.remove_text(position)
- mendapatkan item terpilih: gtk.ComboBox.get_active_text()

Contoh:

```
w = gtk.combo_box_new_text()
w.append_text('Pilihan 2')
w.prepend_text('Pilihan 1')
w.append_text('Pilihan 3')
```

```
w.insert_text(2, 'Pilihan 2 lain')
w.remove_text(2)
```

Menggunakan gtk.TreeModel

Untuk cara yang lebih kompleks, namun fleksibel dan powerful (seperti dapat menambahkan image), combobox dapat dibuat dengan:

```
gtk.ComboBox(model=None)
```

Argumen model sendiri adalah gtk.TreeModel. Berikut adalah contoh untuk menghadirkan combobox dengan beberapa item string (model adalah gtk.ListStore).

```
liststore = gtk.ListStore(str)
texts = ['Pilihan 1', 'Pilihan 2', 'Pilihan 3']
for t in texts:
     liststore.append([t])

w = gtk.ComboBox(liststore)
cell = gtk.CellRendererText()
w.pack_start(cell, True)
w.add_attribute(cell, 'text', 0)
```

Catatan:

Untuk menyediakan fasilitas combobox bagi user, namun user dapat mengisikan langsung nilai selain yang terdapat pada pilihan, gunakanlah gtk.ComboBoxEntry. Prinsip penggunaannya mirip dengan gtk.ComboBox dan dapat dibuat secara mudah dengan gtk.combo_box_entry_new_text().

Lihatlah referensi class untuk gtk.ComboBoxEntry.

gtk.Frame

Frame merupakan sebuah container, dengan bingkai dan label opsional.

```
gtk.Frame(label=None)
```

Contoh:

```
w = gtk.Frame('Configuration')
```

gtk.Image

Image merupakan widget yang dapat menampilkan gambar. Gambar bisa didapat diantaranya dari pixmap, gtk.gdk.image, file, gtk.gdk.Pixbuf, GTK stock, icon set, atau gtk.gdk.PixbufAnimation.

```
gtk.Image()
```

Catatan:

- Untuk mengambil gambar dari file, gunakan gtk.Image.set_from_file(filename)
- Untuk menghapus gambar, gunakan gtk.Image.clear()

Contoh:

```
w = gtk.Image()
w.set_from_file('./smile.png')
```

gtk.Tooltips

Tooltips dapat digunakan untuk menambahkan tips (hint) ke suatu widget.

```
gtk.Tooltips()
```

Catatan:

Gunakan gtk.Tooltips.set_tip(widget, tip_text, tip_private=None) untuk mengatur tip untuk suatu widget dengan tip adalah tip_text.

Contoh:

```
w = gtk.Button('Klik')
t = gtk.Tooltips()
t.set_tip(w, 'Klik untuk keluar dari aplikasi')
```

gtk.Scale

Scale dapat digunakan untuk memilih nilai tertentu dalam batasan, dengan mempergunakan slider. Terdapat dua jenis scale:

- gtk.HScale: gtk.HScale(adjustment=None)
- gtk.VScale: gtk.VScale(adjustment=None)

Keduanya diturunkan dari gtk.Scale, dan gtk.Scale sendiri diturunkan dari gtk.Range. Beberapa method gtk.Range:

- mendapatkan value: gtk.Range.get_value()
- mengatur range: gtk.Range.set_range(min, max)
- memberikan nilai tertentu: gtk.Range.set_value(value)
- mengatur adjustment: gtk.Range.set_adjustment(adjustment)

Catatan:

Untuk memudahkan pengaturan nilai, batas bawah, batas atas, increment, page increment dan page size, buatlah gtk.Adjustment terlebih dahulu.

```
gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0,
page_size=0)
```

Contoh:

```
adj = gtk.Adjustment(1, 1, 10, 1)
w = gtk.VScale(adj)
```

gtk.ProgressBar

Widget Progressbar umum digunakan untuk menampilkan progres secara visual.

```
gtk.ProgressBar()
```

Catatan:

- Mengatur teks pada progress: gtk.ProgressBar.set_text(text).
- Mengatur nilai pada progress: gtk.ProgressBar.set_fraction(fraction). Nilai fraction antara 0.0 dan 1.0.
- Mendapatkan nilai progress: gtk.ProgressBar.get_fraction(). Nilai fraction antara 0.0 dan 1.0.

- Orientasi progressbar dapat diset dengan: gtk.ProgressBar.set_orientation(orientation), dimana orientation adalah salah satu dari: gtk.PROGRESS_LEFT_TO_RIGHT, gtk.PROGRESS_RIGHT_TO_LEFT, gtk.PROGRESS_BOTTOM_TO_TOP atau gtk.PROGRESS_TOP_TO_BOTTOM.

Contoh:

```
w = gtk.ProgressBar()
val = 0.5
w.set_fraction(val)
w.set_text('Percentage: %d%%' %(val*100))
w.set_orientation(gtk.PROGRESS_BOTTOM_TO_TOP)
```

gtk.ScrolledWindow

Widget ScrolledWindow akan menambahkan scrollbar ke child widget. Sangat berguna untuk child widget yang mungkin membutuhkan fungsionalitas scroll seperti gtk.TextView.

```
gtk.ScrolledWindow(hadjustment=None, vadjustment=None)
```

Catatan:

Untuk mengatur bagaimana scrollbar ditampilkan, gunakan:

```
gtk.ScrolledWindow.set_policy(hscrollbar_policy, vscrollbar_policy)
```

hscrollbar_policy dan vscrollbar_policy dapat berupa: gtk.POLICY_ALWAYS (scrollbar selalu tampil), gtk.POLICY_AUTOMATIC (scrollbar tampil ketika diperlukan), gtk.POLICY_NEVER (scrollbar tidak pernah ditampilkan).

Contoh:

Lihat pembahasan gtk.TextView.

gtk.TextView

TextView dapat digunakan untuk menampilkan teks, baik sederhana (hanya teks berukuran kecil) ataupun kompleks (teks dengan atribut tertentu ataupun widget, berukuran kecil ataupun besar).

```
gtk.TextView(buffer=None)
```

Catatan:

- Widget TextView sangatlah powerful. Training hanya akan membahas fungsionalitas dasar, yaitu menampilkan teks.
- Teks yang ditampilkan dan atributnya disimpan dalam gtk.TextBuffer.

```
gtk.TextBuffer(table=None)
```

Contoh:

```
text = ''
for i in range(1, 100):
     text += 'line %d\n' %(i)

buf  = gtk.TextBuffer()
buf.set_text(text)

w = gtk.TextView()
w.set_buffer(buf)

scrolled = gtk.ScrolledWindow()
scrolled.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
scrolled.add(w)
```

gtk.Statusbar

Statusbar dapat digunakan untuk menampilkan status aplikasi, dan umum diposisikan di bagian bawah window. Widget ini memiliki sebuah resize grip yang dapat digunakan untuk meresize window yang mengandung statusbar.

Statusbar di PyGTK bekerja dengan konsep stack, dimana teks status bisa dipush ke stack atau dipop dari stack. Teks status yang paling atas yang akan ditampilkan.

Catatan:

- Ketika mem-push sebuah teks status dengan gtk.Statusbar.push(context_id, text), context_id bisa didapatkan dengan gtk.Statusbar.get_context_id(context_description).

- Untuk mem-pop teks status dari stack, gunakan gtk.Statusbar.pop(context_id).
- Untuk menentukan apakah resize grip ditampilkan, gunakan set_has_resize_grip(setting). Argumen setting bernilai True atau False.
- gtk.Statusbar diturunkan dari gtk.HBox dan oleh karenanya, kita dapat menambahkan widget lain (seperti gtk.ProgressBar) ke dalamnya.

Contoh 1:

```
w = gtk.Statusbar()
id = w.get_context_id('info')
w.push(id, 'System ready.')
```

Contoh 2:

```
w = gtk.Statusbar()
id = w.get_context_id('info')
w.push(id, 'System ready.')
w.pop(id) #no more status text
```

Contoh 3:

```
w = gtk.Statusbar()
id = w.get_context_id('info')
w.push(id, 'System ready.')
w.set_has_resize_grip(False)
```

Contoh 4:

```
w = gtk.Statusbar()
w.push(w.get_context_id('info'), 'System ready.')
p = gtk.ProgressBar()
p.set_fraction(0.5)
w.pack_end(p)
```

gtk.Expander

Expander merupakan widget container yang dapat menyembunyikan child widgetnya menggunakan sebuah tombol panah kecil.

```
gtk.Expander(label=None)
```

Catatan:

Agar accelerator key dapat digunakan dengan karakter underline, gunakan gtk.Expander.set_use_underline(use_underline), dengan use_underline adalah True.

Contoh:

```
lbl = gtk.Label('Detail text')
w = gtk.Expander('_Detail')
w.set_use_underline(True)
w.add(lbl)
```

gtk.ButtonBox

ButtonBox dapat digunakan untuk menempatkan sekelompok tombol secara konsisten. Developer dapat menggunakan:

- gtk.HButtonBox: gtk.HButtonBox()
- gtk.VButtonBox: gtk.VButtonBox()

Catatan:

- Untuk mengatur layout tombol, gunakanlah gtk.ButtonBox.set_layout(layout_style) dengan layout_style adalah salah satu dari gtk.BUTTONBOX_SPREAD, gtk.BUTTONBOX_EDGE, gtk.BUTTONBOX_START atau gtk.BUTTONBOX_END.
- Untuk menambahkan tombol, gunakan: gtk.Container.add(widget), gtk.Box.pack_start(child, expand=True, fill=True, padding=0) atau gtk.Box.pack_end(child, expand=True, fill=True, padding=0).
- Untuk mengatur spacing, gunakan: gtk.Box.set_spacing(spacing).

Contoh:

```
b1 = gtk.Button('Button 1')
b2 = gtk.Button('Button 2')
b3 = gtk.Button('Button 3')
b4 = gtk.Button('Button 4')
w = gtk.HButtonBox()
w.set_layout(gtk.BUTTONBOX_SPREAD)
w.pack_start(b1)
w.pack_start(b2)
```

```
w.pack_start(b3)
w.pack_start(b4)
```

gtk.Paned

Paned dapat digunakan untuk menempatkan dua widget, dimana diantara keduanya dipisahkan oleh resizer. Developer dapat menggunakan:

- gtk.HPaned: gtk.HPaned()
- gtk.VPaned: gtk.VPaned()

Catatan:

- Untuk menambahkan widget di sisi kiri atau atas, gunakan gtk.Paned.add1(child) atau gtk.Paned.pack1(child, resize=False, shrink=True).
- Untuk menambahkan widget di sisi kanan atau bawah, gunakan gtk.Paned.add2(child) atau gtk.Paned.pack2(child, resize=True, shrink=True).

Contoh 1:

```
b1 = gtk.Button('Button 1')
b2 = gtk.Button('Button 2')
w = gtk.HPaned()
w.pack1(b1)
w.pack2(b2)
```

Contoh 2:

```
b1 = gtk.Button('Button 1')
b2 = gtk.Button('Button 2')
box = gtk.HButtonBox()
box.set_spacing(10)
box.set_layout(gtk.BUTTONBOX_END)
box.add(b1)
box.add(b2)
```

```
t = gtk.TextView()
```

```
w = gtk.VPaned()
w.pack1(t)
w.pack2(box)
```

gtk.Notebook

Notebook menyediakan container yang dilengkapi dengan tab. Sangat umum ditemukan dalam dialog preferensi aplikasi.

```
gtk.Notebook()
```

Catatan:

- Notebook adalah widget yang kompleks. Rujuklah ke referensi class gtk.Notebook untuk informasi selengkapnya.
- Untuk menambahkan (append) tab baru, gunakan: gtk.Notebook.append_page(child, tab_label=None). Argumen child adalah widget yang ingin ditambahkan. Sementara, argumen tab_label merupakan widget yang digunakan sebagai label suatu tab. Dengan demikian, label tab tidak harus selalu teks (lihat contoh 2). Method akan mengembalikan indeks halaman.
- Untuk sekedar mengatur title tab, gunakan: gtk.Notebook.set_tab_label_text(child, tab_text). Argumen child adalah child widget yang ditambahkan pada notebook. Sementara, tab_text adalah title untuk widget tersebut.
- Untuk mengatur posisi tab, gunakan: gtk.Notebook.set_tab_pos(pos), dimana pos adalah salah satu dari: gtk.POS_LEFT, gtk.POS_RIGHT, gtk.POS_TOP atau gtk.POS_BOTTOM.
- Untuk mengatur agar area tab notebook berisikan tombol untuk scroll apabila jumlah tab terlalu banyak, gunakan: gtk.Notebook.set_scrollable((scrollable), dimana scrollable adalah True atau False.
- Untuk menghadirkan popup menu berisikan tab-tab yang ada, gunakan: gtk.Notebook.popup_enable().

Contoh 1:

```
t = gtk.TextView()
f = gtk.FontSelection()
```

```
w = gtk.Notebook()

p1 = w.append_page(t)
p2 = w.append_page(f)
w.set_tab_label_text(t, 'Text View')
w.set_tab_label_text(f, 'Select font')

w.set_tab_pos(gtk.POS_BOTTOM)
w.set_scrollable(True)
w.popup_enable()
```

Contoh 2:

```
t = gtk.TextView()
f = gtk.FontSelection()

t_title = gtk.Image()
t_title.set_from_file('./smile.png')

w = gtk.Notebook()

p1 = w.append_page(t, t_title)
p2 = w.append_page(f)
w.set_tab_label_text(f, 'Select font')
```

gtk.Calendar

Calendar merupakan widget yang menampilkan kalender, sehingga pemilihan tanggal dapat dilakukan dengan mudah.

```
gtk.Calendar()
```

Catatan:

- Secara default, tanggal hari ini akan ditampilkan.
- Untuk mengatur bulan dan tahun, gunakan: gtk.Calendar.select_month(month, year). Argumen month berada dalam range 0 (Januari) sampai 11 (Desember).

- Untuk mengatur tanggal, gunakan: gtk.Calendar.select_day(day). Argumen day adalah dari 1 sampai 31. Apabila diisi dengan 0, maka tanggal tidak lagi terpilih.
- Untuk menandai tanggal tertentu secara visual (misal: diformat bold), gunakan: gtk.Calendar.mark_day(day). Untuk menghilangkan penanda, gunakan gtk.Calendar.unmark_day(day).
- Untuk membersihkan semua penanda tanggal, gunakan: gtk.Calendar.clear_marks().
- Untuk mendapatkan tanggal terpilih, gunakan: gtk.Calendar.get_date(), yang akan mengembalikan tuple berisikan year, month dan day.
- Untuk mengatur opsi penampilan kalender, gunakan gtk.Calendar.set_display_options(flags), dimana flags merupakan kombinasi dari: gtk.CALENDAR_SHOW_HEADING (bulan dan tanggal ditampilkan), gtk.CALENDAR_SHOW_DAY_NAMES (nama hari dalam 3 huruf ditampilkan), gtk.CALENDAR_NO_MONTH_CHANGE (bulan tidak dapat diganti), gtk.CALENDAR_SHOW_WEEK_NUMBERS (menampilkan nomor minggu dalam tahun tersebut), gtk.CALENDAR_WEEK_START_MONDAY (mengawali minggu pada senin, bukan default minggu).

Contoh:

```
w = gtk.Calendar()
w.select_month(0, 2020)
w.select_day(1)
w.mark_day(27)
w.mark_day(17)
w.set_display_options(  gtk.CALENDAR_WEEK_START_MONDAY |
                        gtk.CALENDAR_NO_MONTH_CHANGE |
                        gtk.CALENDAR_SHOW_WEEK_NUMBERS |
                        gtk.CALENDAR_SHOW_DAY_NAMES |
                        gtk.CALENDAR_SHOW_HEADING)
```

gtk.Menu, gtk.MenuItem, gtk.MenuBar

Untuk membuat menubar lengkap dengan menu dan menu item, dua cara berikut bisa dilakukan:

- manual: menu disusun item demi item secara manual.
- menggunakan gtk.UIManager: pembuatan menu dengan cara mudah.

Materi training hanya akan menggunakan cara manual.

Untuk membuat menu secara manual, lakukanlah langkah-langkah berikut:

- Buatlah sebuah objek gtk.MenuBar, yang akan berfungsi sebagai menubar. Menubar ini dapat ditambahkan ke container seperti objek lainnya.
- Untuk setiap menu (seperti File, Edit, Help, dan lainnya):
  - Buat sebuah menu (gtk.Menu), yang merepresentasikan menu yang dibuat.
  - Buat berbagai menuitem (gtk.MenuItem), yang mewakili setiap item menu (seperti Open, Save, About, dan lainnya). Menu item tidak harus berupa teks (gtk.MenuItem), tapi bisa berupa separator (gtk.SeparatorMenuItem), tearoff (gtk.TearoffMenuItem), check button (gtk.CheckMenuItem), radio button (gtk.RadioMenuItem), dan image (gtk.ImageMenuItem).
  - Tambahkan setiap menuitem yang dibuat ke menu (gtk.Menu, yang dibuat sebelumnya) dengan gtk.MenuShell.append(child). Argumen child adalah menuitem yang dibuat.
  - Buat satu lagi menuitem (gtk.MenuItem), yang akan menjadi menu yang terlihat pada menubar, yang merupakan menu seperti File, Edit, Help dan lain sebagainya. Setelah itu, set submenu milik menuitem tersebut dengan menu (gtk.Menu) yang dibuat sebelumnya. Untuk mengatur submenu, gunakan: gtk.MenuItem.set_submenu(submenu). Untuk menuitem yang ingin ditempatkan rata kanan (seperti Help pada umumnya), gunakan: gtk.MenuItem.set_right_justified(True).
  - Tambahkan menuitem yang dibuat sebelumnya (mewakili File, Edit, Help dan lain sebagainya) ke menubar dengan: gtk.MenuShell.append(child).
- Catatan: langkah-langkah manual ini pada awalnya terlihat rumit, namun fleksibel dan powerful.

gtk.MenuBar:

gtk.MenuBar()

gtk.Menu:

gtk.Menu()

gtk.MenuItem dan turunannya:

gtk.MenuItem(label=None, use_underline=True)

gtk.CheckMenuItem(label=None, use_underline=True)

gtk.ImageMenuItem(stock_id=None, accel_group=None)

gtk.RadioMenuItem(group=None, label=None, use_underline=True)

```
gtk.SeparatorMenuItem()
gtk.TearoffMenuItem()
```

Contoh:

```
menubar = gtk.MenuBar()

#file
menu_file = gtk.Menu()
item_new = gtk.MenuItem('_New')
item_open = gtk.MenuItem('_Open')
item_save = gtk.MenuItem('_Save')
item_quit = gtk.MenuItem('_Quit')
sep1 = gtk.SeparatorMenuItem()
menu_file.append(item_new)
menu_file.append(item_open)
menu_file.append(item_save)
menu_file.append(sep1)
menu_file.append(item_quit)

item_file = gtk.MenuItem('_File')
item_file.set_submenu(menu_file)



#extra, check, radio, tearoff
menu_extra = gtk.Menu()
item_tearoff = gtk.TearoffMenuItem()
item_check1 = gtk.CheckMenuItem('_Active')
item_radio1 = gtk.RadioMenuItem(None, 'Pilihan _1')
item_radio2 = gtk.RadioMenuItem(item_radio1, 'Pilihan _2')
item_radio3 = gtk.RadioMenuItem(item_radio1, 'Pilihan _3')
sep2 = gtk.SeparatorMenuItem()
menu_extra.append(item_tearoff)
menu_extra.append(item_check1)
```

```
menu_extra.append(sep2)
menu_extra.append(item_radio1)
menu_extra.append(item_radio2)
menu_extra.append(item_radio3)


item_extra = gtk.MenuItem('_Extra')
item_extra.set_submenu(menu_extra)



#help, right justify, image
menu_help = gtk.Menu()
item_content = gtk.MenuItem('_Content')
item_about = gtk.ImageMenuItem(gtk.STOCK_ABOUT)
menu_help.append(item_content)
menu_help.append(item_about)

item_help = gtk.ImageMenuItem(gtk.STOCK_HELP)
item_help.set_submenu(menu_help)
item_help.set_right_justified(True)




menubar.append(item_file)
menubar.append(item_extra)
menubar.append(item_help)
```

gtk.Toolbar, gtk.Toolitem

Untuk membuat toolbar lengkap dengan berbagai tombol dan item lain, dua cara berikut bisa dilakukan:

- manual: tombol dan item disusun secara manual.
- menggunakan gtk.UIManager: pembuatan toolbar dengan cara mudah.

Materi training hanya akan menggunakan cara manual.

Untuk membuat toolbar secara manual, lakukanlah langkah-langkah berikut:

- Buatlah terlebih dahulu semua item yang ingin ditempatkan pada toolbar. Item dapat berupa tombol (gtk.ToolButton), menu (gtk.MenuToolButton), radio button (gtk.RadioToolButton), separator (gtk.SeparatorToolItem) ataupun toggle button (gtk.ToggleToolButton).
- Buat objek gtk.Toolbar dan tambahkan item dengan gtk.Toolbar.insert(item, pos). Argumen item adalah item yang ingin ditambahkan dan pos adalah posisi. Posisi 0 adalah posisi paling awal. Posisi dapat pula diisikan sebagai bilangan negatif dan akan ditambahkan pada akhir toolbar.

gtk.Toolbar:

```
gtk.Toolbar()
```

gtk.ToolItem dan turunannya:

```
gtk.ToolButton(icon_widget=None, label=None)
gtk.MenuToolButton(stock_id)
gtk.RadioToolButton(group=None, stock_id=None)
gtk.SeparatorToolItem()
gtk.ToggleToolButton(stock_id=None)
```

Catatan:

Untuk item berupa gtk.MenuToolButton, pastikan semua menuitem ditampilkan dengan gtk.Widget.show().

Contoh:

```
btn_new = gtk.ToolButton(gtk.STOCK_NEW)
btn_fs = gtk.ToggleToolButton(gtk.STOCK_FULLSCREEN)
sep1 = gtk.SeparatorToolItem()
btn_quit = gtk.ToolButton(gtk.STOCK_QUIT)

menu_extra = gtk.Menu()
item_tearoff = gtk.TearoffMenuItem()
item_check1 = gtk.CheckMenuItem('_Active')
item_radio1 = gtk.RadioMenuItem(None, 'Pilihan _1')
```

```
item_radio2 = gtk.RadioMenuItem(item_radio1, 'Pilihan _2')
item_radio3 = gtk.RadioMenuItem(item_radio1, 'Pilihan _3')
sep = gtk.SeparatorMenuItem()
item_tearoff.show()
item_check1.show()
item_radio1.show()
item_radio2.show()
item_radio3.show()
sep.show()
menu_extra.append(item_tearoff)
menu_extra.append(item_check1)
menu_extra.append(sep)
menu_extra.append(item_radio1)
menu_extra.append(item_radio2)
menu_extra.append(item_radio3)

btn_menu = gtk.MenuToolButton(gtk.STOCK_OPEN)
btn_menu.set_menu(menu_extra)

w = gtk.Toolbar()
w.insert(btn_new, 0)
w.insert(btn_fs, 1)
w.insert(sep1, 2)
w.insert(btn_quit, 3)
w.insert(btn_menu, 4)
```

Dialog

Dialog umum digunakan diantaranya untuk meminta input, menampilkan pesan dan aksi lainnya. Materi training akan membahas gtk.Dialog untuk dialog umum (contoh: menampilkan pesan, meminta input) dan gtk.MessageDialog (menampilkan pesan dengan mudah).

gtk.Dialog

Widget gtk.Dialog terbagi dalam dua bagian secara vertikal, dipisahkan gtk.HSeparator:

- bagian atas: gtk.VBox, tempat developer mem-pack berbagai widget tambahan yang menyusun dialog.
- bagian bawah: action_area, sebuah gtk.HBox, tempat developer menempatkan berbagai tombol.

Pembuatan dialog

Untuk membuat gtk.Dialog, gunakan:

```
gtk.Dialog(title=None, parent=None, flags=0, buttons=None)
```

- title: title dialog
- parent: parent dialog
- flags: dapat merupakan kombinasi dari:
  - gtk.DIALOG_MODAL: mengatur dialog menjadi modal.
  - gtk.DIALOG_DESTROY_WITH_PARENT: didestroy ketika parentnya didestroy.
  - gtk.DIALOG_NO_SEPARATOR: separator tidak digunakan.
- buttons: tuple yang mengandung pasangan tombol/response ID (contoh: ('ok',1,'cancel',2,...)). Tombol dapat berupa teks atau GTK stock. Sementara, response ID adalah integer yang dapat berupa GTK response atau yang diset oleh developer. Daftar GTK response:
  - gtk.RESPONSE_NONE
  - gtk.RESPONSE_REJECT
  - gtk.RESPONSE_ACCEPT
  - gtk.RESPONSE_DELETE_EVENT
  - gtk.RESPONSE_OK
  - gtk.RESPONSE_CANCEL
  - gtk.RESPONSE_CLOSE
  - gtk.RESPONSE_YES
  - gtk.RESPONSE_NO
  - gtk.RESPONSE_APPLY
  - gtk.RESPONSE_HELP

Catatan:

- Langkah-langkah dasar bekerja dengan dialog:
  - Buat dialog, yang akan mengembalikan objek gtk.Dialog. Set button langsung pada saat pembuatan atau tambahkan setelah ini.
  - Pack semua komponen GUI yang menyusun dialog ke vbox dialog.

  - Tampilkan dialog dengan gtk.Dialog.run(). Method ini akan mengembalikan response ID. Response ID bisa didapat dari response ID tombol yang diset dan/atau GTK response (lihat bagian pembuatan dialog sebelumnya).
  - Cek response ID dan lakukan proses yang diinginkan (dapatkan teks hasil input, dan lain sebagainya).
  - Hapus dialog dengan gtk.Widget.destroy().
- Untuk menambahkan button, selain pada saat pembuatan dialog juga dapat menggunakan:
  - gtk.Dialog.add_button(button_text, response_id)
  - gtk.Dialog.add_buttons(...), dengan argumen adalah pasangan tombol/response ID.
- Contoh berbagai dialog bisa dilihat pada dialogs.py.

dialogs.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(400,100)
        self.window.connect('destroy', gtk.main_quit)

        b1 = gtk.Button('Say hello')
        b1.connect('clicked', self.do_hello, self.window)
        b2 = gtk.Button('Get response')
        b2.connect('clicked', self.do_response, self.window)
        b3 = gtk.Button('Get input')
        b3.connect('clicked', self.do_input, self.window)

        self.hbox = gtk.HBox()
        self.hbox.pack_start(b1)
        self.hbox.pack_start(b2)
```

```
        self.hbox.pack_start(b3)

        self.window.add(self.hbox)
        self.window.show_all()

    def do_hello(self, widget, parent):
        buttons = (gtk.STOCK_OK, gtk.RESPONSE_OK)

        label = gtk.Label('Hello!')
        label.show()

        d = gtk.Dialog('Informasi', parent, gtk.DIALOG_MODAL, buttons)
        d.vbox.pack_start(label)

        d.run()
        d.destroy()

    def do_response(self, widget, parent):
        buttons = (gtk.STOCK_OK, gtk.RESPONSE_OK, 'No comment', 1000)

        label = gtk.Label('Hello!')
        label.show()

        d = gtk.Dialog('Konfirmasi', parent, gtk.DIALOG_MODAL, buttons)
        d.vbox.pack_start(label)

        result = d.run()
        d.destroy()

        if result == gtk.RESPONSE_OK:
            print 'You said: OK'
        elif result == 1000:
            print 'You said: No comment'
```

```
    def do_input(self, widget, parent):
        buttons = (gtk.STOCK_OK, gtk.RESPONSE_OK,
            gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL)

        label = gtk.Label('Your name')
        entry = gtk.Entry()

        hbox = gtk.HBox()
        hbox.pack_start(label)
        hbox.pack_start(entry)
        hbox.show_all()

        d = gtk.Dialog('Input', parent, gtk.DIALOG_MODAL, buttons)
        d.vbox.pack_start(hbox)

        result = d.run()

        if result == gtk.RESPONSE_OK:
            print 'Hi, %s' %(entry.get_text())

        d.destroy()

app = Main()
gtk.main()
```

gtk.MessageDialog

Untuk menampilkan pesan dengan mudah, lengkap dengan jenis dialog (informasi/info, peringatan/warning, pertanyaan/question, atau kesalahan/error) dan icon terkait, dengan teks yang mendukung Pango markup, gunakanlah gtk.MessageDialog.

```
gtk.MessageDialog(parent=None, flags=0, type=gtk.MESSAGE_INFO,
buttons=gtk.BUTTONS_NONE, message_format=None)
```

- parent: parent dialog

- flags: kombinasi gtk.DIALOG_MODAL dan gtk.DIALOG_DESTROY_WITH_PARENT (lihat pembahasan gtk.Dialog)
- type: jenis dialog, salah satu dari: gtk.MESSAGE_INFO, gtk.MESSAGE_WARNING, gtk.MESSAGE_QUESTION atau gtk.MESSAGE_ERROR.
- buttons: tombol-tombol yang telah didefinisikan: gtk.BUTTONS_NONE, gtk.BUTTONS_OK, gtk.BUTTONS_CLOSE, gtk.BUTTONS_CANCEL, gtk.BUTTONS_YES_NO, atau gtk.BUTTONS_OK_CANCEL.
- message_format: string pesan.

Catatan:

- Dialog mendukung rendering markup Pango. Untuk mengatur pesan agar dilengkapi dengan markup, gunakan: gtk.MessageDialog.set_markup(str). Lebih lanjut tentang Pango bisa ditemukan pada referensi class.
- Dialog mendukung teks tambahan/sekunder. Atur dengan:
  - gtk.MessageDialog.format_secondary_text(message_format)
  - gtk.MessageDialog.format_secondary_markup(message_format)
- Untuk mengganti gambar yang digunakan pada dialog, gunakan: gtk.MessageDialog.set_image(image). Argumen image merupakan gtk.Image.
- Widget gtk.MessageDialog diturunkan dari gtk.Dialog, lihatlah juga pembahasan gtk.Dialog.
- Untuk mengatur title dialog, gunakan: gtk.Window.set_title(title).
- Contoh berbagai dialog bisa dilihat pada msgdialogs.py.

msgdialogs.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(400,100)
        self.window.connect('destroy', gtk.main_quit)

        self.hbox = gtk.HBox()
```

```
        b1 = gtk.Button('Info')
        b1.connect('clicked', self.do_info, self.window)
        b2 = gtk.Button('Warning')
        b2.connect('clicked', self.do_warning, self.window)
        b3 = gtk.Button('Question')
        b3.connect('clicked', self.do_question, self.window)
        b4 = gtk.Button('Error')
        b4.connect('clicked', self.do_error, self.window)

        self.hbox.pack_start(b1)
        self.hbox.pack_start(b2)
        self.hbox.pack_start(b3)
        self.hbox.pack_start(b4)

        self.window.add(self.hbox)
        self.window.show_all()

    def do_info(self, widget, parent):
        d = gtk.MessageDialog(parent, gtk.DIALOG_MODAL,
                gtk.MESSAGE_INFO, gtk.BUTTONS_OK,
                'Hi!')

        img = gtk.Image()
        img.set_from_file('./smile.png')
        img.show()

        d.set_image(img)

        d.set_title('Smile :)')

        d.run()
        d.destroy()

    def do_warning(self, widget, parent):
        d = gtk.MessageDialog(parent, gtk.DIALOG_MODAL,
```

```
                gtk.MESSAGE_WARNING, gtk.BUTTONS_OK,
                'Warning')
        d.run()
        d.destroy()

    def do_question(self, widget, parent):
        d = gtk.MessageDialog(parent, gtk.DIALOG_MODAL,
                gtk.MESSAGE_QUESTION, gtk.BUTTONS_YES_NO)
        d.set_markup('Are you <b>ready</b>?')
        d.format_secondary_markup('We will take off <u>soon</u>.')

        result = d.run()
        if result == gtk.RESPONSE_YES:
            print 'Great!'
        else:
            print 'Hurry up!'

        d.destroy()

    def do_error(self, widget, parent):
        d = gtk.MessageDialog(parent, gtk.DIALOG_MODAL,
                gtk.MESSAGE_ERROR, gtk.BUTTONS_OK,
                'Unrecoverable error occured.')
        d.run()
        d.destroy()


app = Main()
gtk.main()
```

Lain-lain

Timeout

Untuk melakukan tindakan setiap interval tertentu, developer bisa

mempergunakan gobject.timeout_add().

Untuk menambahkan timeout, gunakan:

```
gobject.timeout_add(interval, callback, ...)
```

- interval: interval waktu dalam satuan mili detik.
- callback: fungsi yang akan dipanggil setiap interval.
- ...: argumen opsional yang akan dilewatkan ke callback.
- Mengembalikan integer ID event source

Definisi callback

Callback adalah fungsi biasa, namun dengan catatan berikut:

- Jumlah argumen harus sama dengan yang dilewatkan melalui gobject.timeout_add().
- Apabila callback ingin dipanggil lagi, maka callback harus mengembalikan True.

Menghapus timeout

Untuk menghapus timeout yang dibuat, sehingga callback tidak lagi dipanggil setiap interval tertentu, selain mengembalikan False pada callback, developer juga bisa menggunakan:

```
gobject.source_remove(tag)
```

- tag adalah ID event source yang dikembalikan oleh gobject.timeout_add()

timeout.py:

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

import pygtk
pygtk.require('2.0')
import gtk
import gobject
import time
```

```
class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(300,200)
        self.window.connect('destroy', gtk.main_quit)

        self.source_id = -1

        self.action = gtk.Button('START')
        self.action.connect('clicked', self.action_check)

        self.label = gtk.Label()
        self.vbox = gtk.VBox()

        self.vbox.pack_start(self.action)
        self.vbox.pack_start(self.label)

        self.window.add(self.vbox)
        self.window.show_all()

    def action_check(self, widget):
        button_label = widget.get_label()
        if button_label == 'START':
            button_label = 'STOP'
            self.source_id = gobject.timeout_add(1000, self.timer)
        elif button_label == 'STOP':
            button_label = 'START'
            gobject.source_remove(self.source_id)
        widget.set_label(button_label)

    def timer(self):
        now = time.asctime()
        self.label.set_label(now)
        return True
```

```
app = Main()
gtk.main()
```

Idle

Untuk melakukan suatu tindakan ketika PyGTK sedang idle, developer bisa mempergunakan gobject.idle_add().

Untuk menambahkan idle, gunakan:

```
gobject.idle_add(callback, ...)
```

- callback: fungsi yang akan dipanggil ketika idle
- ...: argumen opsional yang akan dilewatkan ke callback.
- Mengembalikan integer ID event source

Definisi callback

Callback adalah fungsi biasa, namun dengan catatan berikut:

- Jumlah argumen harus sama dengan yang dilewatkan melalui gobject.idle_add().
- Apabila callback ingin dipanggil lagi, maka callback harus mengembalikan True.

Menghapus idle

Untuk menghapus idle yang dibuat, sehingga callback tidak lagi dipanggil ketika idle, selain mengembalikan False pada callback, developer juga bisa menggunakan:

```
gobject.source_remove(tag)
```

- tag adalah ID event source yang dikembalikan oleh gobject.idle_add()

idle.py:

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

import pygtk
```

```
pygtk.require('2.0')
import gtk
import gobject

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(300,200)
        self.window.connect('destroy', gtk.main_quit)

        self.label = gtk.Label()

        gobject.timeout_add(100, self.timeout)
        gobject.idle_add(self.idle)

        self.window.add(self.label)
        self.window.show_all()

    def timeout(self):
        self.label.set_label('processing timeout')
        return True

    def idle(self):
        self.label.set_label('idle...')
        return True

app = Main()
gtk.main()
```

Events pending

Ketika berada dalam suatu tugas dengan waktu proses yang panjang, update ke komponen GUI yang dilakukan tidak akan berefek.

Sebagai contoh. Ketika suatu tombol ditekan, di dalam callback, perulangan

dari 0 sampai 999 akan dilakukan. Untuk setiap perulangan, counter akan ditampilkan pada gtk.Label, dimaksudkan sebagai update. Pada kenyataannya, yang terjadi begitu tombol ditekan adalah:

- Program akan freeze sebentar (mengulang 0 sampai 999).
- Pada akhirnya, gtk.Label akan berisikan angka 999.

Sementara, yang diinginkan adalah, setiap dari 0 sampai 999 tersebut akan ditampilkan pada label. Hal yang diinginkan tersebut tidak terjadi karena:

- Semua event GTK, termasuk update pada window dan komponen lain, ditangani di dalam mainloop.
- Ketika kode-kode di dalam callback dikerjakan, mainloop tidak dapat menangani event.

Oleh karena itu, apa yang perlu dilakukan developer adalah meminta GTK untuk memroses event-event yang pending ketika proses sedang dilakukan. Caranya, sisipkan kode berikut pada bagian kode yang mungkin memakan waktu lama ketika dikerjakan:

```
 while gtk.events_pending():
   gtk.main_iteration(False)
```

Sebagai catatan, ini akan efektif ketika suatu proses merupakan rangkaian dari banyak subproses lain.

Cobalah memodifikasi contoh eventpending.py sehingga pemrosesan event-event pending tidak dilakukan dan bandingkan hasilnya.

evenpending.py:

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

import pygtk
pygtk.require('2.0')
import gtk

class Main:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
```

```
        self.window.connect('destroy', gtk.main_quit)

        self.vbox = gtk.VBox()

        self.label = gtk.Label()
        self.button = gtk.Button('Make me busy')
        self.button.connect('clicked', self.do_busy, self.label)

        self.vbox.pack_start(self.label)
        self.vbox.pack_start(self.button)

        self.window.add(self.vbox)

        self.window.show_all()

    def do_busy(self, widget, label):
        for i in range(1000):

            while gtk.events_pending():
                gtk.main_iteration(False)

            label.set_label('%d' %(i))

app = Main()
gtk.main()
```

15. CGI

Header HTTP

a) Aplikasi web bekerja dalam HyperText Transfer Protocol (HTTP).

b) Dalam bekerja dengan HTTP, melibatkan penggunaan header-header HTTP.

c) Contoh header adalah:

   1. Content-Type: menentukan MIME type konten yang dikirim.
   2. Set-Cookie: mengirimkan cookie
   3. Location: redireksi ke lokasi tertentu
   4. Selengkapnya: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

d) Programmer yang bekerja dengan CGI perlu mengetahui header-header tersebut.

e) Header-header dicetak ke stdout.

f) Setelah HTTP header diberikan, kirimkan baris kosong 2 kali. Setelah itu, programmer bekerja dengan HTML.

CGI

a) CGI: Common Gateway Interface:

   1. CGI dapat dibangun dengan berbagai bahasa pemrograman dan digunakan pada berbagai sistem. CGI tidak mengharuskan penggunaan bahasa tertentu di sistem tertentu.
   2. CGI berfungsi sebagai pintu gerbang ke sistem-sistem lain.
   3. Protokol untuk menjalankan software eksternal dari web server, untuk menghadirkan dynamic content .

b) Pengembangan aplikasi CGI dengan Python:

   1. Pengguna Python hanya membutuhkan web server dan Python terinstall di sistem.
   2. Standard library Python dapat mempermudah pemrograman CGI.
   3. Pastikan setiap script (di Linux) telah diberikan hak akses executable (chmod +x <script.py>) sebelum diakses.

c) Komunikasi dengan web server melibatkan stdin, stdout dan environment variable:

   1. Dalam aplikasi umum, stdin umumnya keyboard dan stdout umumnya console/layar.
   2. Dalam CGI, data yang dikirim oleh user (diantaranya lewat web browser) akan dikirim lewat stdin. Output program CGI lewat stdout akan

dikirimkan ke user oleh web server.

3. Beberapa environment variabel akan diset dan dapat digunakan oleh developer.

Hello World

Hello World

Script hello.py berisikan contoh Hello World CGI.

hello.py:

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

print 'Content-type: text/html'
print

print 'Hello World'
```

Output:

- Ketika script diakses dari web browser, tulisan Hello World akan ditampilkan.

Penjelasan:

- Output akan menggunakan HTML, dengan MIME text/html. Header Content-Type digunakan.

Redireksi

Script location.py berisikan contoh redireksi ke URL lain dengan header Location.

location.py:

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

```
print 'Content-type: text/html'
print 'Location: test.py'
print
```

Output:

- Ketika script diakses dari web browser, halaman aktif akan berpindah ke test.py.

Penjelasan:

- Untuk redireksi, kita bisa mempergunakan header Location.

Traceback manager: modul cgitb

Modul cgitb dapat digunakan selama proses pengembangan untuk melaporkan exception yang terjadi. Penggunaan modul ini sangat membantu dalam pengembangan aplikasi CGI dengan Python.

Script cgierror.py berisikan contoh penggunaan modul cgitb.

cgierror.py:

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

import cgitb
cgitb.enable()

print 'Content-type: text/html'
print

print 1/0
```

Output:

- Ketika script diakses dari web browser, pesan kesalahan akibat pembagian dengan 0 akan ditampilkan, terformat rapi.

Penjelasan:

- Setelah import modul dilakukan, gunakan fungsi enable().

Catatan:

- Sebaiknya, jangan gunakan modul ini ketika aplikasi digunakan di lingkungan produktif. Informasi yang dilaporkan bisa mengekspos beberapa detil sistem.

Mendapatkan input

Dukungan CGI: modul cgi

Pengembangan aplikasi CGI dapat dilakukan dengan relatif lebih mudah dengan bantuan modul cgi. Dengan mempergunakan modul ini, kita bisa mendapatkan input dari user, termasuk upload file, dengan cepat, mudah, dan reliable.

Input single value

Script singlevar.py berisikan contoh mendapatkan input dari user.

singlevar.py:

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

import cgi
import os

print 'Content-type: text/html'
print

form = cgi.FieldStorage()

html_form = '''
<form action='singlevar.py'>
name: <input type='text' name='name'>
<input type='submit'>
</form>
'''
```

```
if not form.has_key('name'):
    print html_form
else:
    name = form.getfirst('name', '')
    name = cgi.escape(name)
    print 'Hi, %s' %(name)
```

Output:

- Ketika script diakses dari web browser, sebuah form akan ditampilkan. User dapat mengisi nama pada field name dan klik pada tombol Submit untuk mengirim nama yang diisikan.
- Setelah itu, server akan menampilkan pesan Hi, <nama>.

Penjelasan:

- Kita menggunakan pendekatan satu file untuk form dan prosesornya. Dengan demikian, untuk parameter action form, kita dapat mengisikan dengan nama file script kita.
- Pertama-tama, kita akan membuat objek form, yang merupakan instance dari class FieldStorage modul cgi. Objek form nantinya akan berisikan input dari user.

```
form = cgi.FieldStorage()
```

- Apabila form tidak memiliki key 'name', maka form input (variabel html_form) akan ditampilkan.
- Apabila form memiliki key 'name', maka name bisa didapatkan dengan method getfirst(). Dalam contoh ini, getfirst() akan berusaha mendapatkan variabel name, dan apabila tidak ditemukan, akan mengembalikan string kosong.

```
name = form.getfirst('name', '')
```

- Input dari user tidak bisa selalu dipercaya. Oleh karena itu, escaping dilakukan dengan fungsi escape() modul cgi. Proses escaping akan mengkonversi karakter "&", "<" dan ">" menjadi HTML sequence.

- Sebuah form kita siapkan dalam variabel html_form. Di dalam form

tersebut, kita mendefinisikan input:

- text, variabel: name (<input type='text' name='name'>)
- submit

Input multiple value

Script multivar.py berisikan contoh mendapatkan input dari user, dalam bentuk satu variabel, dengan multiple nilai yang diasosiasikan. Hal ini umumnya ditemukan dalam kasus dimana user bisa memilih lebih dari satu nilai, untuk satu field.

multivar.py:

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

import cgi
import os

print 'Content-type: text/html'
print

form = cgi.FieldStorage()

html_form = '''
<form action='multivar.py'>
Python version:<br>
<br><input type='checkbox' name='version' value='2.4'>2.4
<br><input type='checkbox' name='version' value='2.5'>2.5
<br><input type='checkbox' name='version' value='2.6'>2.6
<br><input type='submit'>
</form>
'''

if not form.has_key('version'):
    print html_form
else:
```

```
    version = form.getlist('version')
    print 'You are using Python version: '
    for i in version:
        print i
```

Output:

- Ketika script diakses dari web browser, sebuah form akan ditampilkan. Di form tersebut, user bisa memilih versi Python yang digunakan. Sebagai catatan, user bisa memilih lebih dari satu versi.
- Setelah user memilih dan klik tombol Submit, maka server akan menampilkan versi Python yang dipilih oleh user.

Penjelasan:

- Bacalah penjelasan sebelumnya, apabila diperlukan.
- Variabel version diasosiasikan pada tiga checkbox, dengan nilai (value) yang berbeda.
- Untuk mendapatkan multiple value untuk sebuah variable, gunakan method getlist().

Upload file

Script upload.py berisikan contoh upload file.

upload.py:

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

import cgi
import os

#windows check
try:
     import msvcrt
     msvcrt.setmode (0, os.O_BINARY) #stdin
     msvcrt.setmode (1, os.O_BINARY) #stdout
except ImportError:
```

```
     pass
#end of windows check

print 'Content-type: text/html'
print

form = cgi.FieldStorage()

html_form = '''
<form action='upload.py' method='post' enctype='multipart/form-data'>
Select file: <input type='file' name='myfile'>
<input type='submit'>
</form>
'''

if not form.has_key('myfile'):
    print html_form
else:
    myfile = form['myfile']
    if myfile.filename:
        fn = os.path.basename(myfile.filename)
        open('/tmp/' + fn, 'wb').write(myfile.file.read())
        print 'File %s uploaded successfully.' %(fn)
```

Output:

- Ketika script diakses dari web browser, sebuah form akan ditampilkan. Di form tersebut, user bisa memilih file yang akan diupload.
- Setelah user memilih dan klik tombol Submit, file akan diupload. Apabila sukses, maka file hasil upload akan disimpan dan pesan akan ditampilkan.

Penjelasan:

- Bacalah penjelasan sebelumnya, apabila diperlukan.
- Windows membutuhkan stdin dan stdout diset ke modus binary. Oleh karena itu, kita akan melakukan import msvcrt. Apabila berhasil, maka kita set modus untuk handle file bersangkutan.

```
#windows check
try:
      import msvcrt
      msvcrt.setmode (0, os.O_BINARY) #stdin
      msvcrt.setmode (1, os.O_BINARY) #stdout
except ImportError:
      pass
#end of windows check
```

- Form untuk file upload memiliki enctype dengan nilai multipart/form-data. Untuk form method, kita menggunakan post. Untuk upload file, kita menggunakan input dengan tipe file.
- Untuk mendapatkan file hasil upload, akseslah form dengan key adalah nama variable input file, yang merupakan instance FieldStorage.
  - Atribut filename adalah nama file yang dikirim user. Gunakan os.path.basename() untuk mendapatkan basename nama file tersebut.
  - Atribut file adalah objek file hasil upload. Simpan file hasil upload dengan membuka file baru (modus wb) dan menuliskan semua yang berhasil dibaca dari objek file upload.

Request method

Untuk mengetahui request method yang digunakan, kita bisa membaca environment variable REQUEST_METHOD, seperti dicontohkan dalam script reqmethod.py.

reqmethod.py:

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

import cgi
import os

print 'Content-type: text/html'
print

form = cgi.FieldStorage()
```

```
html_form = '''
<form action='reqmethod.py' method='post'>
name: <input type='text' name='name'>
<input type='submit'>
</form>
'''

if not form.has_key('name'):
    print html_form
else:
    name = form.getfirst('name', '')
    name = cgi.escape(name)
    print 'Hi, %s' %(name)
    print 'Request method: %s.' %(os.environ['REQUEST_METHOD'])
```

Output:

- Ketika script diakses dari web browser, sebuah form akan ditampilkan. User dapat mengisi nama pada field name dan klik pada tombol Submit untuk mengirim nama yang diisikan.
- Setelah itu, server akan menampilkan pesan Hi, <nama> dan diikuti pesan Request method: <method>.
- Akseslah os.environ untuk mendapatkan environment variable.

Catatan:

- Cobalah untuk mengakses reqmethod.py dengan query string name=test dan lihatlah hasilnya.

Cookies

HTTP adalah protokol stateless. Setelah user me-request dan server merespon, maka komunikasi selesai. Apabila developer ingin membangun kondisi dimana server mengetahui user sedang berada dalam sesi tertentu (misal: sedang login sebagai user tertentu, dalam proses transaksi pembelian barang, dan lain-lain), maka harus dikerjakan tersendiri, seperti dengan melibatkan penggunaan HTTP cookies.

HTTP Cookies, atau umum disebut cookies saja, adalah data yang dikirim oleh server, diterima dan disimpan oleh client (umumnya web browser) dan dikirim balik ke server ketika client mengakses server tersebut. Cookies dapat memiliki atribut tertentu, dan salah satunya adalah kapan suatu cookie expired.

Web browser modern umumnya mendukung cookies, dan menyediakan fasilitas untuk mendisablenya. Walau, secara default, web browser umumnya diset untuk menerima cookies.

Menggunakan Python, bekerja dengan cookies sangat dipermudah dengan modul Cookie.

Mengatur cookies

Contoh bekerja dengan cookies dan mengatur kapan suatu cookie berakhir dapat ditemukan pada script cookie1.py berikut.

cookie1.py:

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

import os, Cookie

cookie = Cookie.SimpleCookie()
cookie['user'] = 'test'
cookie['user']['expires'] = 10

print cookie
print 'Content-type: text/html'
print

cookie_str = os.environ.get ('HTTP_COOKIE')
if not cookie_str:
        print '<br>First time user (or Cookies disabled?)'
else:
        cookie.load (cookie_str)
        user = cookie['user'].value
        print '<br>User = %s' %(user)
```

Output:

- Ketika pertama kali cookie1.py diakses dari web browser, pesan 'First time user (or Cookies disabled?)' akan ditampilkan.
- Ketika kedua kali diakses, dalam jeda waktu di bawah 10 detik, maka pesan 'User = test' akan ditampilkan.
- Ketika akses berikutnya dilakukan, namun setelah 10 detik dari request sebelumnya, maka pesan 'First time user (or Cookies disabled?)' akan kembali ditampilkan. Hal ini disebabkan karena cookies yang kita set telah berakhir (expired).

Penjelasan:

- Untuk mengatur dan mengirimkan cookies:
  - Import modul Cookie
  - Instansiasi SimpleCookie (dari modul Cookie)
  - Setelah itu, kita bisa membuat cookie baru dengan memberikan key baru untuk objek SimpleCookie.
  - Setiap key baru diberikan, sebuah instance Morsel akan dibuat, yang mewakili setiap cookie baru yang dibuat. Untuk mendapatkan nilai cookie yang diset, atribut value dapat digunakan. Setiap cookie dapat memiliki key-key berikut:
    - comment: komentar
    - domain: domain dimana cookie valid. Awali dengan sebuah titik.
    - secure
    - expires: waktu dimana cookie berakhir setelah dikirimkan, dalam satuan detik. Secara default, berakhir setelah browser ditutup.
    - max-age
    - version
    - path: path dimana cookie valid. Secara default, berlaku pada path script tersebut. Set ke / agar valid untuk keseluruhan domain.
    - Baca juga: http://www.w3.org/Protocols/rfc2109/rfc2109
  - Cetak objek SimpleCookie

```
cookie = Cookie.SimpleCookie()
cookie['user'] = 'test'
cookie['user']['expires'] = 10
```

```
print cookie
```

- Untuk mendapatkan cookie (yang dikirim balik):
  - Baca environment variable (os.environ) HTTP_COOKIE
  - Bangun struktur data dengan method load() objek SimpleCookie
  - Akses atribut value cookie untuk mendapatkan nilai cookie

```
cookie_str = os.environ.get ('HTTP_COOKIE')
if not cookie_str:
        print '<br>First time user (or Cookies disabled?)'
else:
        cookie.load (cookie_str)
        user = cookie['user'].value
        print '<br>User = %s' %(user)
```

Mendeteksi cookies

Cookies bisa didisable oleh user, lewat pengaturan preferensi web browser. Untuk mendeteksi apakah cookies diaktifkan atau tidak, lakukanlah langkah-langkah berikut:

1. kirimkan cookies
2. redireksi/refresh halaman
3. periksa apakah cookies tersedia

Langkah-langkah ini diperlukan karena: pada pertama kali request, server hanya mengirimkan cookies, namun user tidak mengirimkan balik (sehingga tidak ada cookies). Pada saat redireksi dilakukan, user melakukan request (dan oleh karenanya, mengirimkan balik cookies). Pada request kedua kali ini, keberadaan cookies bisa terdeteksi.

Script detectcookie.py berisikan contoh pendeteksian cookies.

detectcookie.py:

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

```
import Cookie
import os
import cgi

cookie = Cookie.SimpleCookie()
cookie['user'] = 'test'
cookie['user']['expires'] = 10

use_cookie = False
start_check = False

form = cgi.FieldStorage()
checkcookie = form.getfirst ('checkcookie')
if checkcookie:
    cookie_str = os.environ.get ('HTTP_COOKIE')
    if cookie_str:
        use_cookie = True
else:
    start_check = True

if start_check == True:
    print 'Location: detectcookie.py?checkcookie=1'



print cookie
print 'Content-type: text/html'
print

if use_cookie == True:
    print 'Cookie is ENABLED.'
else:
    print 'Cookie is DISABLED.'
```

Output:

– Ketika script diakses dari web browser, pesan 'Cookie is ENABLED' atau

'Cookie is DISABLED' akan ditampilkan, sesuai dengan pengaturan cookies di web browser.

Penjelasan:

- Pada akhirnya, variabel use_cookie akan bernilai True atau False. True berarti cookies diaktifkan. Pertama-tama, nilai use_cookies diset False.
- Variabel start_check akan menentukan apakah redireksi akan dilakukan atau tidak. Apabila bernilai True, maka redireksi dilakukan (dengan header Location). Pertama-tama, nilai start_check diset False.
- Script akan berusaha mendapatkan variabel checkcookie dari query string. Apabila didapatkan, maka cookies akan dicek. Apabila tidak didapatkan, maka redireksi dilakukan.

Session

Session dapat diartikan sebagai cookies, namun yang disimpan di server. Data yang dikirimkan antara server dan client umumnya berupa Session ID. Di server, session ID tersebut umumnya menjadi bagian dari nama file, yang digunakan untuk menyimpan informasi session.

Session dapat diimplementasikan dengan cookies. Apabila cookies tidak diaktifkan, session bisa pula diimplementasikan dengan Query string ataupun dengan field hidden. Salah satu kelemahan implementasi dengan query string dan field hidden adalah session ID harus selalu dimaintain, baik di query string, ataupun di field hidden. Sementara, salah satu kelemahan session dengan cookies adalah tidak dapat bekerja ketika cookies di-disable. Sebagian developer menggunakan pendekatan untuk menggunakan salah satu dari query string atau field hidden apabila cookies di-disable.

Training ini hanya memfokuskan pada session yang diimplementasikan dengan Cookies.

Terdapat beberapa hal yang harus dilakukan ketika mengimplementasikan session:

- session ID: definisikan nama variabel session ID dan nilai yang mungkin. Nama variabel bebas. Sebagai contoh adalah SID, atau SESSIONID. Sementara, nilai yang mungkin, haruslah unik. Modul md5 dan sha bisa dipergunakan untuk hash time. Apabila diperlukan, kombinasikan dengan modul random, ataupun dengan teknik lainnya.
- Format file data session. Data session (cookies dan lainnya) perlu disimpan di filesistem server. Developer bisa mempergunakan format yang diinginkan.
- Mekanisme penggunaan session.