Python Dasar

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) Gunakan dua baris kosong antara class.

e) Gunakan satu baris kosong antara fungsi.

f) Import setiap modul dilakukan per baris, dengan urutan: standard library, pustaka pihak ketiga, pustaka lokal.

g) Struktur program Python:

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

h) 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: lowercase atau UPPERCASE, setiap kata dipisahkan underscore.

i) Case sensitive!

2. Sekilas tentang Python

a) Open Source
   1. Sesuai dengan sertifikasi Open Source Initiative.
   2. Kompatibel dengan GPL, menurut Free Software Foundation. Walau demikian, tidak ada pembatasan copyleft GPL.
   3. Bebas digunakan, termasuk untuk produk proprietary.
   4. Selengkapnya, http://www.python.org/psf/license/.

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 sangat kaya.

f) Berjalan di sangat banyak sistem.

g) Dapat diextend dengan C/C++

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. Contoh akan dibahas dalam sesi training.

e) Untuk membaca string dari standard input, raw_input() bisa digunakan. Contoh akan dibahas dalam sesi training.

f) Beberapa fungsi lain:

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

Catatan untuk instruktur:

- Di dalam sesi interactive, instruktur akan menjelaskan berbagai contoh penggunaan, termasuk menggunakan python sebagai kalkulator.
- Demo tipe data dan modul bisa dilakukan.
- Demo fungsi lain bisa dilakukan, apabila memungkinkan.

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 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)
	  • 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
		• Memiliki sifat iterable

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
  - 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)
- 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

File	• File (file)
	  • Dibuka dengan open(name[, mode[, 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	

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. Seleksi/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.
- Beberapa contoh akan diberikan pada sesi training.
- 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.
- Beberapa contoh akan diberikan pada sesi training.

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.

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.
- Untuk merujuk ke anggota 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).

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...
>>>
```

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
```

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
- sha: SHA-1 message digest
- binascii: konversi binary-ascii
- csv: baca tulis file csv
- glob: pathname pattern expansion
- linecache: akses ke baris tertentu dalam file
- md5: MD5 message digest

- 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

Compiled module

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

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
```

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

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().

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']
>>>
```